Lecture

Organic Computing II

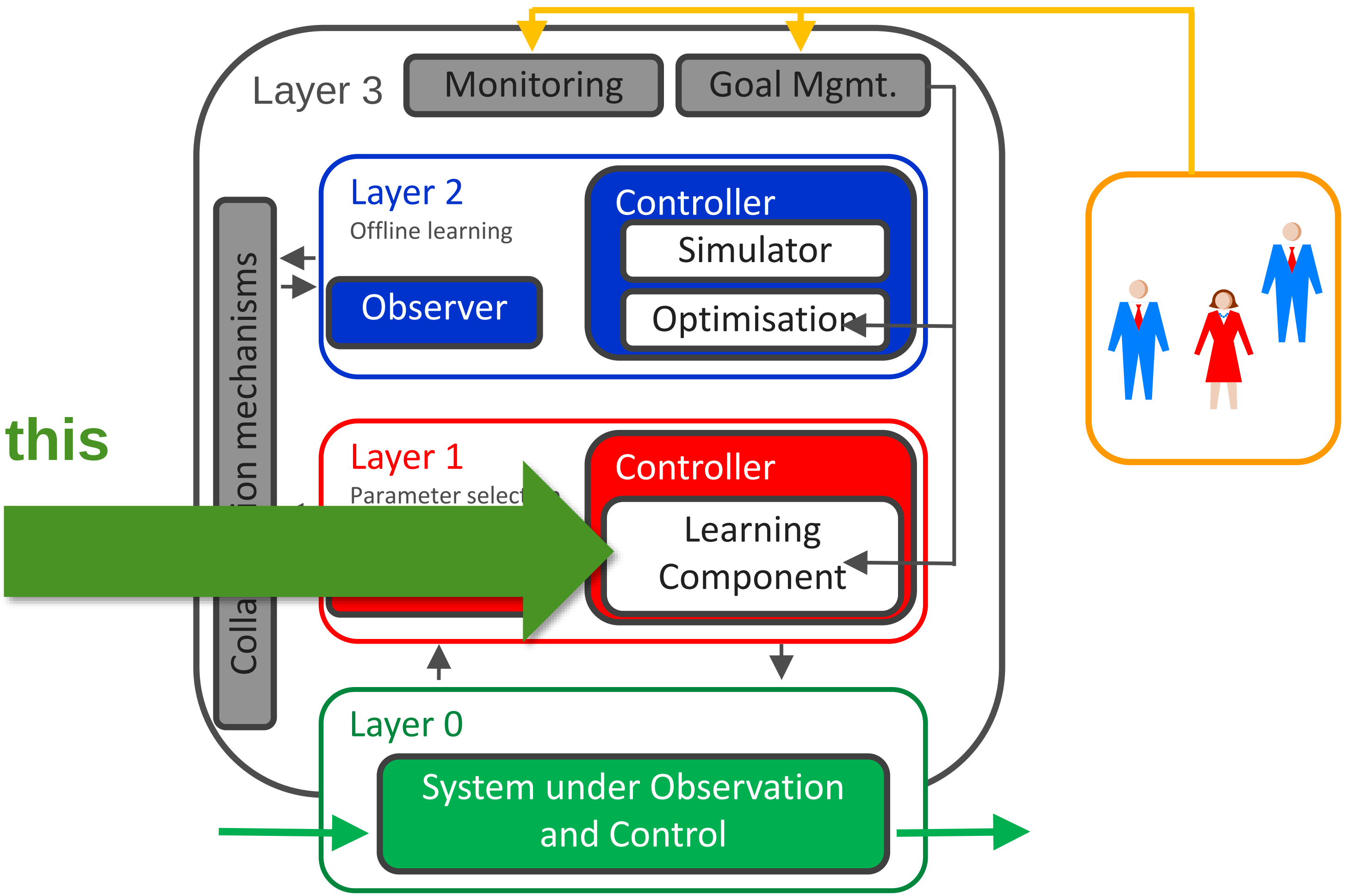
Summer term 2019

Chapter 6: Learning

Lecturer: Anthony Stein, M.Sc.

Slides have been created in cooperation with Prof. Dr. Sven Tomforde (Uni Kassel),
Prof. Dr. Müller-Schloer (Uni Hannover) and Prof. Dr. Jörg Hähner (Uni Augsburg)

Focus of this Lecture!



Agenda

Content

- Motivation
- Extended Classifier System
- XCS-O/C
- Artificial Neural Networks
- Conclusion and further readings

Goals

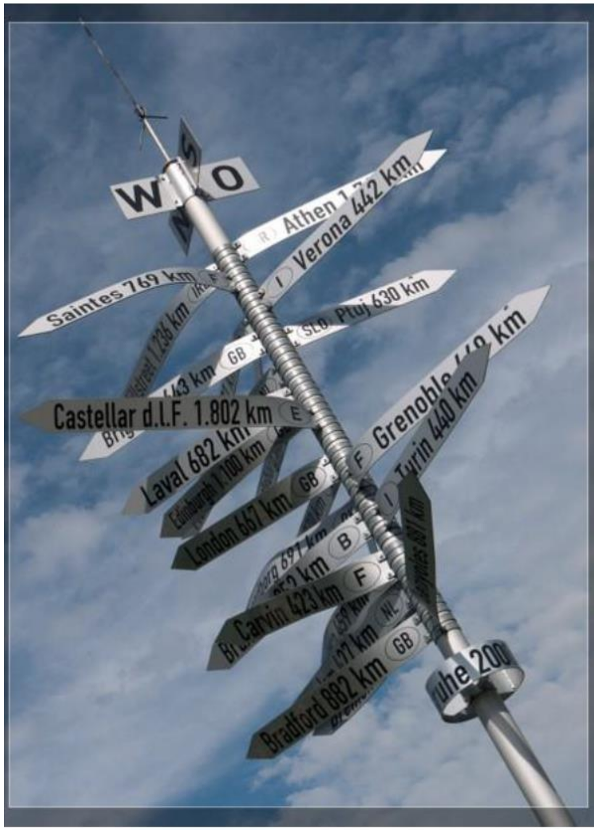
Students should be able to:

- Explain what machine learning is and why it is needed in organic systems.
- Compare basic concepts such as supervised vs. reinforcement learning or exploration vs. exploitation.
- Outline an XCS and explain the main loop with all components.
- Discuss the necessary modifications to XCS for OC.
- Explain basic Feed-Forward ANNs and how backpropagation works

Agenda

ORGANIC
COMPUTING

- **Motivation**
- Extended Classifier System
- XCS-O/C
- Artificial Neural Networks
- Conclusion and further readings



Learning in organic systems

Complex learning tasks:

- Sparse and imbalanced data
  - E.g. due to non-uniform distributions and class imbalances
- Non-stationary environments
  - May exhibit severe changes in the target concepts.
  - Also called *concept drift*.
- Necessity of exploration boundaries
  - Unrestricted or unknown feature spaces
  - Continuous or large discrete action spaces
  - Legal constraints
    → Trial-and-error must be avoided!
- Complexity of underlying problem space
  - Functions mapping inputs to certain outputs regarding are complex.
  - E.g. due to their dimensionality, continuity, obliqueness and curvature.
- Knowledge and expected behaviour must be represented in a human comprehensible manner (e.g. as rules).

Motivation

- Machine learning techniques seem to be a promising approach for continuous self-improvement in Organic Computing systems.
  → How can computers be programmed so that problem solving capabilities are built up by specifying "what is to be done" rather than "how to do it"? (Holland, 1975).

- Major issues:
  - How can the system react to unforeseen situations?
  - How can the system automatically improve its performance (if possible) at runtime?
  - How can knowledge (and expected behaviour) be encoded in a human comprehensible manner?
  - Overall: flexible and autonomous reaction to changes of the environments and/or the system itself are desirable.

Motivation (2)

- Machine learning is used when:
  - human expertise does not exist (navigating on Mars)
  - humans are unable to explain their expertise (speech recognition)
  - solution changes in time (routing on a computer network)
  - solution needs to be adapted to particular cases (user biometrics)
  - human learning is not feasible (map protein sequences to secondary structures)
  - …

- The slides for the introduction part to Machine Learning are mainly based on the book:

Alpaydin, E. (2004). Introduction to machine learning. MIT press.

Definition of learning (2)

- Common definition:

  "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 tasks in T, as measured by P, improves with experience E."

- *Example:*
  - consider a program that learns playing checker
  - task *T*: play checker
  - performance measure *P*: percentage of the games won
  - experience *E*: play against itself

Mitchell, T. M. (1997). Machine learning. 1997. Burr Ridge, IL: McGraw Hill, 45.

The inductive learning problem

- consider a given set of examples (training set)
- make accurate predictions about future examples (validation set)
- with the goal of predicting concrete values for unknown feature sets, the learning problem can be mapped to learning a function

  $f(X) = Y$, whereas X is an input example and Y the desired output

Supervised Learning

Supervised Learning

training = desired (target) output

error = (target output – actual system output)

ORGANIC COMPUTING

- Supervised learning means that the training set of (X, Y) is provided (by a ``supervisor'' or ``teacher'')
- Unsupervised learning means only the input data X and feedback on the prediction performance are provided
- Example supervised learning: Face recognition
  - x: Bitmap picture of person's face
  - *f*(x): Name of the person
- Example unsupervised learning: Customer segmentation
  - x: Customer's buying habits, address, age,…
  - *f*(x): Customer class

Expected learning results

Supervised learning

1. Prediction of future cases:
   Use the learned model (or *hypothesis*) to predict the output for future input
2. Knowledge extraction:
   The hypothesis is easy to understand
3. Compression:
   The hypothesis is simpler than the data it explains
4. Outlier detection:
   Exceptions that are not covered by the hypothesis (e.g. fraud)

Unsupervised learning

1. Clustering of data
2. Density estimation of data
   ➞ gain insights into the organisation of data

Reinforcement Learning

training information = evaluation (“rewards” / “penalties”)

Goal: achieve as much reward as possible!

- Act “successfully“ in the environment
- Implication: maximise the sequence of rewards R_t

- What is Reinforcement Learning?
  - German: “Bestärkendes Lernen“
  - Learning from interaction
  - Goal-oriented learning
  - Learning **by/from/during** interaction with an external environment
  - Learning “what to do“ (how to map situations to actions) to maximise a numeric reward

Reinforcement learning

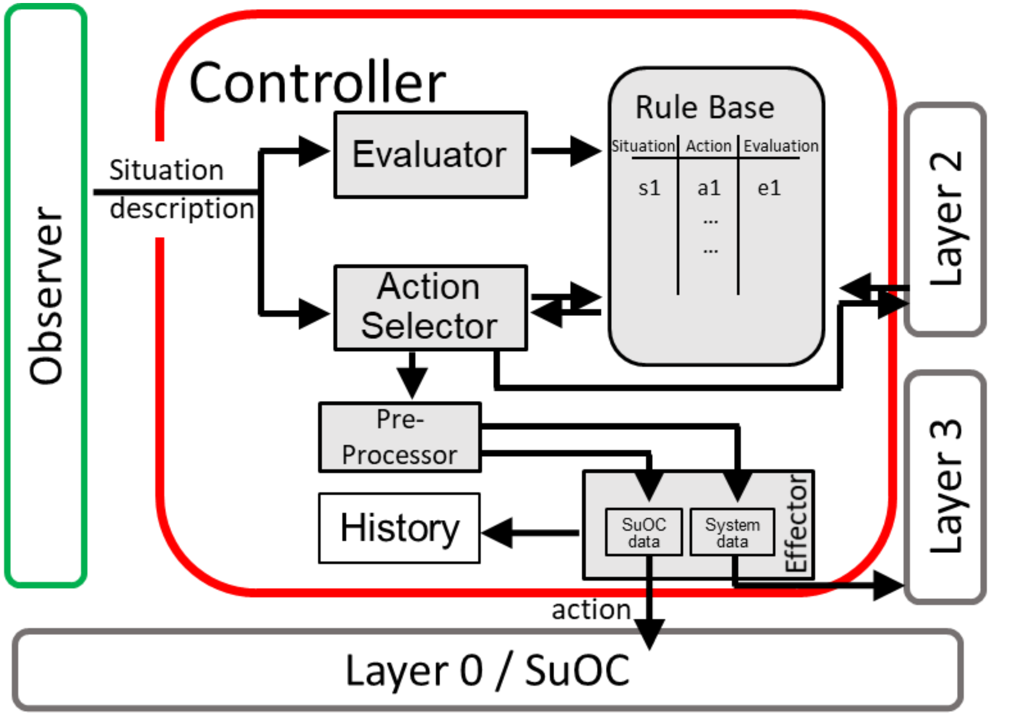
- Learning based on feedback
  - feedback tells the system how well it performs (**not** what it should be doing)
  - no supervised output; but a delayed “reward”
- Aspects:
  - credit assignment problem
  - game playing
  - robot in a maze
  - multiple agents, partial observability, ...

This is exactly what we will have to cope with in most OC applications!

Recap: Layer 1 Controller

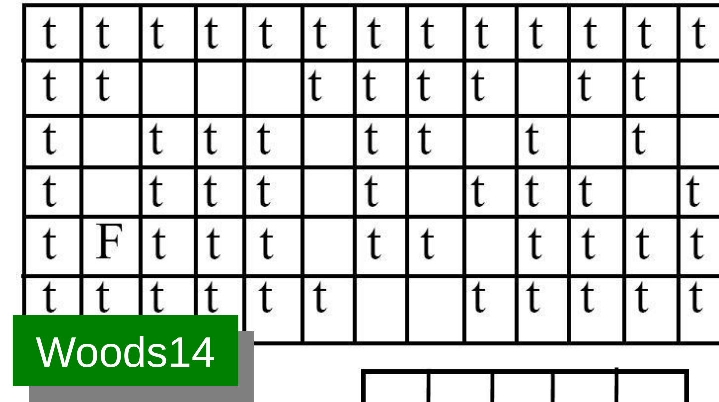
Observer/Controller

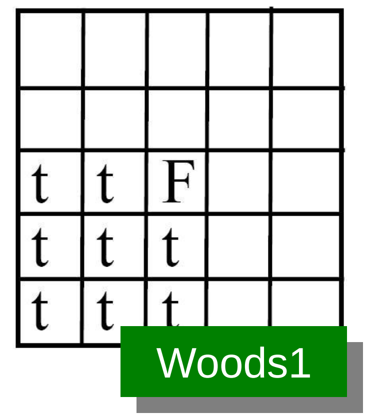
- Controller has to learn from feedback.
- Basic concept: rule-based system
- Learning is done by „book-keeping“ attributes, i.e. evaluation parameters.
- These are modified depending on the observed success.



Example: ‘Woods’ scenario / Maze

- Example of an Animat problem
- Basis: rectangular toroidal regular (n x m)-grid
- Each grid cell may contain a tree (t), food (F), or it may be empty.
- Food and trees fixed per instance
- Animat/agent/robot is initially randomly placed on empty cell.
- Walks around, looking for food
- In each step, agent can go to one of the eight neighboring cells (empty and food cells only).



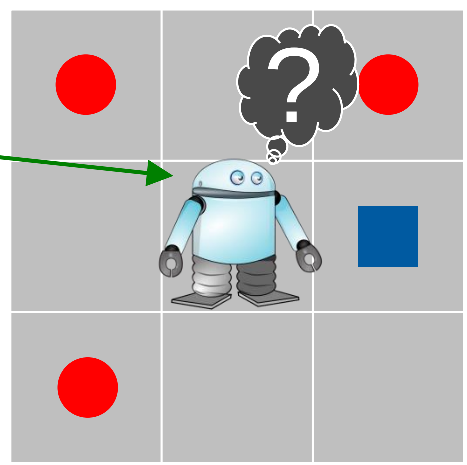


Woods1: optimal average number of steps to reach food: 1.7 steps (Bull & Hurst, “ZCS redux”)

Example: 'Woods' and the underlying problem

- Question: Can we build an agent that can efficiently find food "in the Woods" without global knowledge?
- One idea to build such an agent:
  - Suppose the agent can "see" the eight surrounding cells.
  - Based upon this perception, it has to decide where to go next.
  - Reward is paid once the food is found.

Agent
- Observation: 8 cells
- Status



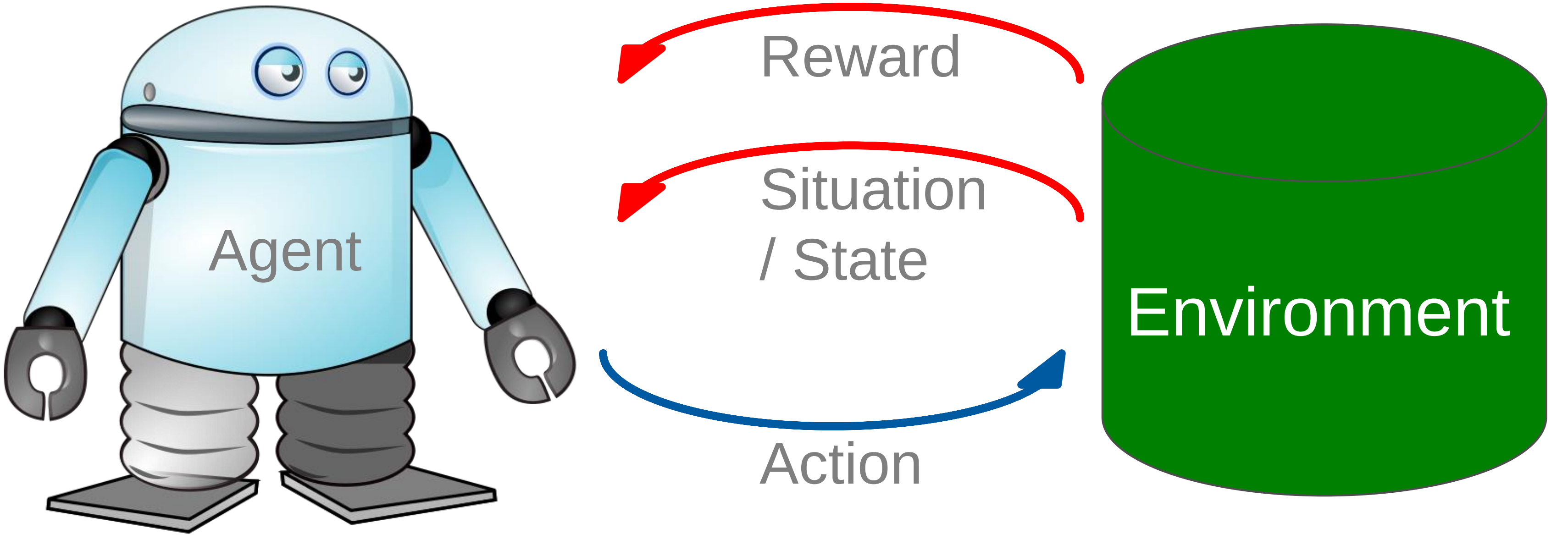
Obstacle

Previous cell

Where to go?

Agent model

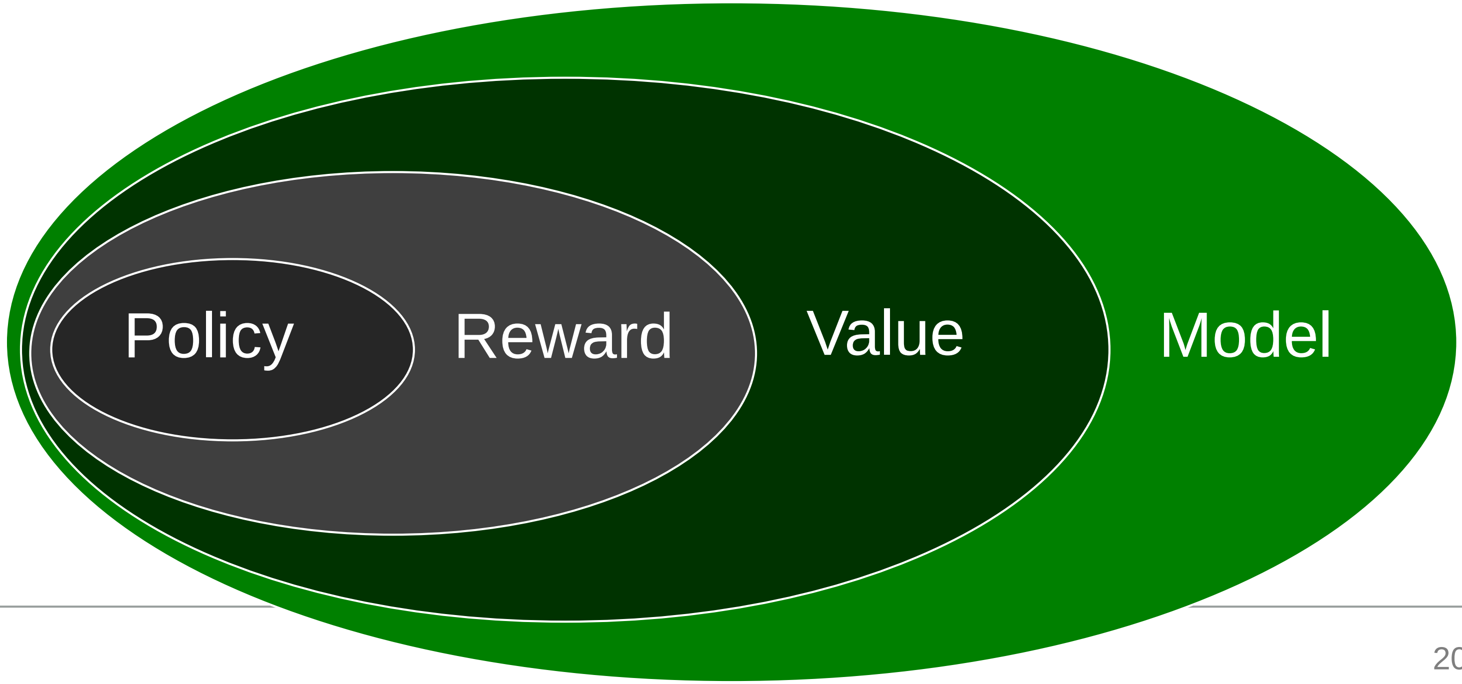
- The complete agent
  - Chronologically situated
  - Constant learning and planning
  - Affects the environment
  - Environment is stochastic and uncertain



ORGANIC COMPUTING

Elements

- Policy: What to do in a particular situation?
- Reward: What is good or bad behaviour (experience)?
- Value: What is a good action due to the expected reward?
- Model: What follows from the actions? What is the impact?



Exploration vs. exploitation

Exploration:
A process of visiting entirely new regions of a search space.

vs.

Exploitation:
A process of visiting regions of a search space based on previously visited points (neighbourhood).

To be successful, a search algorithm needs to find a good balance between exploration and exploitation.

- Exploration is important in early stages:
  - seek good patterns
  - spread out through the search space
  - avoid local optima
- Exploitation is important in later stages:
  - exploit good patterns
  - focus on good areas of the search space
  - refine to global optimum

Exploration vs. exploitation (2)

The Exploration / Exploitation Problem: Formalisation

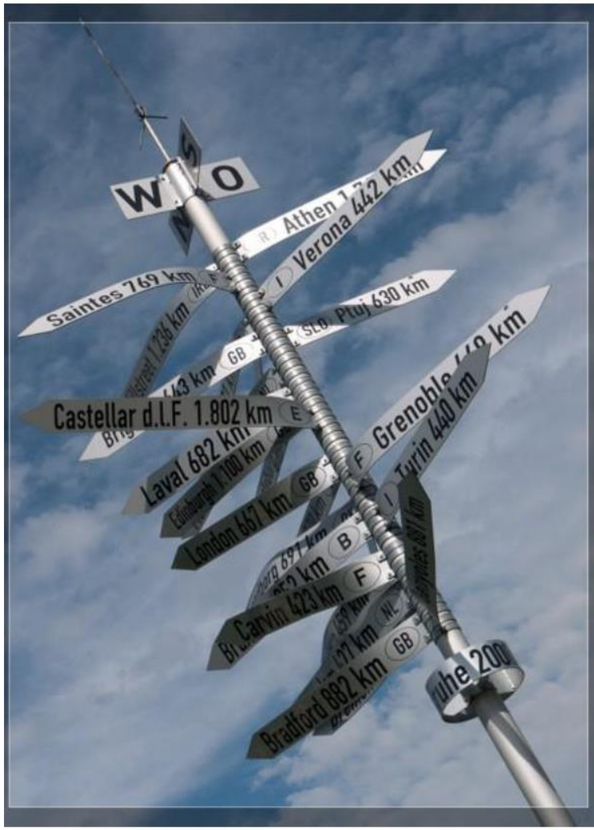
- Suppose values are estimated:
  $Q_t(a) \approx Q^*(a)$; estimation of action values
- The greedy-action for time t is:

$$a_t^* = \arg max_a \, Q_t(a)$$
$$a_t = a_t^* \Rightarrow exploitation$$
$$a_t \neq a_t^* \Rightarrow exploration$$

- Insights:
  - You cannot explore all the time, but also not exploit all the time.
  - Exploration should never be stopped, but it should be reduced.

Agenda

- Motivation
- **Extended Classifier System**
- XCS-O/C
- Artificial Neural Networks
- Conclusion and further readings



Names to remember in LCS research

ORGANIC COMPUTING

- Initial Learning Classifier System (LCS) was introduced by John H. Holland in 1975.
- He was (and still is) interested in complex adaptive systems.
- How can computers be programmed so that problem-solving capabilities are built up by specifying “what is to be done” rather than “how to do it”? (Holland, 1975)
- An important development in LCS was done by Stewart W. Wilson in 1995.
- Based on the initial approach by Holland, Wilson proposed a simplified and more efficient classifier system called Extended Classifier System (XCS).
- XCS is today one of the most studied classifier systems.
- Many extensions have been proposed.

Initial approach by Holland

- Initially system
  - Holland designed a first system in 1978.
  - System is called CS1.
- System contains
  - Set of classifiers (condition/action)-pairs
    → Not called “rule” since they compete (classifier is a “may rule”)!
  - Input interface to receive state from the environment
  - Output interface to apply actions to the environment
  - Internal message list as an internal “workspace” for I/O
  - Evolutionary process (genetic algorithm) to generate new classifiers

The Extended Classifier System (XCS) by Wilson

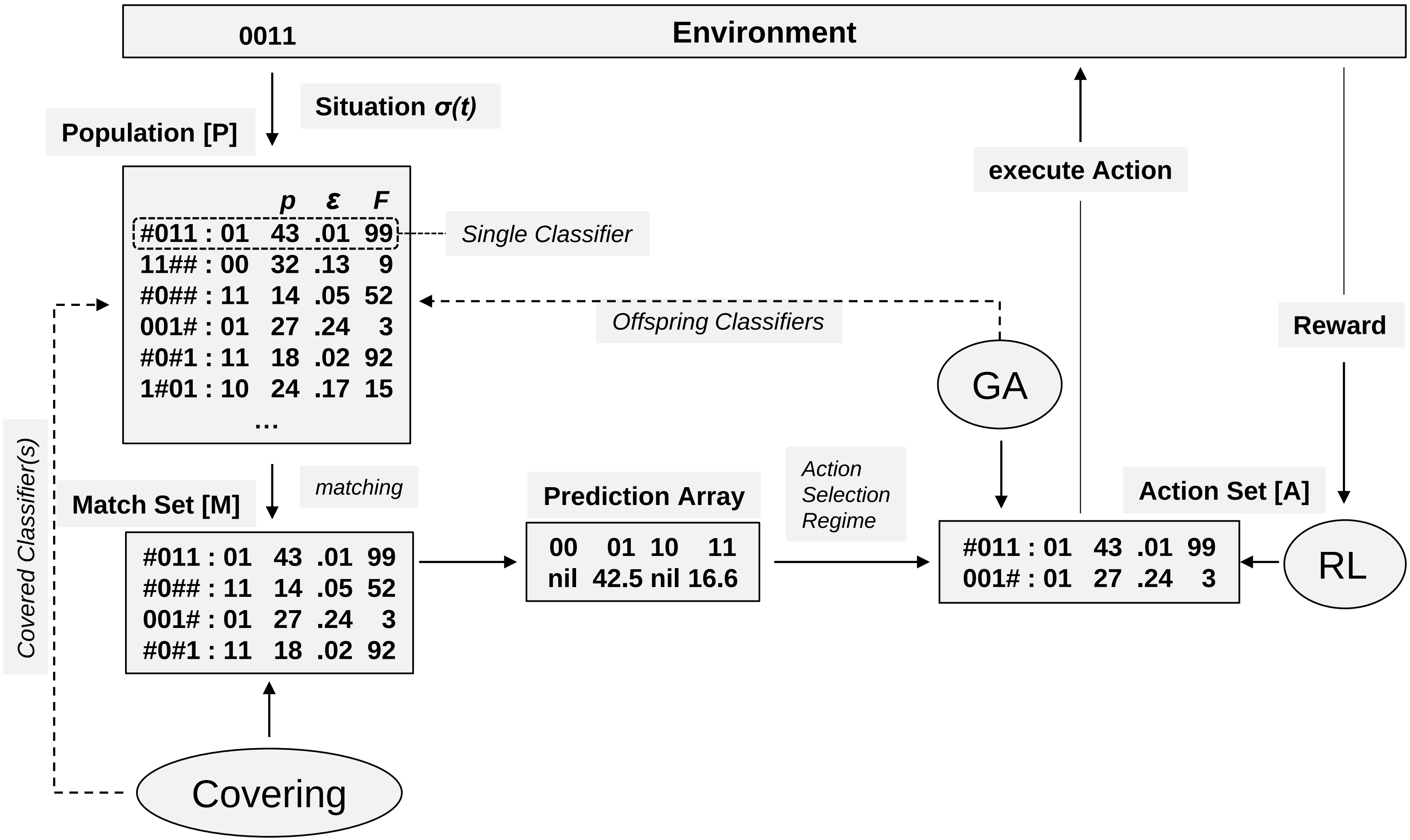
- XCS is a rule-based (online) learning system.
- It can be used for pure classification as well as for regression problems.
- It is a derivative of the overall class of *Learning Classifier Systems (LCS)*, initially proposed by Holland in 1978 (CS-1)
- Wilson in 1994 simplified Hollands CS-1 to the so-called *Zeroth-Classifier System (ZCS).*
- In 1995 Wilson presented the *Extended Classifier System (XCS).*
- Initially designed for binary problems, Wilson further extended XCS toward the ability to cope with real-valued inputs (XCSR) in 2001.

Why `Classifier‘ system?

- XCS stores rules (termed `classifiers‘) in a limited set of max. N classifiers called population $[P]$.

- A single classifier cl is comprised of:
  - A condition C that defines a subspace of the input space X
  - An action a that determines a reaction executed on the environment (e.g. `0‘ and `1‘ for `turn left‘ or `turn right‘)
  - A predicted payoff scalar p which is an estimate of the expected reward when the action a of this classifier is selected for execution
  - An absolute error of the payoff prediction ϵ
  - A measure of accuracy termed fitness F which is some sort of inverse function of ϵ
  - Some more so-called `book-keeping‘ parameters (e.g. experience)

One cycle through XCS



A single iteration through the main loop

1. At each timestep t, XCS retrieves a situation $\sigma(t)$ from the observed environment.

2. XCS scans $[P]$ for matching classifiers and builds a so-called match set $[M]$.

3. Among all matching classifiers, the `prediction array‘ PA calculates the most promising action a.

4. All classifiers from $[M]$ with the selected action a, from another subset $[A]$ called the action set.

5. The selected action a_{exec} is actualised on the environment which in turn delivers a so-called payoff or reward r.

6. r is used to updated and refine all classifiers in $[A]$, since these particular classifiers advocated the same action as the one executed.

ORGANIC COMPUTING

Why such a triple ranking by p, ε, and F?

- What is the difference of a classifier's strength and its accuracy?
  - Strength = predicted payoff p
  - Accuracy = Fitness (inverse of prediction error ϵ)
- Is a classifier predicting a high payoff also an accurate one?
  - When a classifier predicts a high payoff, this does not necessarily mean that its prediction is correct!
- Is it beneficial to know low performing (regarding p) but highly accurate (F) classifiers?
  - Yes, indeed!
  - The system has an indicator which action delivers low payoff, and thus will decide more likely against this action.

Wilson's "Generalisation Hypothesis"

- Wilson hypothesised that XCS constructs classifiers that are maximally general and accurate at the same time.
- Thus, XCS attempts to construct a map/approximation of the underlying payoff-landscape, that is $X \times A \rightarrow P$, by means of single classifiers:
  - X is the input space (possible input)
  - A is the action space (possible outputs)
  - P is the payoff space (possible rewards)
- This map/approximation shall be:
  - *Complete*, in the sense that the entire payoff landscape is covered.
  - *Compact,* in terms of the # physical classifiers (macro-classifiers).
  - *Accurate, since* the system error shall be as minimal as possible (of course).
  - *Maximally general, since* the shape of a classifier (determined by its condition) shall be large enough to cover the environmental niche within X but specific enough to remain accurate.

Wilson's "Generalisation Hypothesis" (2)

- The separation of strength and accuracy combined with the incorporated `niche' genetic algorithm exerts evolutionary pressure toward the aforementioned properties.

- The GA favours accurate (high fitness) classifiers within the environmental niche.

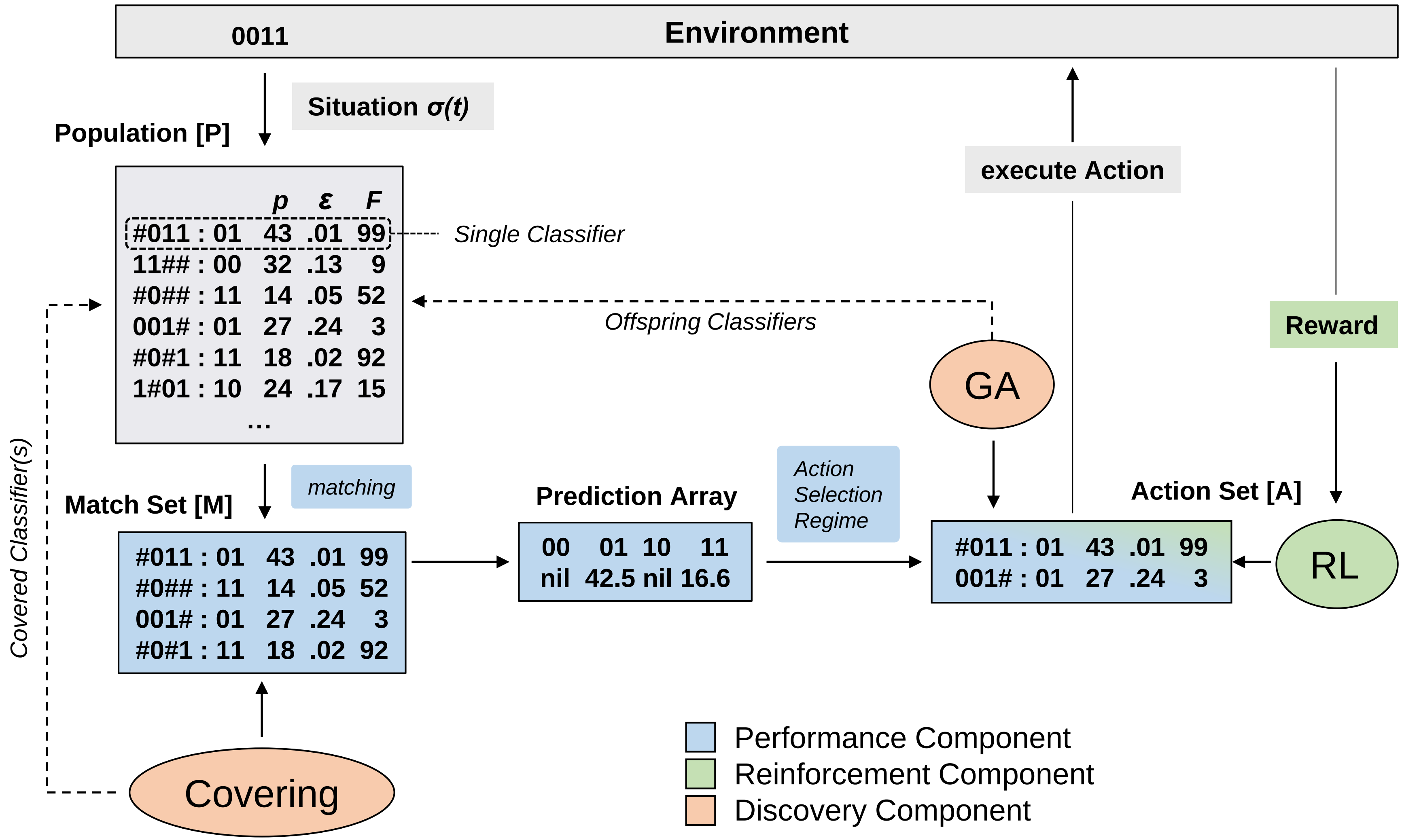
- Thus, accurate classifiers are more likely to reproduce and will eventually take over the environmental niche.

XCS‘ algorithmic structure

XCS‘ three main components

- Performance component
  - Matching, Payoff Prediction, Action Selection
- Reinforcement component
  - Attribute update, deferred credit assignment
- Discovery component
  - Covering of non-explored niches, refinement of poorly explored niches

XCS‘ algorithmic structure (2)



XCS‘ algorithmic structure (2)

Matching

- At each time step t XCS retrieves a binary string on length n
- This string is denoted as $\sigma(t) \in \{0,1\}^n$
- Example for $n = 6$ and $t = 1$: $\sigma(1) = 011001$
- Each classifier maintains a condition or schema C.
- The conditions are encoded ternary, i.e. $C \in \{0,1,\#\}^n$.
- The # symbol serves as wildcard or `don‘t care‘ operator.
- Examples of conditions: (is matching $\sigma(1)$?)
  - 0#1001 (yes)
  - #01001 (no)
  - 011##1 (yes)

Matching is the process of scanning the entire population [P] for classifiers with a condition that fits the situation $\sigma(t)$

XCS' algorithmic structure (3)

The system prediction

- The system prediction $P(a)$ is a fitness-weighted sum of predictions of all classifiers advocating action a

$$P(a) = \frac{\sum_{cl \in [M] | cl.a = a} cl.F * cl.p}{\sum_{cl \in [M] | cl.a = a} cl.F}$$

- Especially at this place, the separation of strength and accuracy plays a major role!

- For each possible action $a \in A$ there exists one entry within the PA.
  → There may be several classifiers supporting the same action.

XCS‘ algorithmic structure (4)

Update rules:

- $\epsilon_j = \epsilon_j + \beta\left(\left|P - p_j\right| - \epsilon_j\right)$
- $p_j = p_j + \beta(P - p_j)$
- $F_j = F_j + \beta\left(k'_j - F_j\right),\ k'_j = \frac{k_j}{\sum_{cl_i \in [A]} cl_i.k},\ k_j = \alpha\left(\frac{\epsilon_j}{\epsilon_0}\right)^{-v}$

- β is the learning rate (typically set to 0.2)
- α (often set to 0.1) and ν (usually set to 5) control how strong accuracy decreases when the error is higher than ϵ_0
- ϵ_0 defines the targeted error level of the system
- In single-step problems: P is set to the reward r_{imm}
- Classifier attributes are updated by means of the modified delta rule (Widrow-Hoff delta rule) in combination with the moyenne adaptiv modifee (MAM) technique.

ORGANIC COMPUTING

Covering

- Covering is the process of generating a novel classifier that matches the current input whenever:
  - Match set $[M]$ is empty (i.e. no matching cl in [P]).
  - $[M]$ is poor, i.e. average fitness below a certain threshold.
  - $[M]$ contains less then θ_{mna} distinct actions.
- The condition of the covered classifier cl_{cov} is set to the current input.
- Additionally, each bit is replaced by a # (for generalisation purposes) with probability $P_{\#}$.
- Values for p, ϵ and F are set to predefined initial values (typically 10.0, 0.0 and 0.01).

XCS‘ algorithmic structure (6)

Genetic Algorithm:

- One of the most essential part of XCS is the incorporated niche Genetic Algorithm (GA).
- It is triggered when the average time of all classifiers in $[A]$ since the last GA invocation is greater than θ_{GA} (often set to 50).
- The GA selects two parents from [A] with a probability proportional to their fitness values (roulette-wheel selection).
  - The higher a classifier‘s fitness, the higher the selection chance.
- The selected parents are copied to generate two offspring classifiers cl_{off}.

ORGANIC COMPUTING

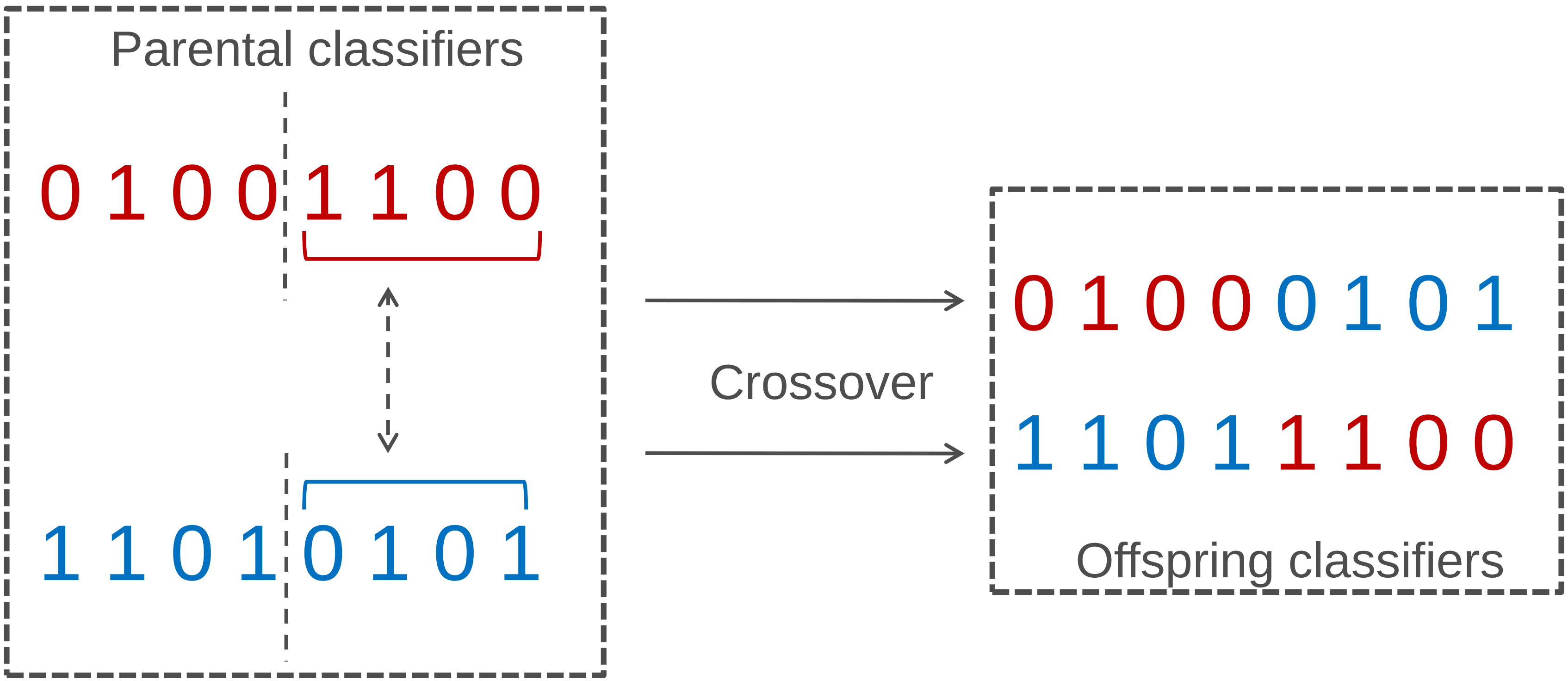
Genetic operators

- The conditions of both cl_{off} are crossed (crossover operator):
  - One-point crossover: Each offspring classifier‘s condition is split at a certain point and switched with the other offspring classifier.
  - n-point crossover: more than one point is determined for switching.
  - Uniform crossover: Each value is switched with probability $P_\chi = 0.8$.

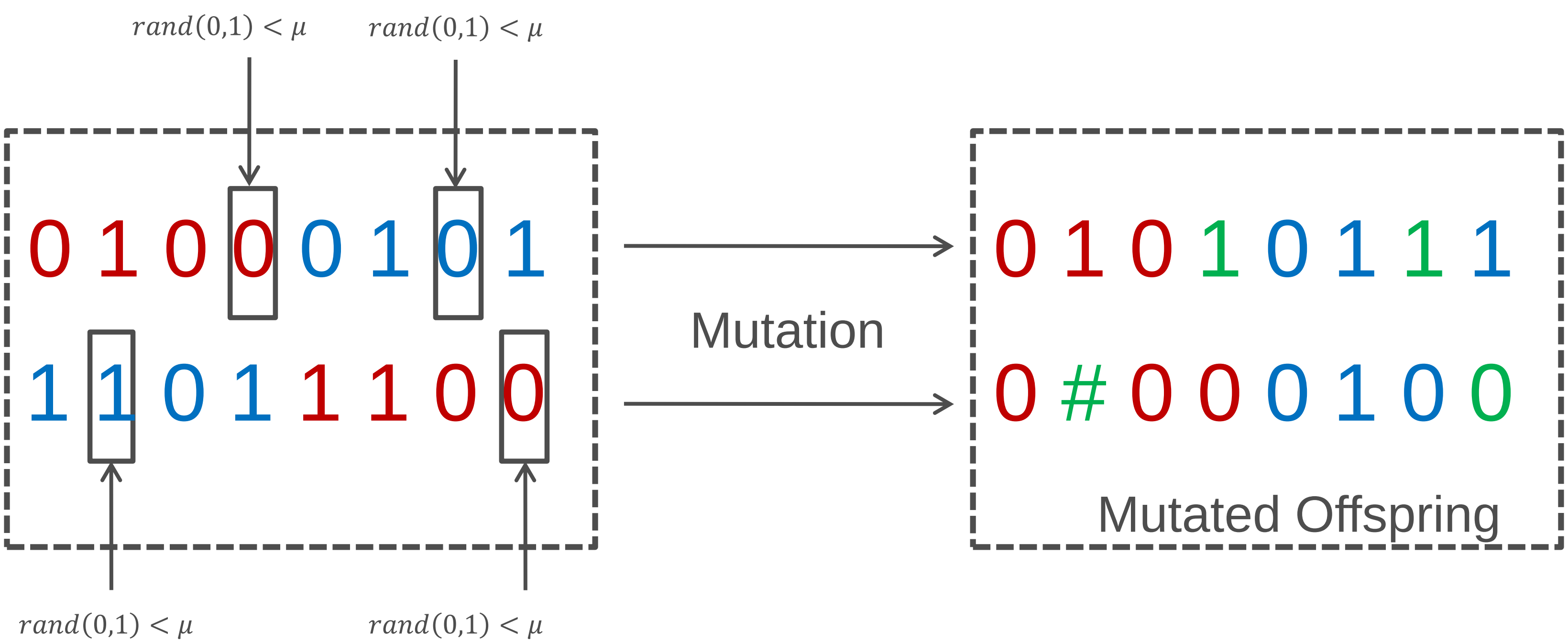
- Afterward, each bit is flipped with probability $P_\mu = 0.04$ to one of the other allowed alleles, that is $\{0,1,\#\}$.

XCS‘ algorithmic structure (8)

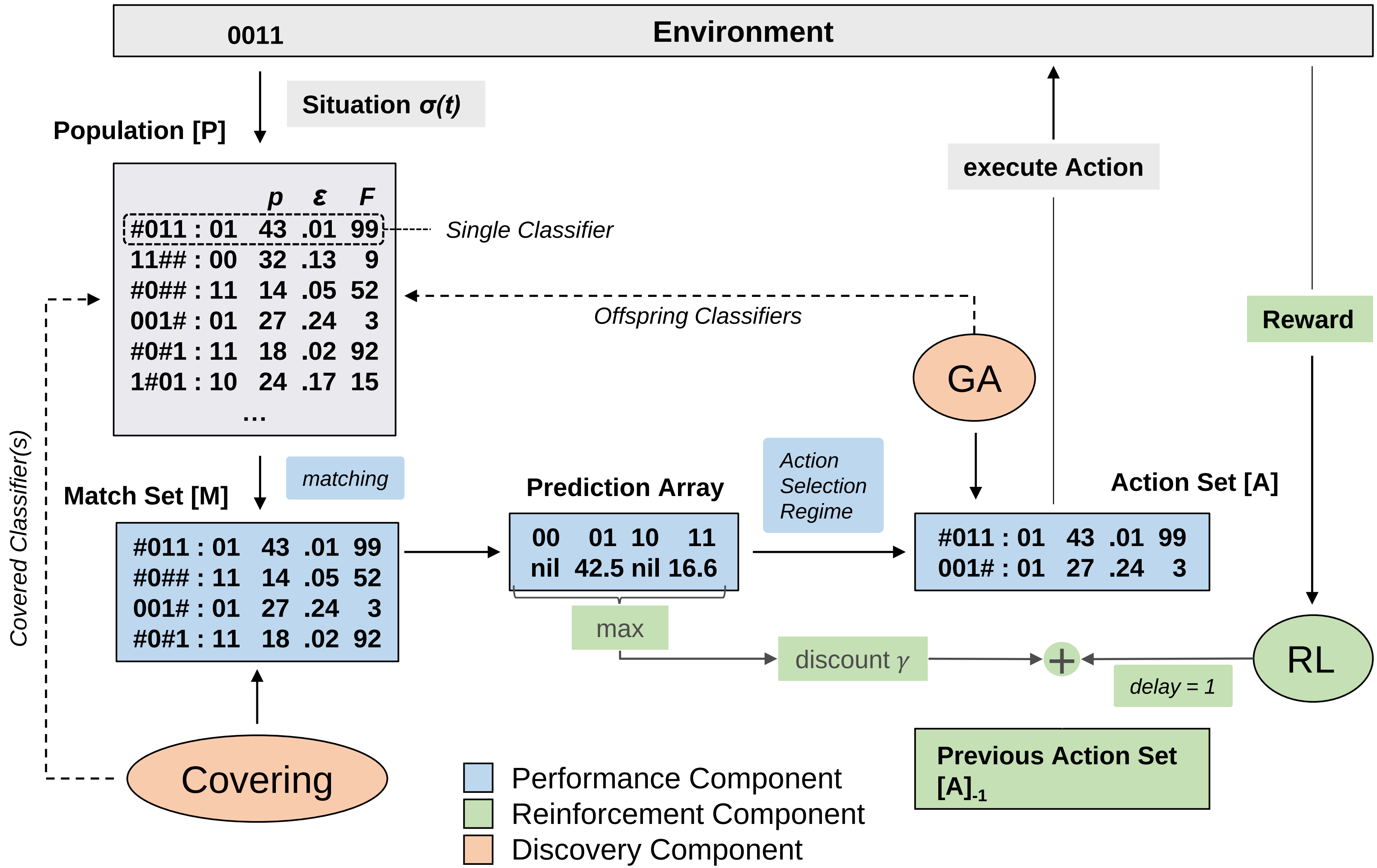
One-point crossover:



Mutation:


$rand(0,1) < \mu$
$rand(0,1) < \mu$
0 1 0 0 0 1 0 1
1 1 0 1 1 1 0 0
$rand(0,1) < \mu$
$rand(0,1) < \mu$
Mutation
0 1 0 1 0 1 1 1
0 # 0 0 0 1 0 0
Mutated Offspring

Multi-step problems



ORGANIC COMPUTING

Credit assignment

- r may or may not be retrieved in each step.
- Update of classifier attributes is performed on the action set of the previous time step $t-1$ ($[A]_{-1}$).
- The maximum system prediction $P(a)$ from the PA is discounted by a factor γ (usually $\gamma = 0.95$).
- Additionally, the reward from the previous time-step is added in (may be 0).
- This delay allows to retrieve "information from the future".

Distinguish:

- In single-step environments $P = r_{imm}$.
- In multi-step problems $P = r_{t-1} + \gamma * \max P(a)$.

Single- vs. multi-step problems

Real-world problems

- E.g.: traffic control
- There is no ‘end‘ of the process!
- Hence: there is no reward!
- However, we can handle the control problem as single-step problem.
  - Activate XCS in discrete cycles.
  - Perform observation and adaptation loop.
  - Use utility function: (i) to estimate success, (ii) to analyse conditions.

- For the remainder of this lecture, we only consider single-step problems with immediate reward.