Intelligent Systems

Chapter 15: Learning

Winter Term 2019 / 2020

Prof. Dr.-Ing. habil. Sven Tomforde

Institute of Computer Science / Intelligent Systems group

About this Chapter

Content

- Motivation
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
- Example application
- Variants
- Conclusion
- Further readings

Goals

Students should be able to:

- Explain what reinforcement 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.

Agenda

- **Motivation**
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
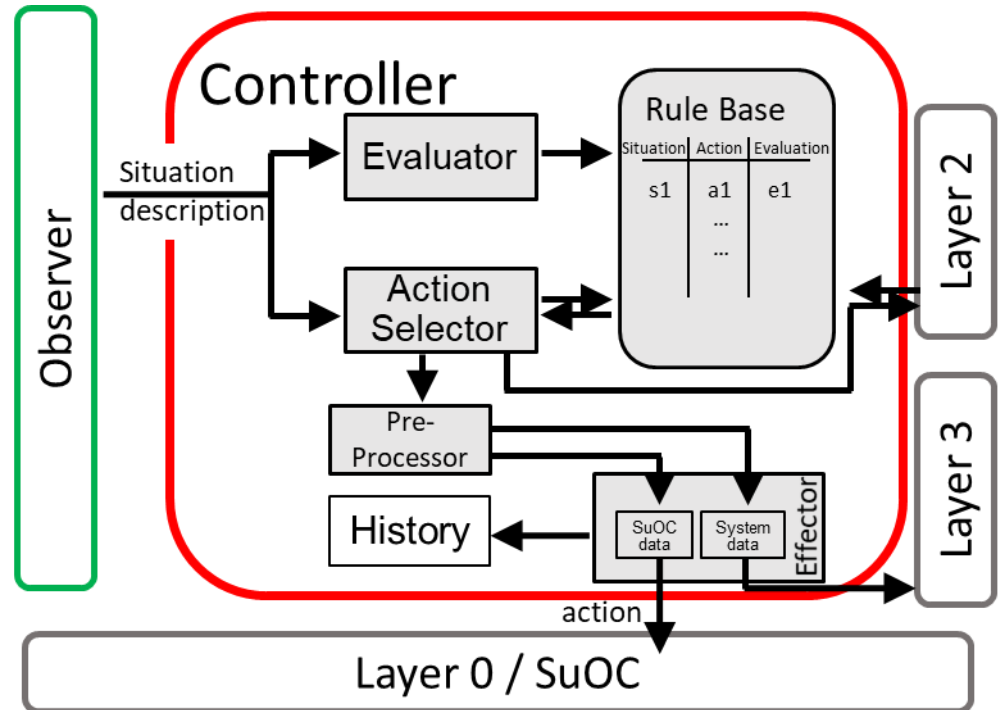- Example application
- Variants
- Conclusion
- Further readings

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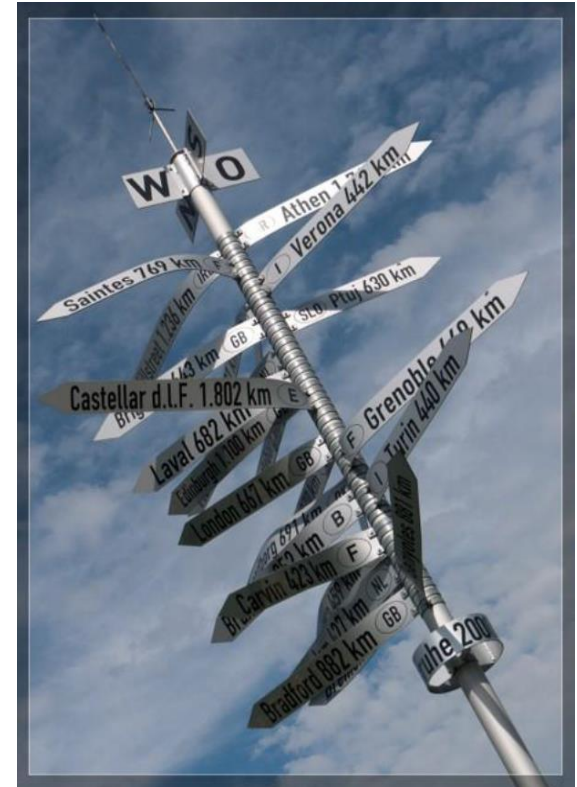
- Machine learning techniques seem to be a promising approach for continuous self-improvement in Organic Computing systems.
→ How can computers be programmed so that problem solving capabilities are built up by specifying "what is to be done" rather than "how to do it"? (Holland, 1975).

- 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.
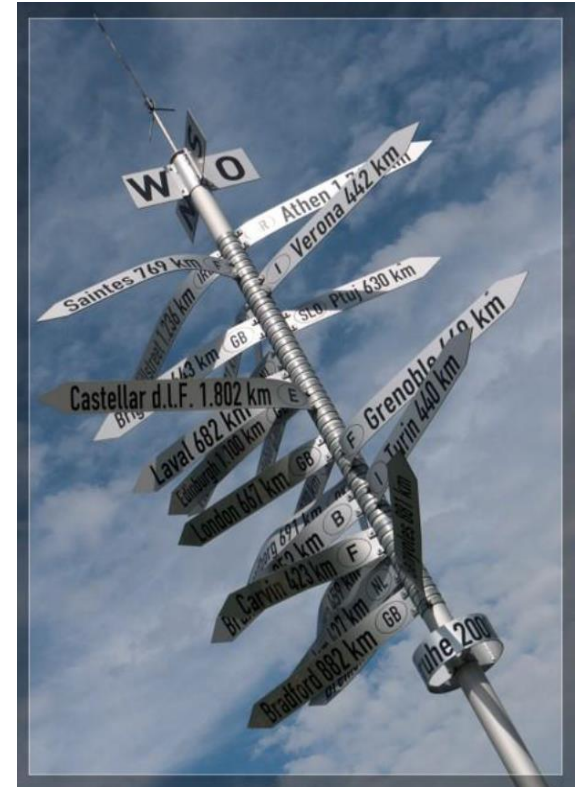
Motivation (2)

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.

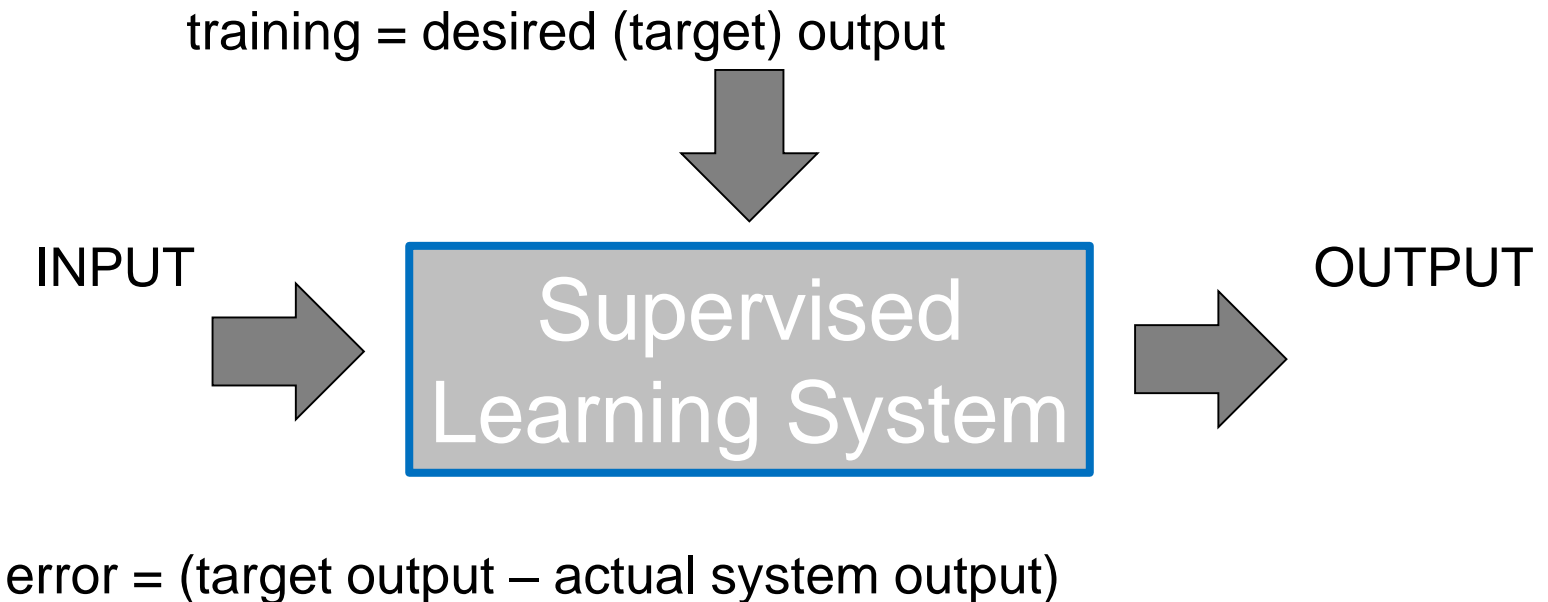# Agenda

Machine Learning

Common definition for "**Machine Learning**" (by Mitchell):

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.

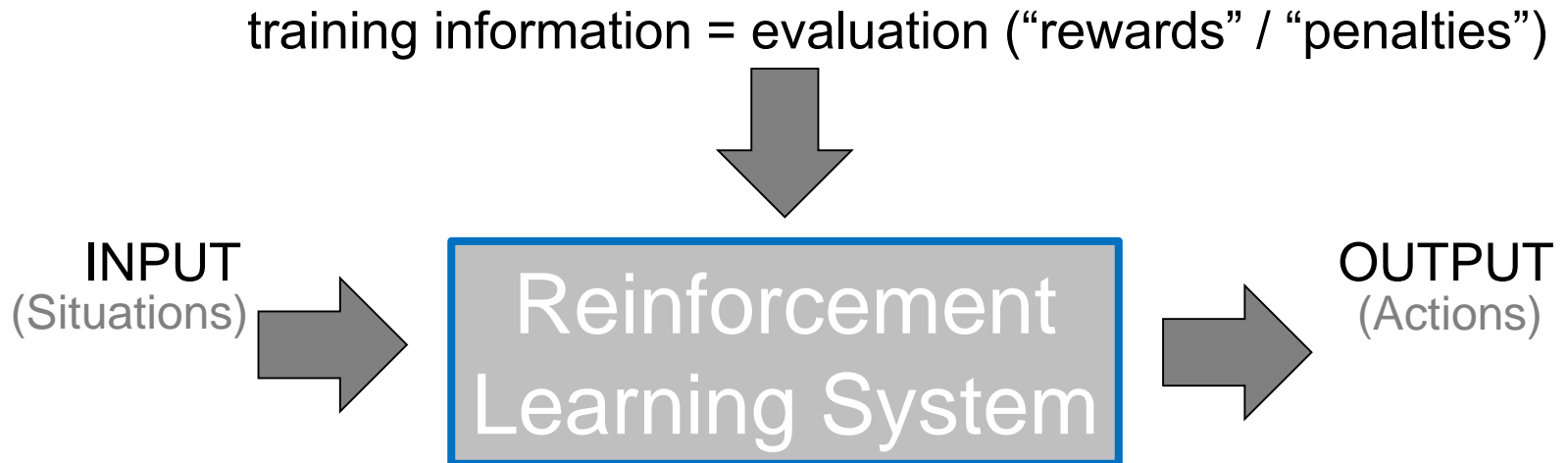→ *Increase the system performance by taking experiences with the same problem instance into account!*

Reinforcement Learning

- ## 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

Supervised Learning

training = desired (target) output

INPUT → Supervised Learning System → OUTPUT

error = (target output – actual system output)

C A U
Christian-Albrechts-Universität zu Kiel

Reinforcement Learning

training information = evaluation ("rewards" / "penalties")

INPUT
(Situations)

Reinforcement
Learning System

OUTPUT
(Actions)

Goal: achieve as much reward as possible!

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

10

- 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).


Woods14

Woods1

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.
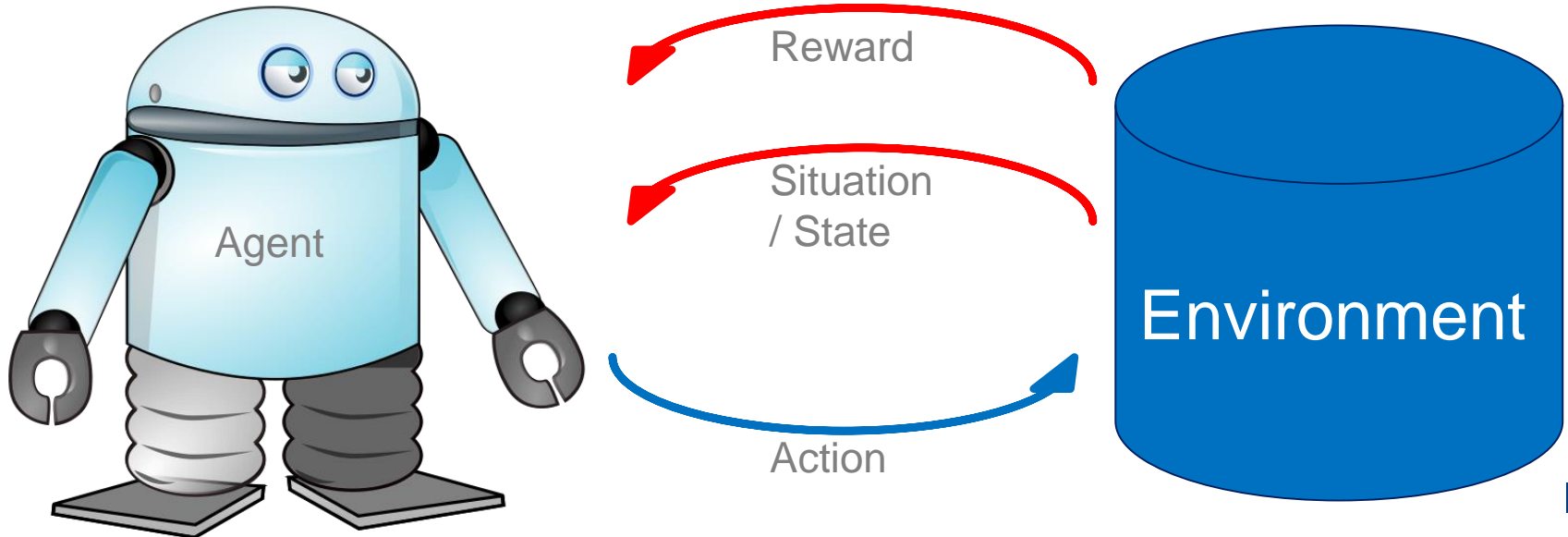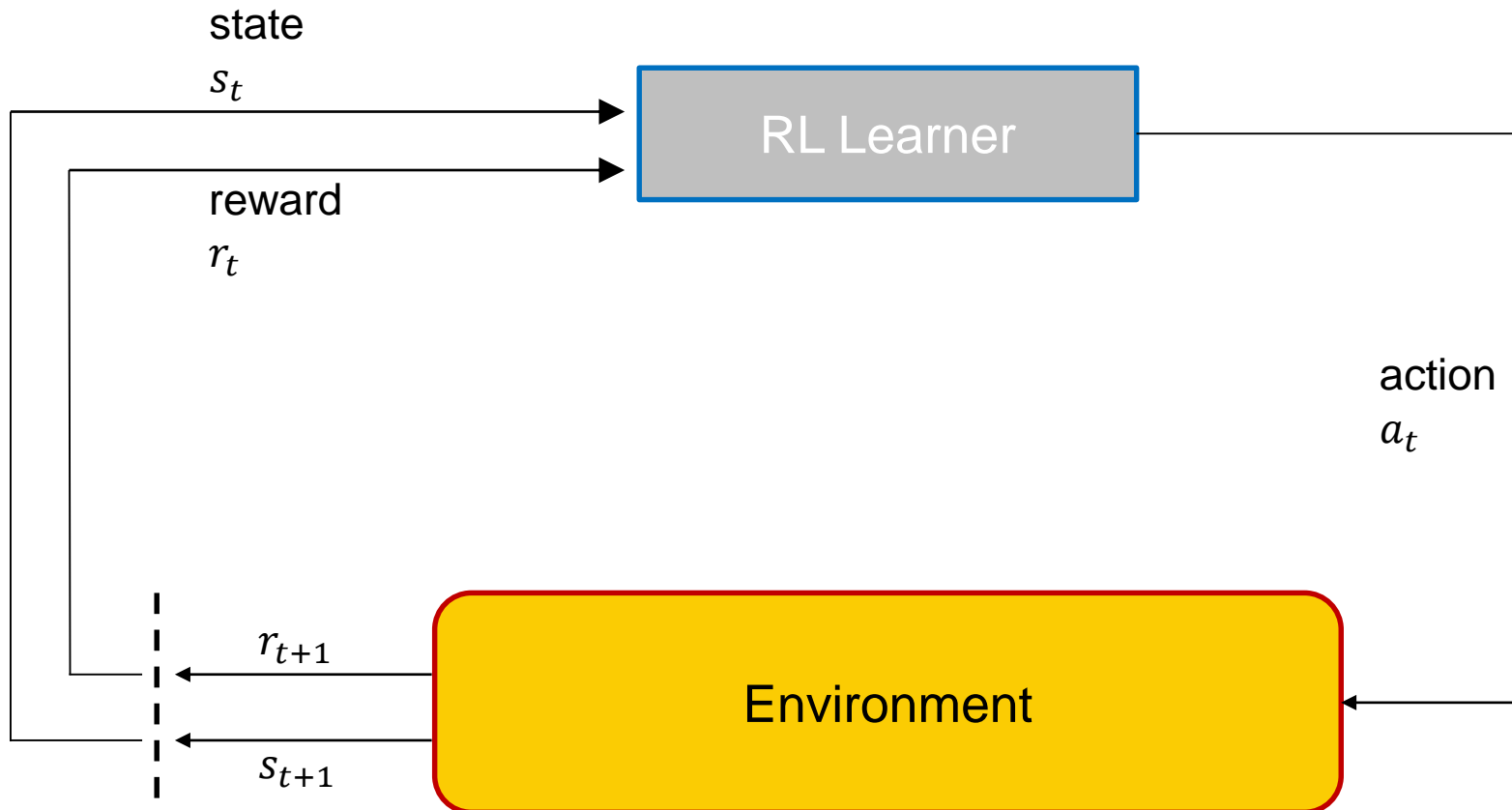
Agent
- Observation: 8 cells
- Status
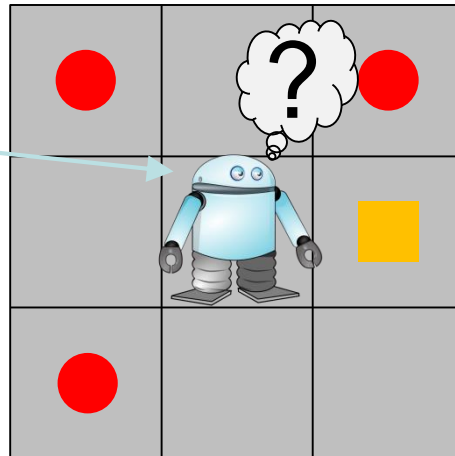
● Obstacle

■ Previous cell

Where to go?

Prof. Dr.-Ing. Sven Tomforde / Intelligent Systems group

12

Agent model

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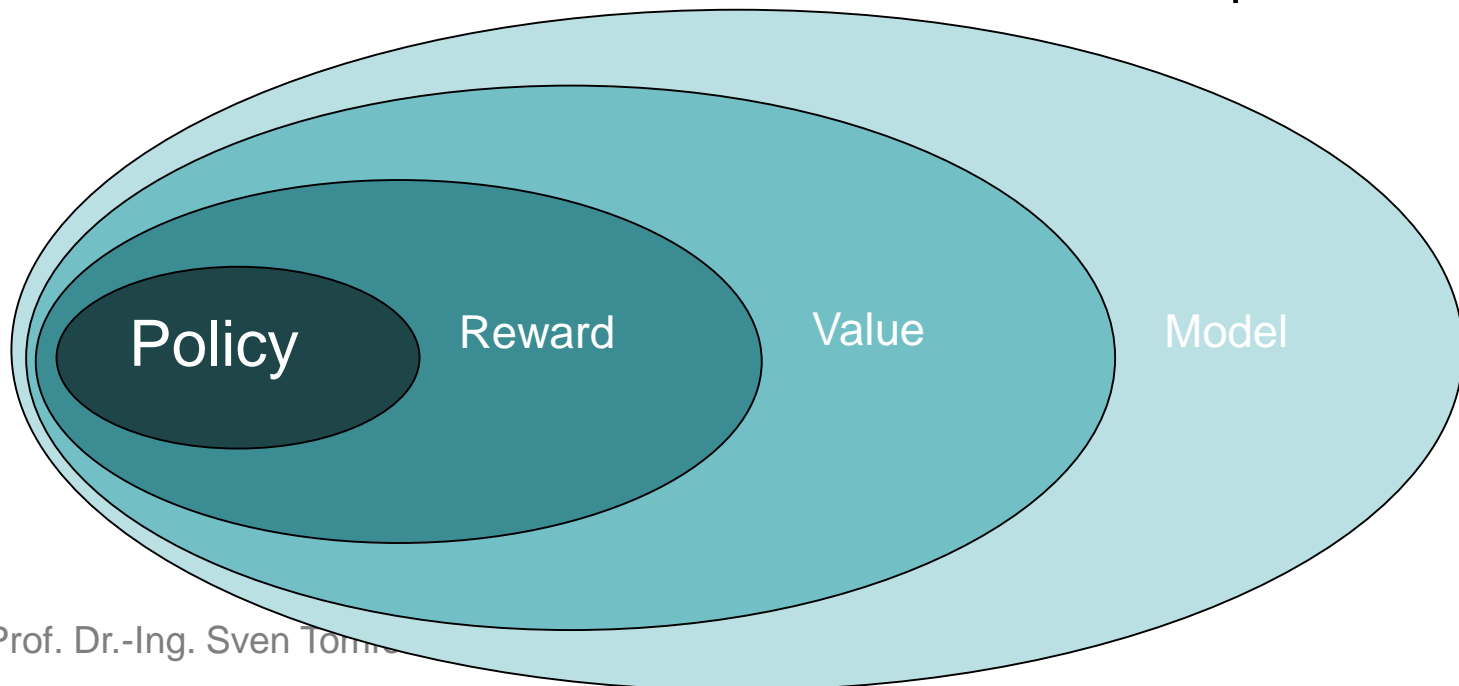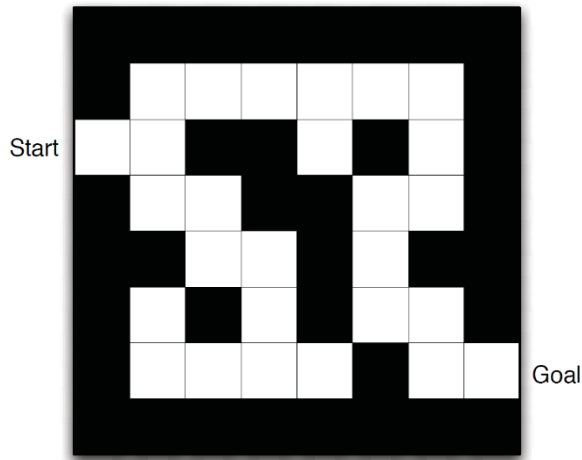 - Affects the environment
  - Environment is stochastic and uncertain

Agent

Reward

Situation / State

Action

Environment

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?
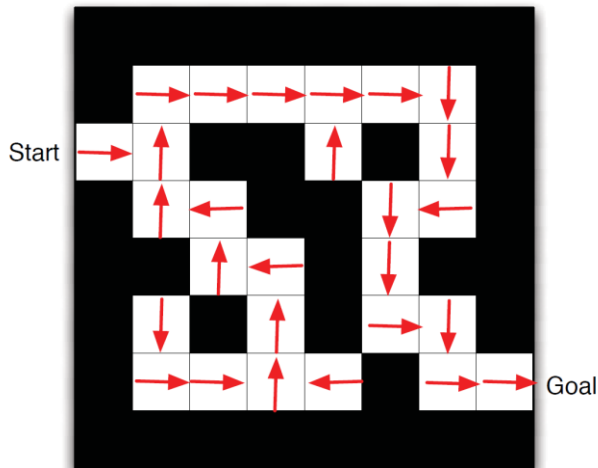- Model: What follows from the actions? What is the impact?
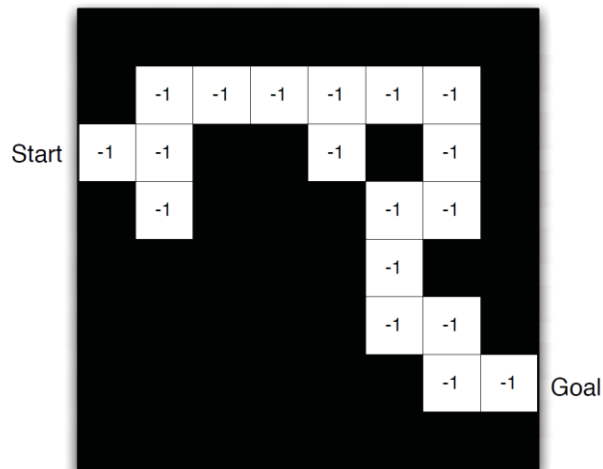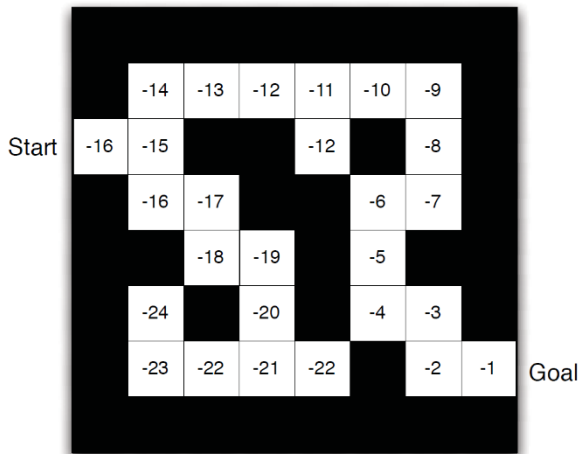
MAZE Example

- Rewards: -1 per time step
- Actions: N, E, S, W (north, east, south, west)
- States: Agent's location

Policy:

- Arrows represent policies
- One policy $\pi(s)$ for each state s

Value function:

- Numbers represent value $v(s)$ of each state s
- One policy $\pi(s)$ for each state s

Model:

- Grid layout represents transition model
- Numbers represent immediate reward from each state s (same for all a)
- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state

Exploration vs. exploitation

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

vs.

Exploitation:
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

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.
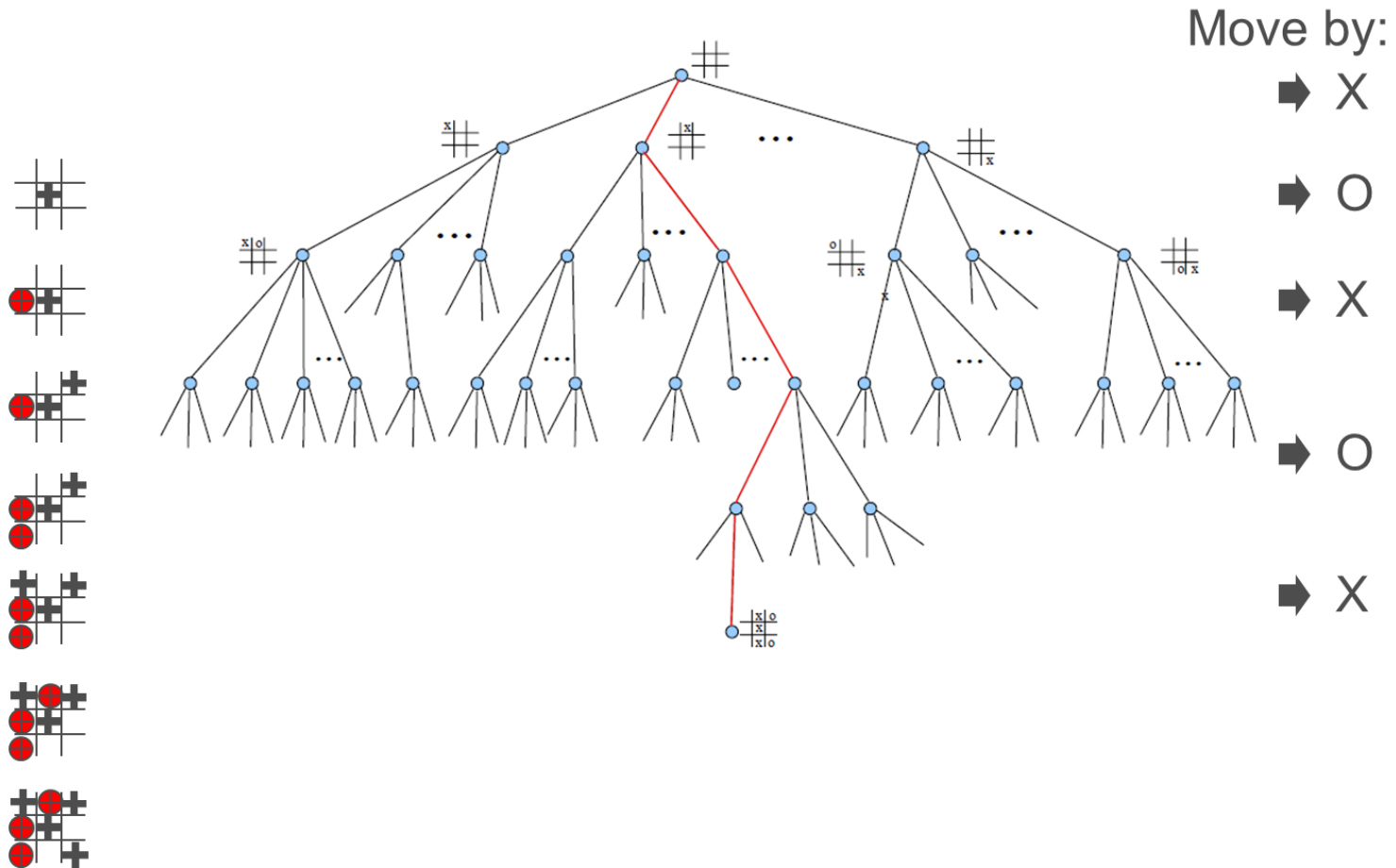
Example: Tic-Tac-Toe (2)

Step 1: Create a table with one entry per state

State	p(x)	comment
♯	0.5	
♯	0.5	
⋮	⋮	
⊞	1.0	Game won!
⋮	⋮	
⊞	0.0	Game lost!

p(x) = (estimated) probability to win

Step 2: Play a lot of games!

- For each move, look ahead one step.

Current state

Possible next states
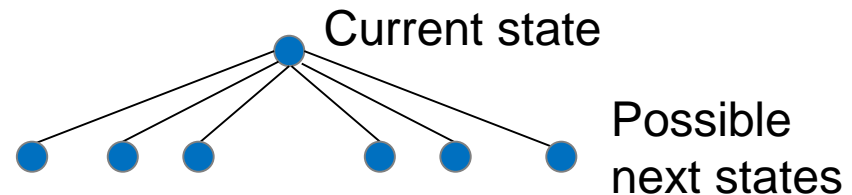
- Choose move with highest p(x): greedy.
- With a certain probability (e.g. 10%), choose a random move (an exploring move).

Move by:

Opponent

Agent

Opponent

Agent

Opponent

Agent

a — Starting position

b

c

d

e*

e

f

g
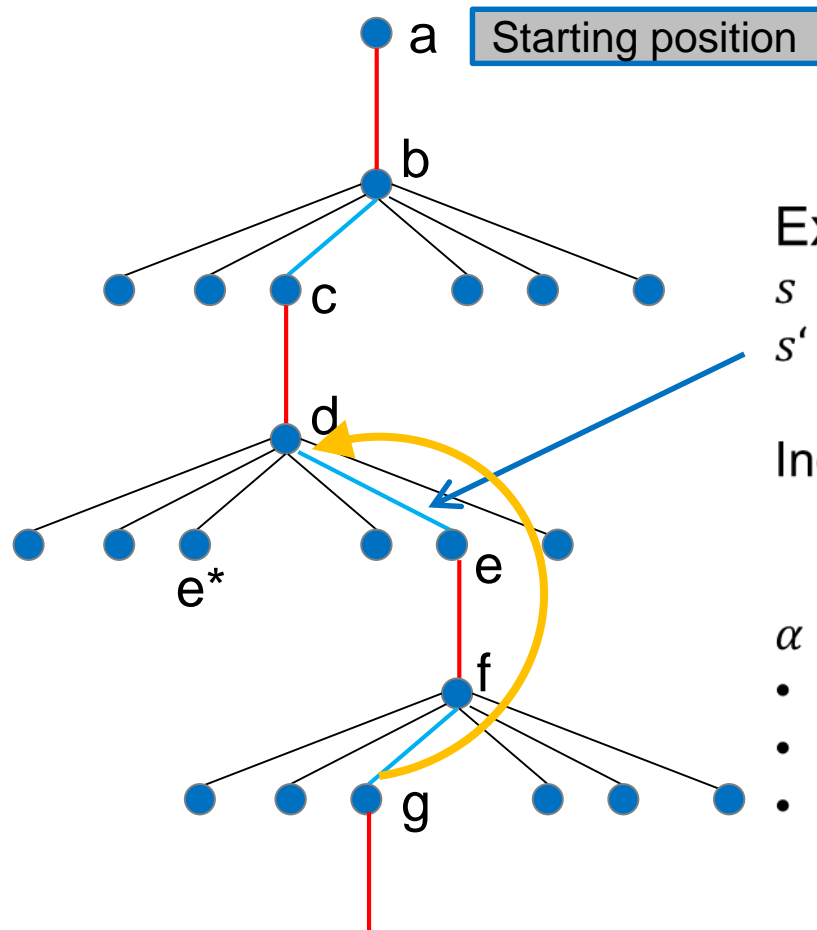
Exploring move:

s - State before greedy move

s' - State after greedy move

Increment $p(s)$ to $p(s')$:

$$p(s) \leftarrow p(s) + \alpha\,[p(s') - p(s)]$$

α is the learning rate
- A small positive value
- E.g.: 0.1
- Defines the rate of adaptation
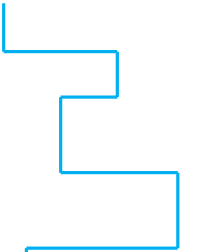
Improvements for the Tic-Tac-Toe player

- Take notice of symmetries!
  - Representation / Generalisation?
  - How can it fail?
- Random moves
  - Why do we need random moves?
    → Exploration vs. exploitation!
  - What about the 10%?
- Can we learn from random moves?
- Can we learn offline?
  - Pre-learning by playing against oneself?
  - Domain knowledge of experts?
  - Using the learned models of the opponent?
- …

Generalisation of the knowledge

Table

Generalised approximation (function)

State   p

State   p

S_1
S_2
S_3

Training

⋮

S_n

Tic-Tac-Toe

$\rightarrow$ Obviously, a simple example. But why?

- Finite, small number of states
- Deterministic (look-ahead of one)
- All states are recognisable
- Direct evaluation of move possible
- No noise when observing the state, etc.
- …

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).

Q-Learning

Example for a simple RL technique

- Q-Learning
- One of the early breakthroughs in reinforcement learning
- An off-policy Temporal Difference Learning algorithm known presented by Watkins in 1989
- Maintain list of Q values for all state-action pairs
- Defined as: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$

S_t - State at time t; $\quad\quad$ A_t - Action at time t
$Q(S_t, A_t)$ - Estimated value of applying A_t in S_t
α – Learning rate; $\quad\quad$ γ – Discount factor
R_{t+1} - Reward at time t+1

Algorithm:

> **Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**
>
> Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
> Repeat (for each episode):
>   Initialize S
>   Repeat (for each step of episode):
>   Choose A from S using policy derived from Q (e.g., ϵ-greedy)
>   Take action A, observe R, S'
> $Q(S, A) \leftarrow Q(S, A) + \alpha \big[R + \gamma \max_a Q(S', a) - Q(S, A) \big]$
> $S \leftarrow S'$
>   until S is terminal

- What are the major drawbacks of the Q-learning technique?

- Or in other words: Why is it seldom applicable to real-world problems in intelligent systems?

Agenda

- Motivation
- Reinforcement learning
- **Extended Classifier System**
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
- Example application
- Variants
- Conclusion
- Further readings

Names to remember in LCS research

- 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

- Initial 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.

- 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)
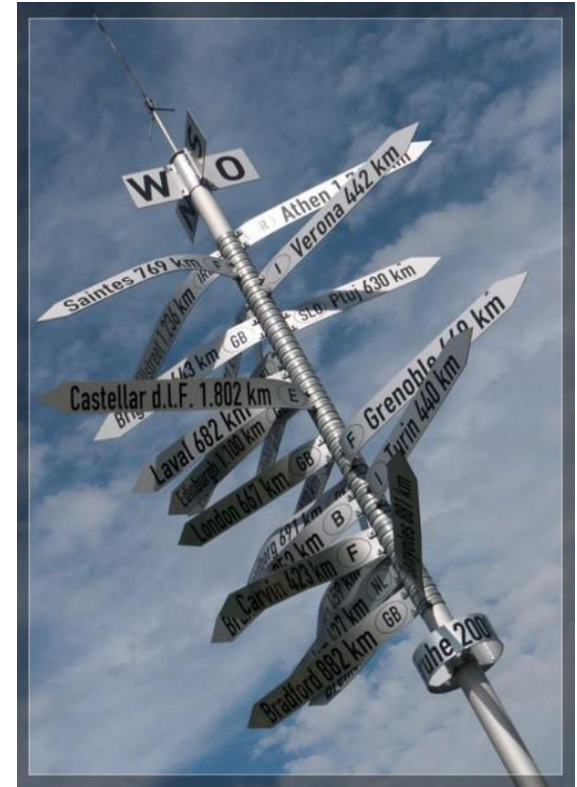
One cycle through XCS
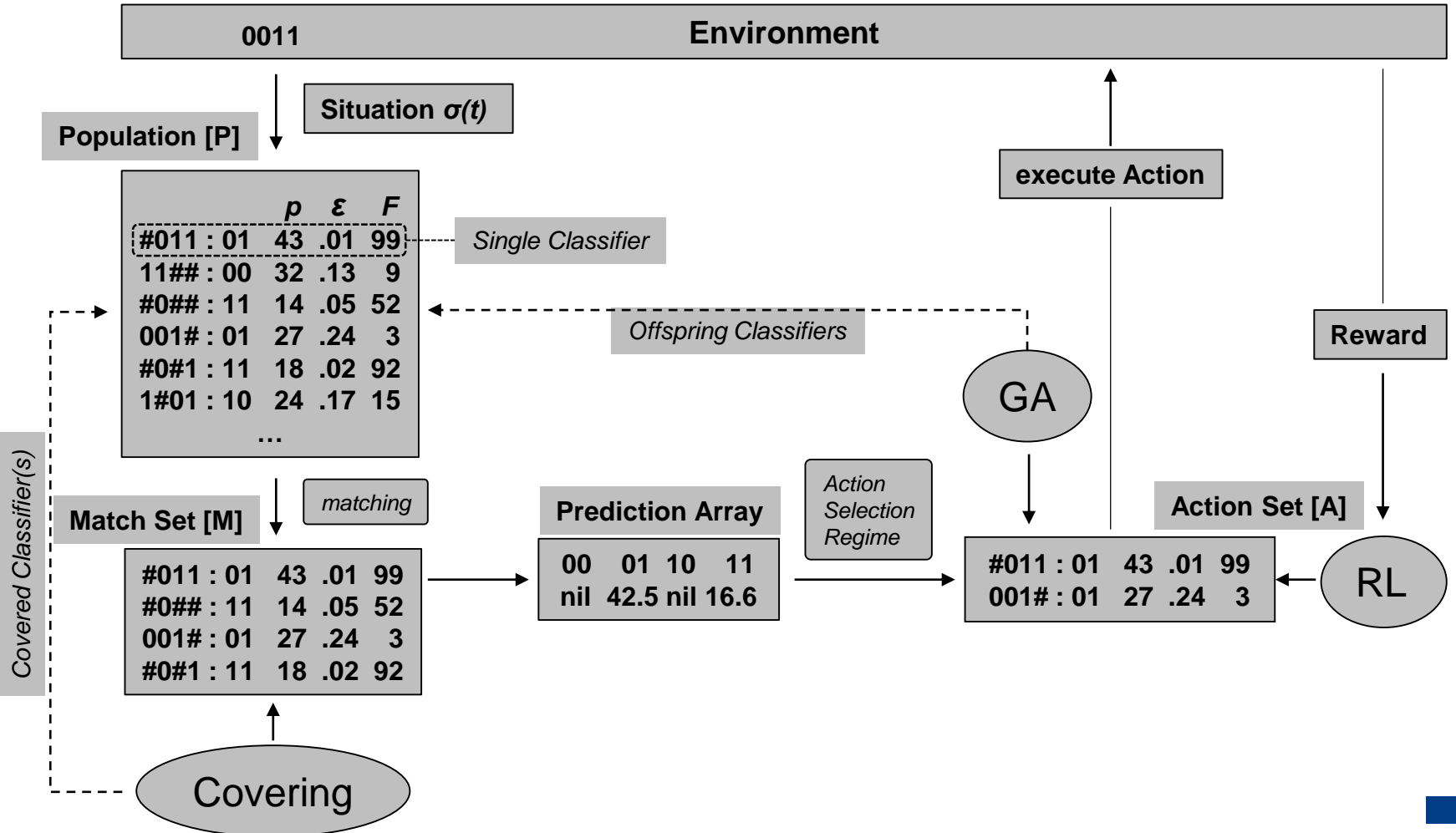
A single iteration through the main loop

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.

Agenda

- Motivation
- Reinforcement learning
- Extended Classifier System
- **Strength vs. accuracy**
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
- Example application
- Variants
- Conclusion
- Further readings

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.
  - *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.

Agenda

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
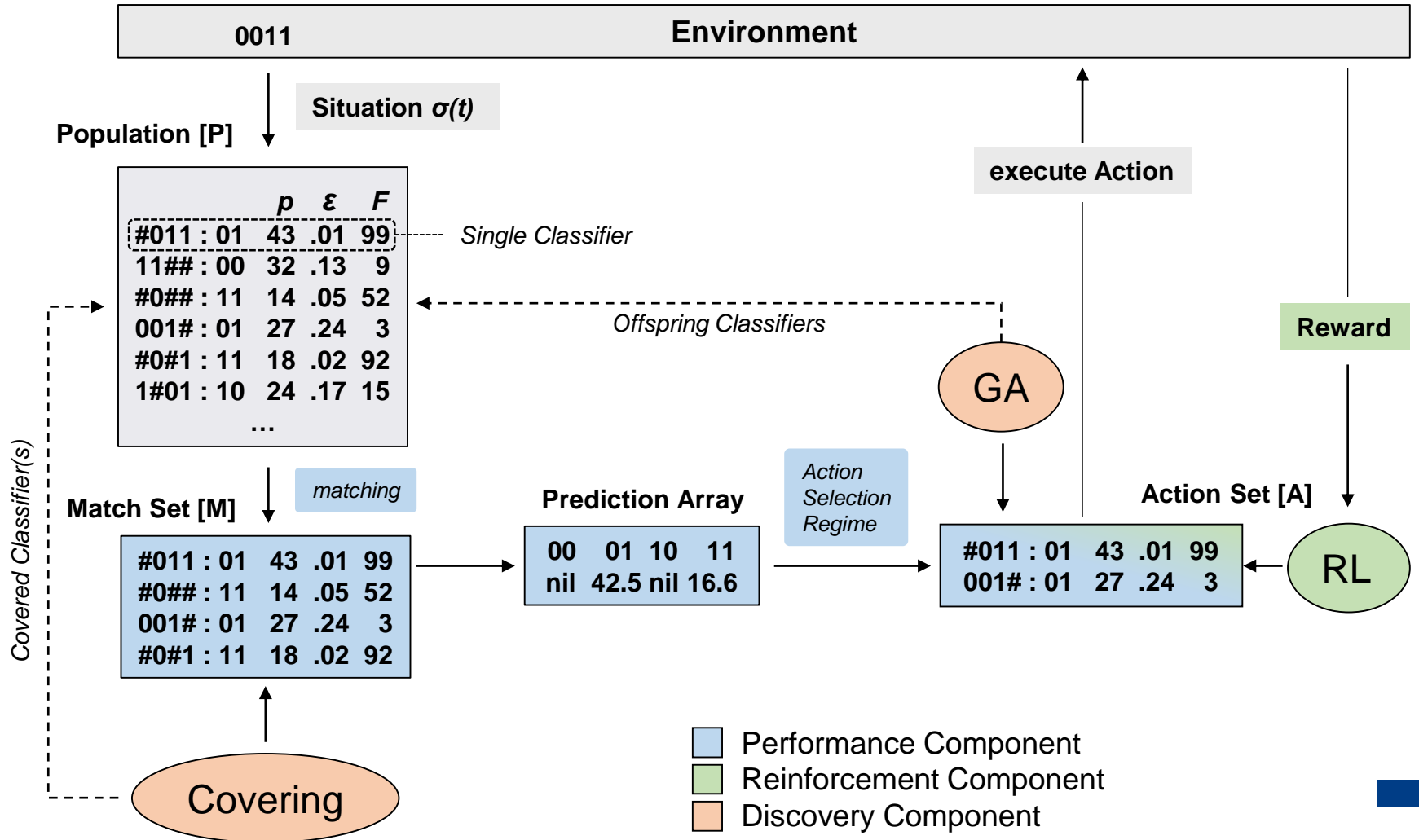
Matching

- 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 $t = 1$: $\quad \sigma(1) = 011001$
- 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 $\sigma(t)$

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.

Update rules:

- $\epsilon_j = \epsilon_j + \beta(|P - p_j| - \epsilon_j)$

- $p_j = p_j + \beta(P - p_j)$

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

- β is the learning rate (typically set to 0.2)
- α (often set to 0.1) and v (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.

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).

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.

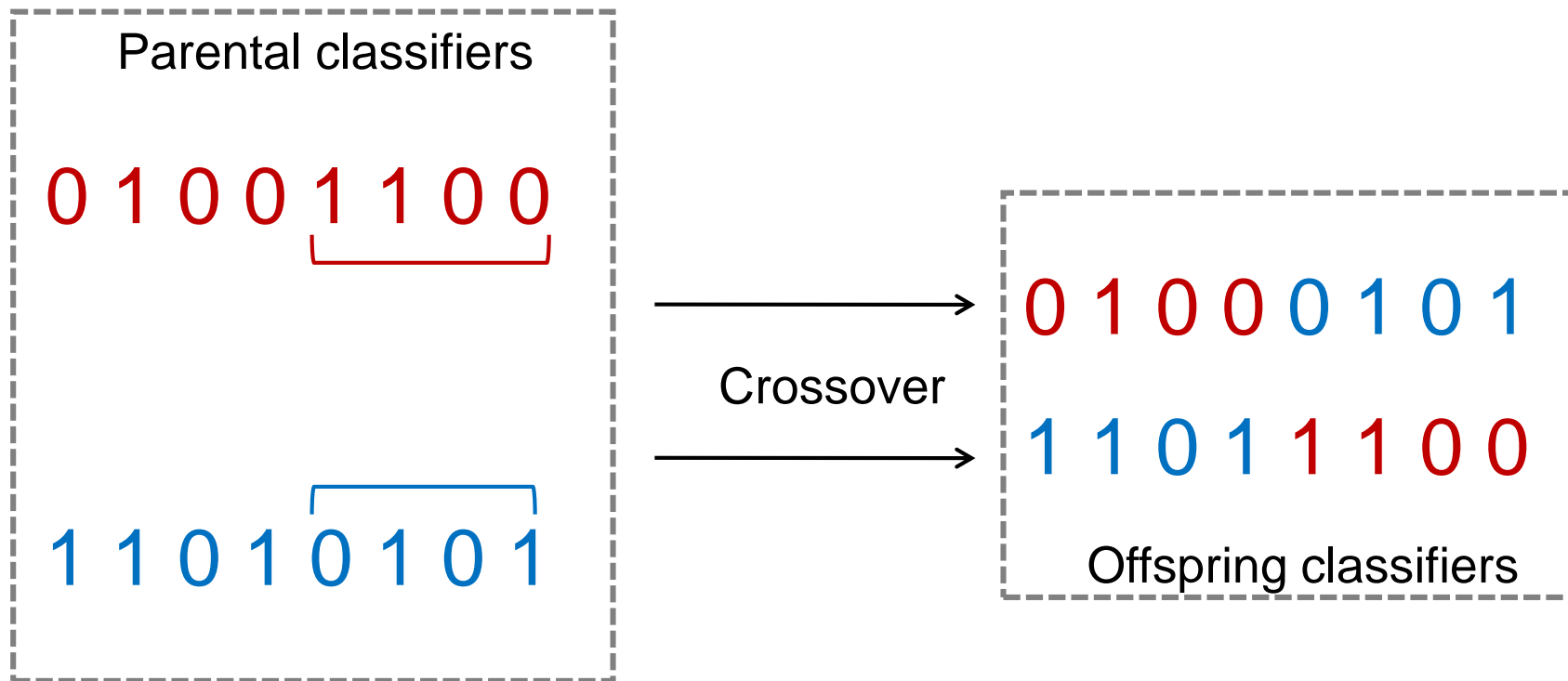- The selected parents are copied to generate two offspring classifiers cl_{off}.

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 $\{0, 1, \#\}$.

One-point crossover:

Parental classifiers

0 1 0 0 1 1 0 0

1 1 0 1 0 1 0 1

Crossover

0 1 0 0 0 1 0 1

1 1 0 1 1 1 0 0

Offspring classifiers

Prof. Dr.-Ing. Sven Tomforde / Intelligent Systems group

50

Mutation:

Agenda

- Motivation
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- **Credit assignment**
- Real-world applicability
- XCS-O/C
- Example application
- Variants
- Conclusion
- Further readings

Credit assignment

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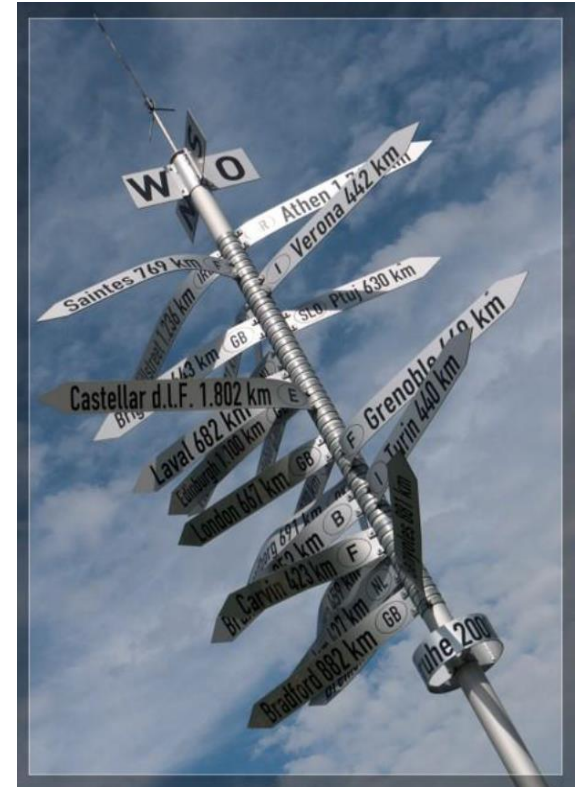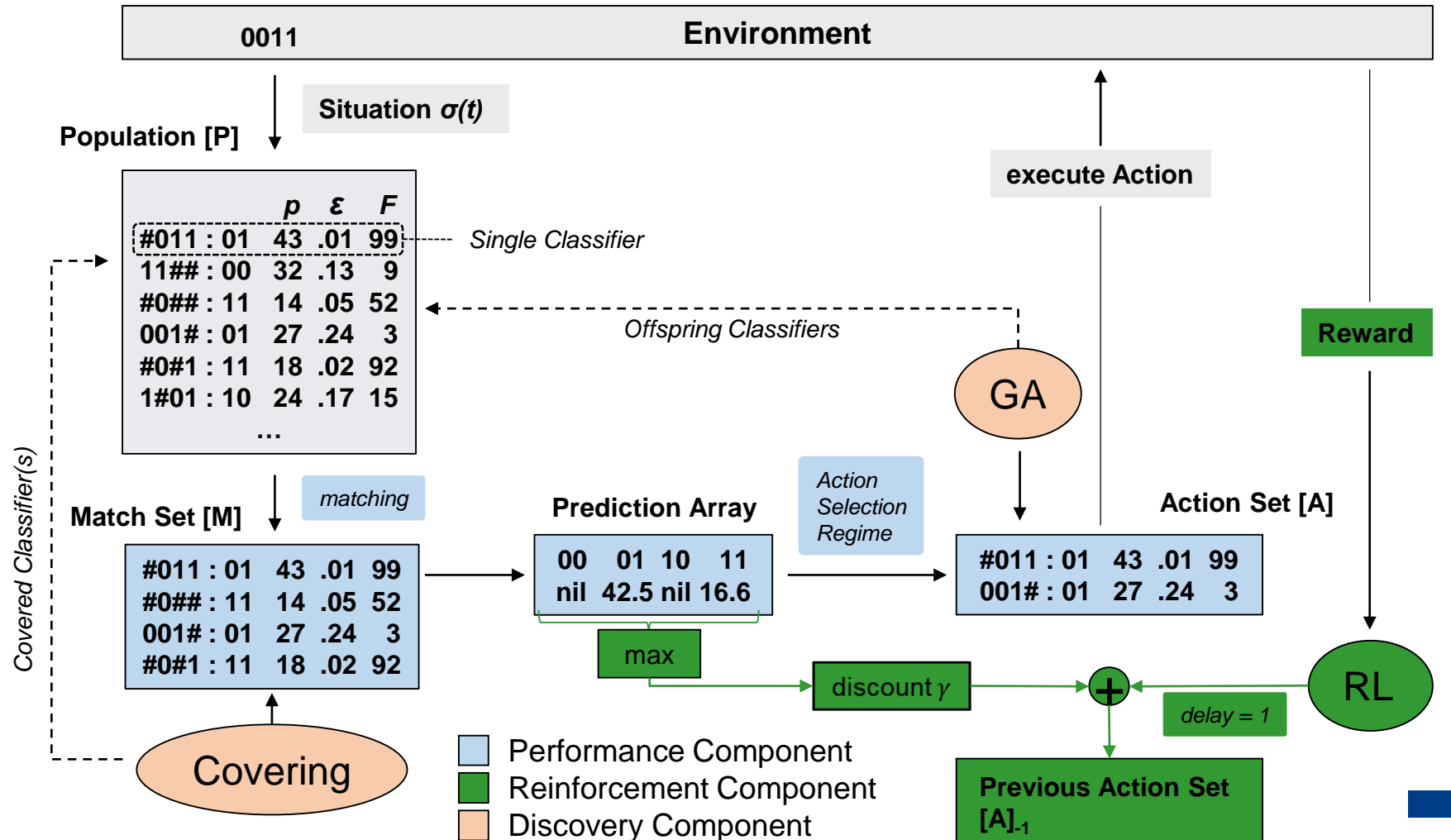- Update of classifier attributes is performed on the action set of the previous time step $t-1$ ($[A]_{-1}$).
- The maximum system prediction $P(a)$ from the PA is discounted by a factor γ (usually $\gamma = 0.95$).
- Additionally, the reward from the previous time-step is added in (may be 0).
- This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

Distinguish:

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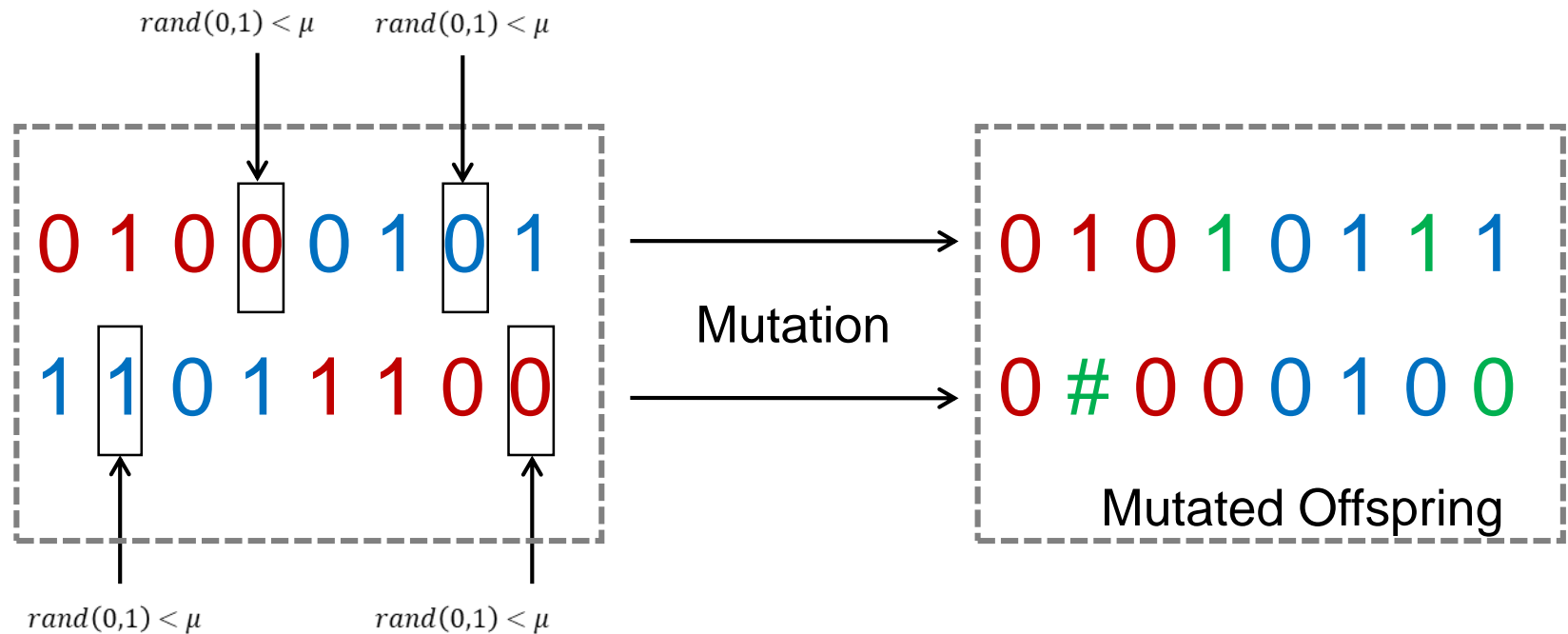
Single- vs. multi-step problems

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.

Agenda

- Motivation
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- **Real-world applicability**
- XCS-O/C
- Example application
- Variants
- Conclusion
- Further readings

From binary to real-valued problem domains
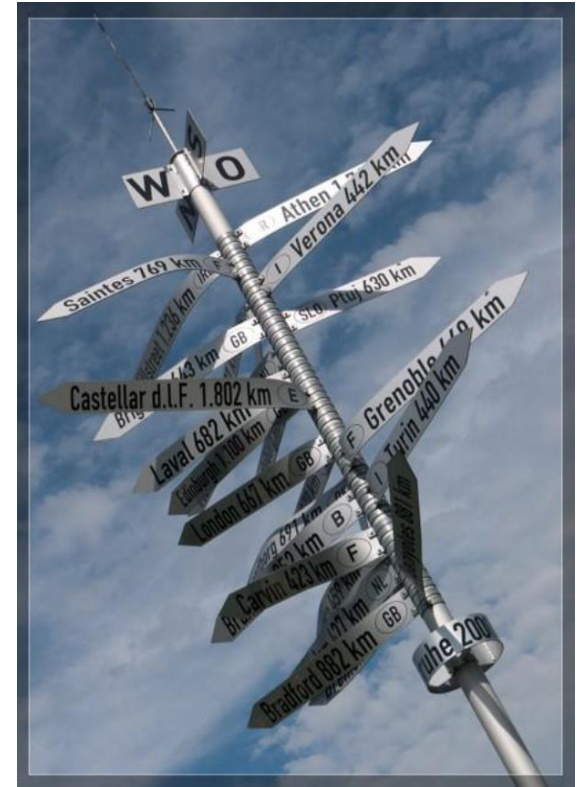
- Wilson proposed changes to conventional XCS to allow for real-valued input (cf. [Wilson2000]).

- To accomplish this, some changes were necessary regarding the internal calculations:
  – Representation of the condition
  – Matching
  – Covering
  – GA (Crossover, Mutation)

- The extended system is called XCSR in literature.

The condition part C:

- The situation $\sigma(t)$ is now represented as an n-dimensional input/feature vector.
- The input vector is defined as $\vec{x}_t = (x_1 \ldots x_n) \in X \subseteq \mathbb{R}^n$.
- Thus: a binary string is not appropriate anymore!

- For each dimension x_i a so-called interval predicate has to be defined.
- An interval predicate is a tuple (l_i, u_i) representing:
    - a lower l_i and
    - an upper bound u_i .
- The geometric interpretation of a condition for real-valued inputs is that of hyper-rectangles.
- Accordingly, this condition representation is called hyper-rectangular representation.

- In the following, we assume that the input space X is normalised to the standard interval: [0,1].

- Thus, for an $n = 2$ dimensional problem space a classifier's condition C may look like:

$$cl.\,C \;=\; [(0.30, 0.70), (0.55, 0.95)]$$

- E.g.: This condition would match the input
$$\sigma(t) = (0.4,\ 0.75)$$

- In general, a classifier matches the current input if and only if
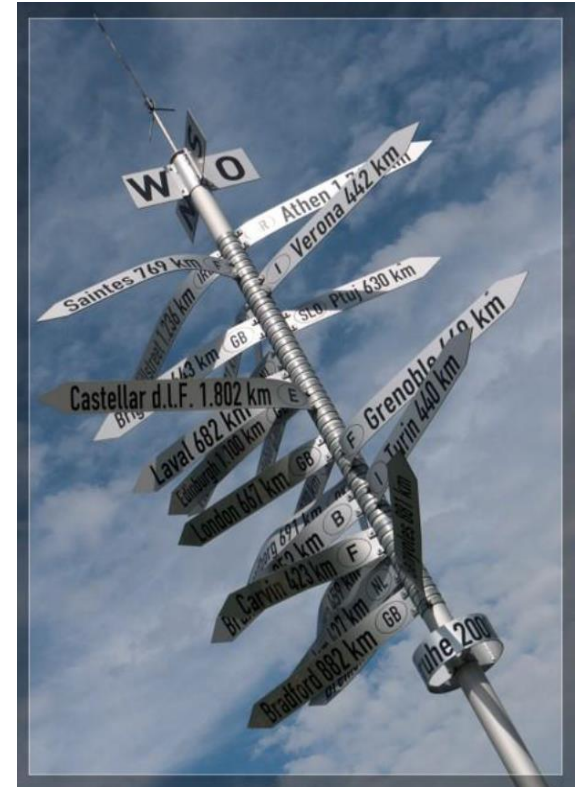
$$\forall i: \quad l_i \leq x_i < u_i \quad i = 1 \ldots n$$

- When covering occurs the newly generated classifier is initialised with predefined initial values as before.

- The condition is set to the current situation

$$\sigma(t) = (x_1 \ldots x_n)$$

- Additionally, to provide interval predicates

$$(l_i, u_i), \ i = 1 \ldots n$$

  - $l_i = x_i - rand(r_0)$
  - $u_i = x_i + rand(r_0)$
  - $rand(r_0)$ delivers a uniformly distributed random number between 0 and r_0.
  - r_0 is a predefined default spread.

- Crossover is actualised as in standard XCS
  - It can be distinguished whether it is allowed to cross in-between a certain interval predicate or only between the interval predicates

- When an allele is selected for mutation its current value is updated according to the following rule:
  - $l_i \pm rand(m_0)$
  - $u_i \pm rand(m_0)$
  - m_0 is a predefined mutation value to extend or shrink the current interval.

- The alleles to mutate are selected probabilistically as in standard XCS.

# Agenda

Modifications needed

- Exploring configuration space online using GA can be dangerous!
    - System will try suboptimal (or even bad) solutions.
    - Example: system could set all traffic lights to green.
    - We cannot allow failures!
- Learning requires experience!
    - What to do in case of missing knowledge?
    - How to avoid failures if the action is unknown (i.e. bad/good)?
- Approach:
    1. Provide 'sandbox' for trying novel behaviour.
    2. Use action that works under 'similar' conditions.
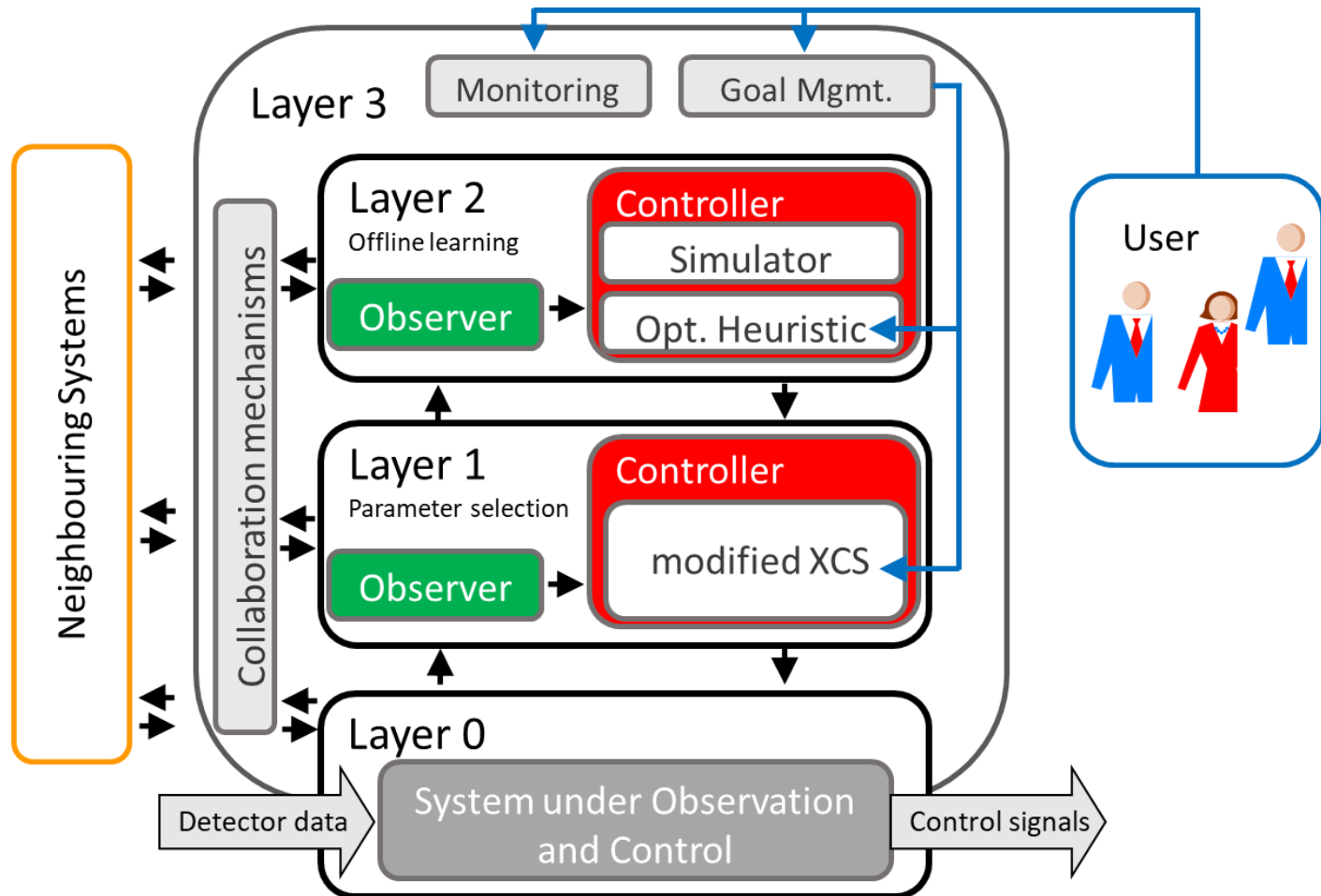
Covering

- Original XCS: covering creates new classifier for the current situation randomly.
- OC: application specific widening of existing classifiers.
  - Select "closest" classifier.
  - Copy classifier.
  - Widen condition until matching.
- Trade-off between "use only tested solutions" and quick reaction time.
- Additionally: threshold used to trigger rule generation at Layer 2.
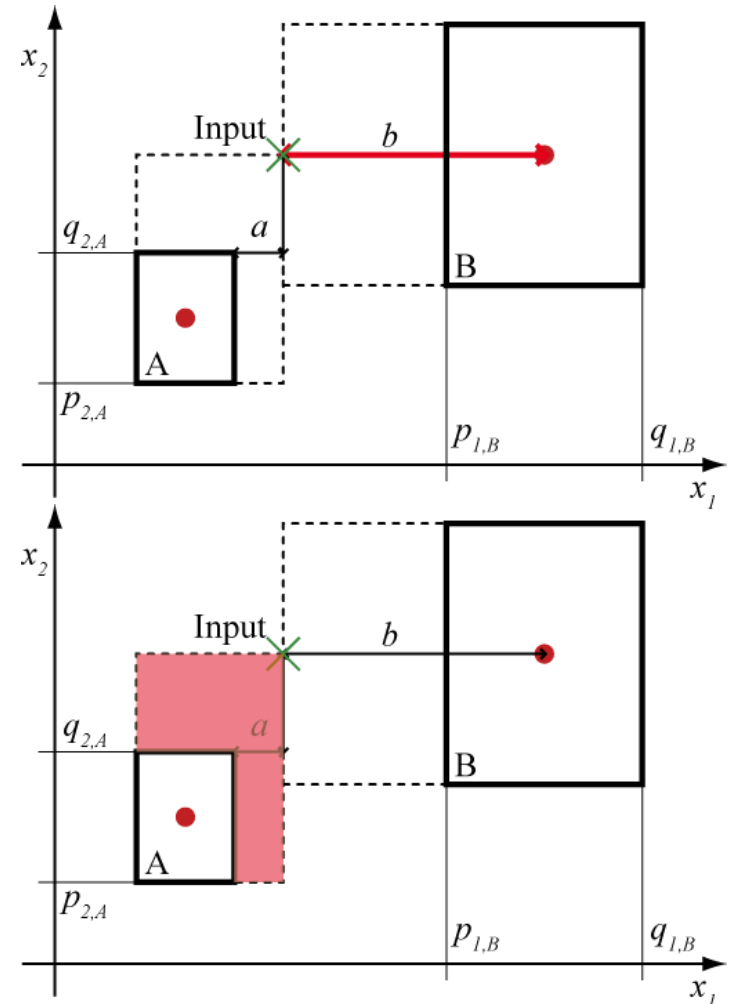
Modifications: rule generation

Idea:

- Covering at Layer 1 solves only a part of the problem.
- Building of match set: trade-off between "use only matching solutions" and competition needed for learning.
- What to do with an empty population?

Approach:

- Generation of new rules is done in a separate component.
- Learning is done offline in a simulator (Layer 2).
- Offline means: takes some time…
  → In the meantime, Layer 1 reacts with covering.

- Generation of candidate actions.
- Quality of action is tested using simulations.
- Simulator is configured using observed conditions (situation).
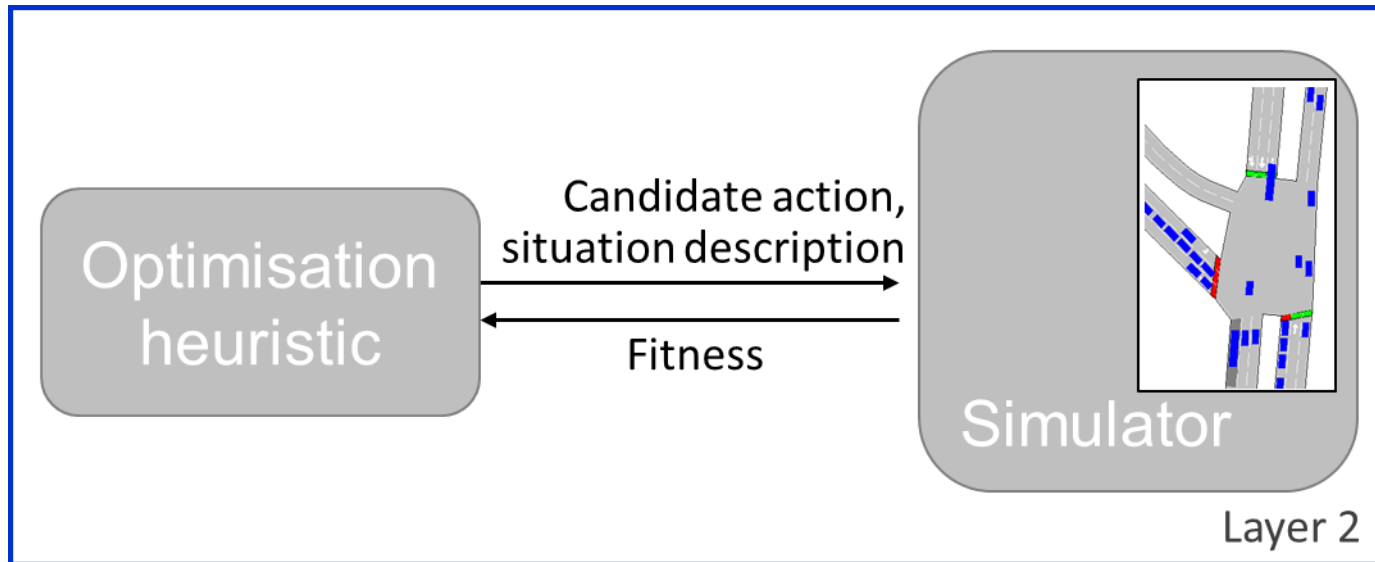
- Fitness is measured.
- Process is repeated until stop criterion is reached.

Novel rule

- Build as follows:
    - Condition: situation description, widened using standard interval.
    - Action: parameter set as result of optimisation process.
    - Prediction: measured in simulation.
      → I.e. utility as observed for best action.
    - Fitness: average of all classifiers in population
    - Error: zero
- Added to rule base of Layer 1

Process is activated:

- If match set is empty
- If fitness of rules in match set is below certain threshold
- Periodically

Agenda

- Motivation
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
- **Example application**
- Variants
- Conclusion
- Further readings

Checkerboard Problem - CBP(2,8)

cf. [Stone2003]

Task of XCS-R:
Of which colour is the field encompassing the query point $\vec{x}_q$?

2 dimensions
$n = 2$
each within $[0,1]$

$\vec{x}_q \in \mathbb{R}^n$
$x_i \in [0,1], i = 1 \dots n$

8 divisions $n_d = 8$
for each dimension
with alternate field
colours (black/white)

Prof. Dr.-Ing. Sven Tomforde / Intelligent Systems group

70

- A CBP(2,8) situation $\sigma(t)$ represents coordinates.
  - E.g.: $\vec{x}_q = (0.25, 0.79)$

- Possible actions are `black` and `white`, respectively `0` and `1`, thus $A := \{0,1\}$

- Reward is 1000 for correct guess and 0 for wrong guess

Single-step or multi-step problem?
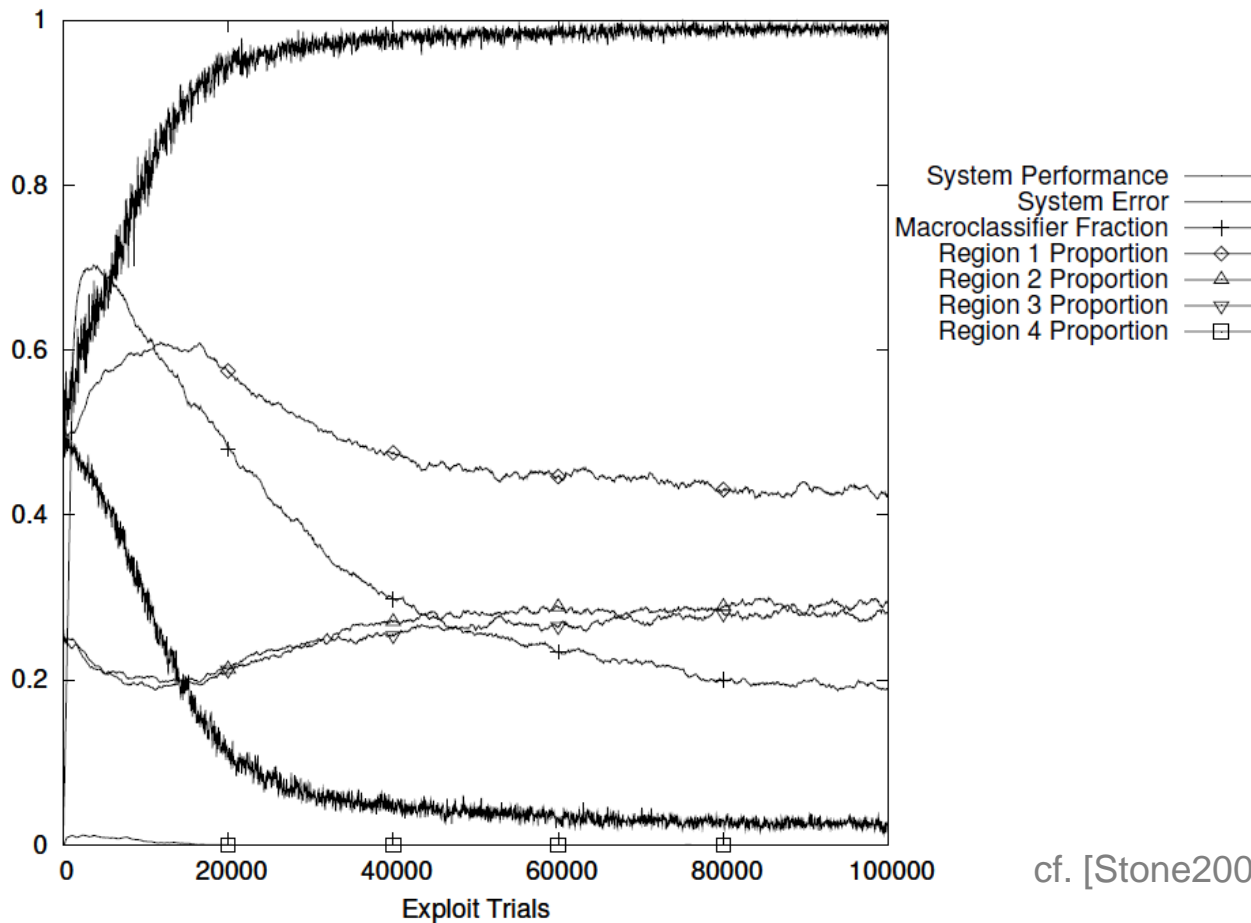
cf. [Stone2003]

Agenda

- Motivation
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
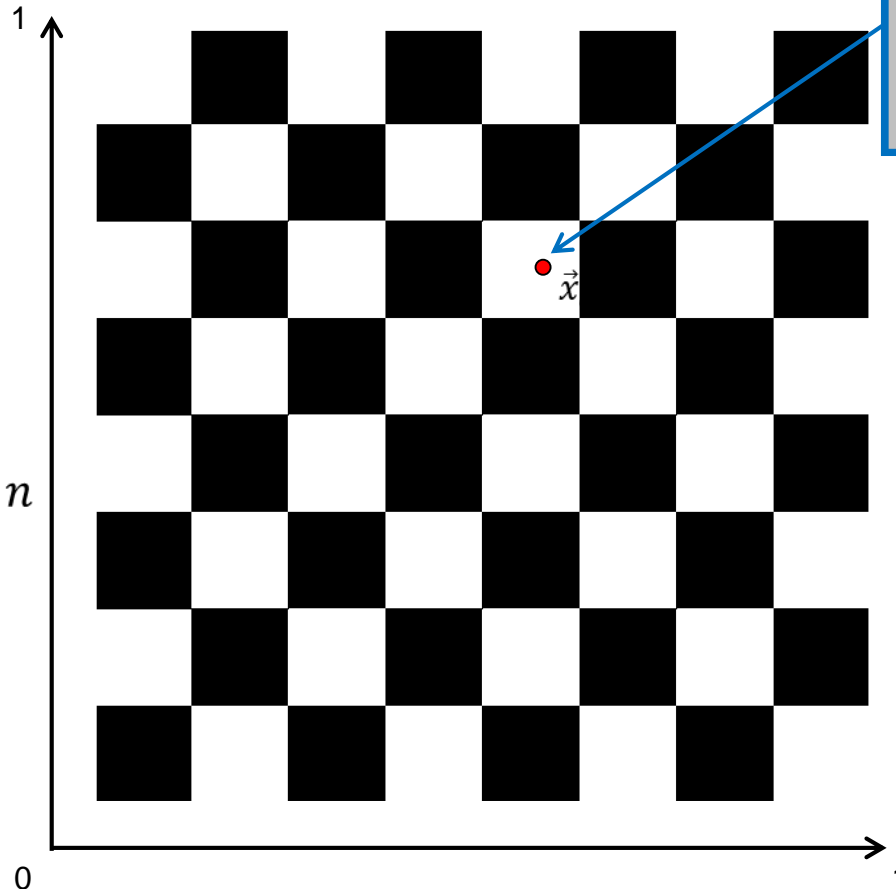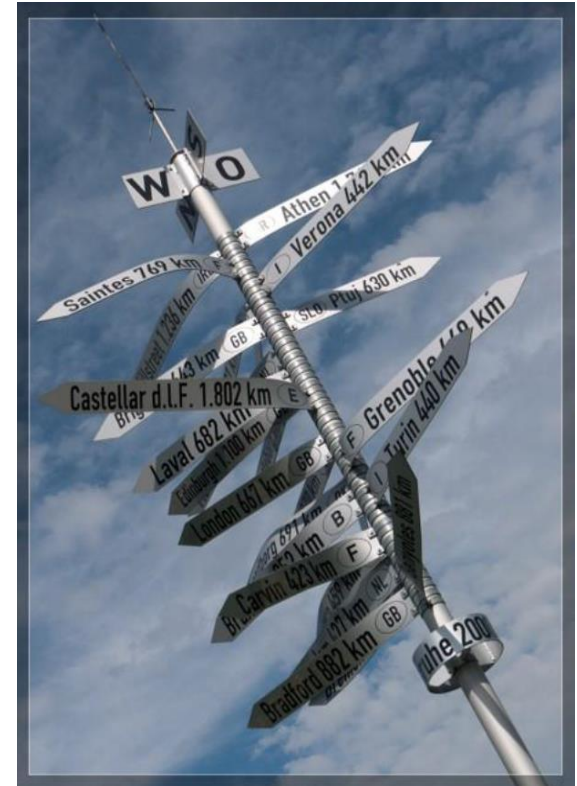- Example application
- **Variants**
- Conclusion
- Further readings

Modifications:

- Representation
  - Use classifiers as basis for interpolation
  - Avoids knowledge gaps
- Generalisation
  - More sophisticated rule combination concepts
  - Recombination of partly matching classifiers
- Involve user
  - Combine, e.g., with active learning concept.
  - Proactive knowledge generation
- Second-order optimisation
  - XCS comes with several parameters
  - Adapt them at runtime (i.e. customisation)

Alternative optimisation techniques

XCS makes use of a Genetic Algorithm

- Is part of the research domain of Evolutionary Computing

- There are alternatives!

- Requirements:
  - Find "good enough" solutions
  - Come up with preliminary result quite fast

- Possible methods and techniques:
  - If available: approximation functions or mathematical functions
  - Swarm-based optimisation heuristics, e.g. Particle Swarm Optimisation
  - OC-based, e.g. Role-based imitation algorithm by Cakar et al.
  - Mimicking physical processes, e.g. simulated annealing
  - An many more…
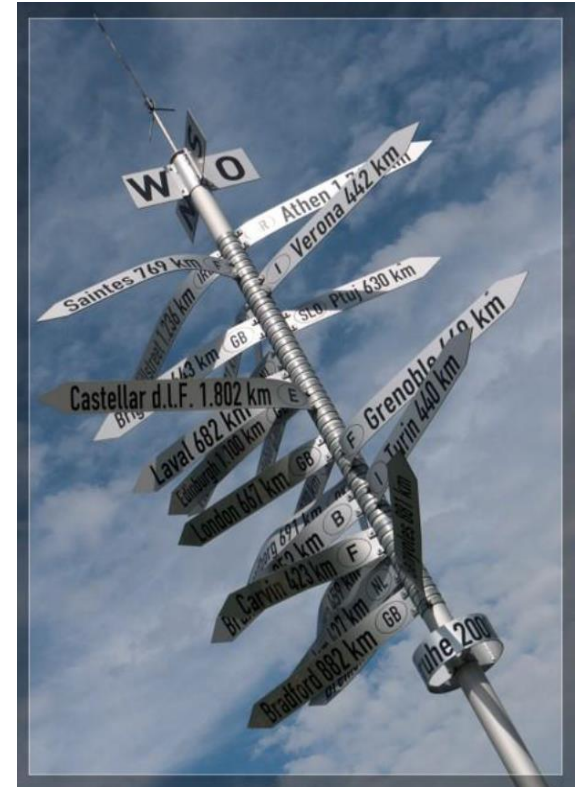
Hardware solutions

Everything in software…?

- Concepts for hardware solutions investigated in the context of OC
- Groups at Munich and Tübingen
- Result: Learning Classifier Table
  - Initial training and rule generation in software
  - Logic at runtime at FPGA
  - Scope in hardware: perform main loop for action selection and modify evaluation parameters

Further issues and challenges

- Besides the hyper-rectangular condition structure, alternative geometric shapes have been proposed
  - Hyper-spherical or Hyper-ellipsoidal representation
  - Weaken the negative effect of high prediction errors in the corners of rectangles

- There are situations at which XCS is hardly able to learn
  - *Covering challenge*: Covering-Deletion Cycle
  - *Schema challenge*: from over-specification and over-generalisation to maximal general/specific and accurate subspaces
  - *Detrimental forgetting* of rarely sampled niches

Agenda

- Motivation
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
- Example application
- Variants
- **Conclusion**
- Further readings

Summary

This chapter:

- Explained the concept of reinforcement learning.

- Discussed the challenges of learning control strategies in organic systems.

- Presented the Extended Classifier System and its modifications for usage in organic systems.

- Compared concepts such as strength vs. accuracy, exploration vs. exploitation, etc.

- Highlighted how credit assignment is realised and summarised the corresponding challenges for real-world problems.

By now, students should be able to:

- Explain how learning is done in organic systems.

- Summarise the ideas of reinforcement learning in technical applications.

- Outline the structure and process of XCS and its variants.

- Highlight the tasks of the major components and their impact on the learning process.

- Explain how what credit assignment is and how it is realised in OC.

- Discuss the major concepts and customisations used in organic systems.

Beyond this chapter

Reinforcement Learning is a large field

- Check the "Autonomous Learning" lecture in the upcoming term

LCSs are used due to their explainability and generalisation capabilities

- Current trend is to combine e.g. Q-learner with artificial neural networks (solves generalisation, but not explanation problem)

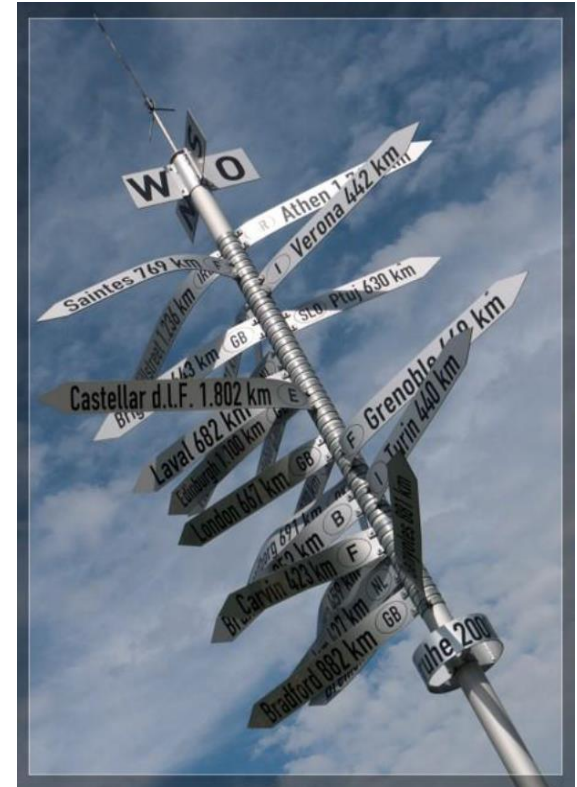Current activities concerning utilisation in intelligent systems

- Proactive knowledge generation
- Combination with other knowledge sources such as humans
- Transfer of knowledge between similar systems
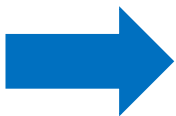
Just ask if you're interested in a topic for a thesis / project work / HiWi position

Agenda

- Motivation
- Reinforcement learning
- Extended Classifier System
- Strength vs. accuracy
- Algorithmic structure of XCS
- Credit assignment
- Real-world applicability
- XCS-O/C
- Example application
- Variants
- Conclusion
- **Further readings**

Further readings

Basic readings

- [Wilson1995]: Wilson, S. W.: Classifier Fitness Based on Accuracy. In: Evolutionary Computation 3 (1995), no. 2, pp. 149-175

- [Wilson1998]: Wilson, S. W.: Generalisation in the XCS Classifier System. Morgan Kaufmann. Genetic Programming 1998: Proceedings of the Third Annual Conference, 1998, pp. 665-674

- [Wilson2000]: Wilson, S.: Get Real! XCS with Continuous-Valued Inputs. In: Lanzi, P.; Stolzmann, W. & Wilson, S. (Eds.): Springer Berlin Heidelberg. LNCS 1813, Learning Classifier Systems, 2000, pp. 209-219

- [Stone2003]: Stone, C. & Bull, L.: For Real! XCS with Continuous-Valued Inputs. In: Evolutionary Computation 11 (2003), no. 3, pp. 298-336

- [Butz2002]: Butz, M. & Wilson, S. W.: An Algorithmic Description of XCS. In: Soft Comput. 6 (2002), no. 3-4, pp. 144-153

Prof. Dr.-Ing. Sven Tomforde / Intelligent Systems group

82

End

- Questions….?