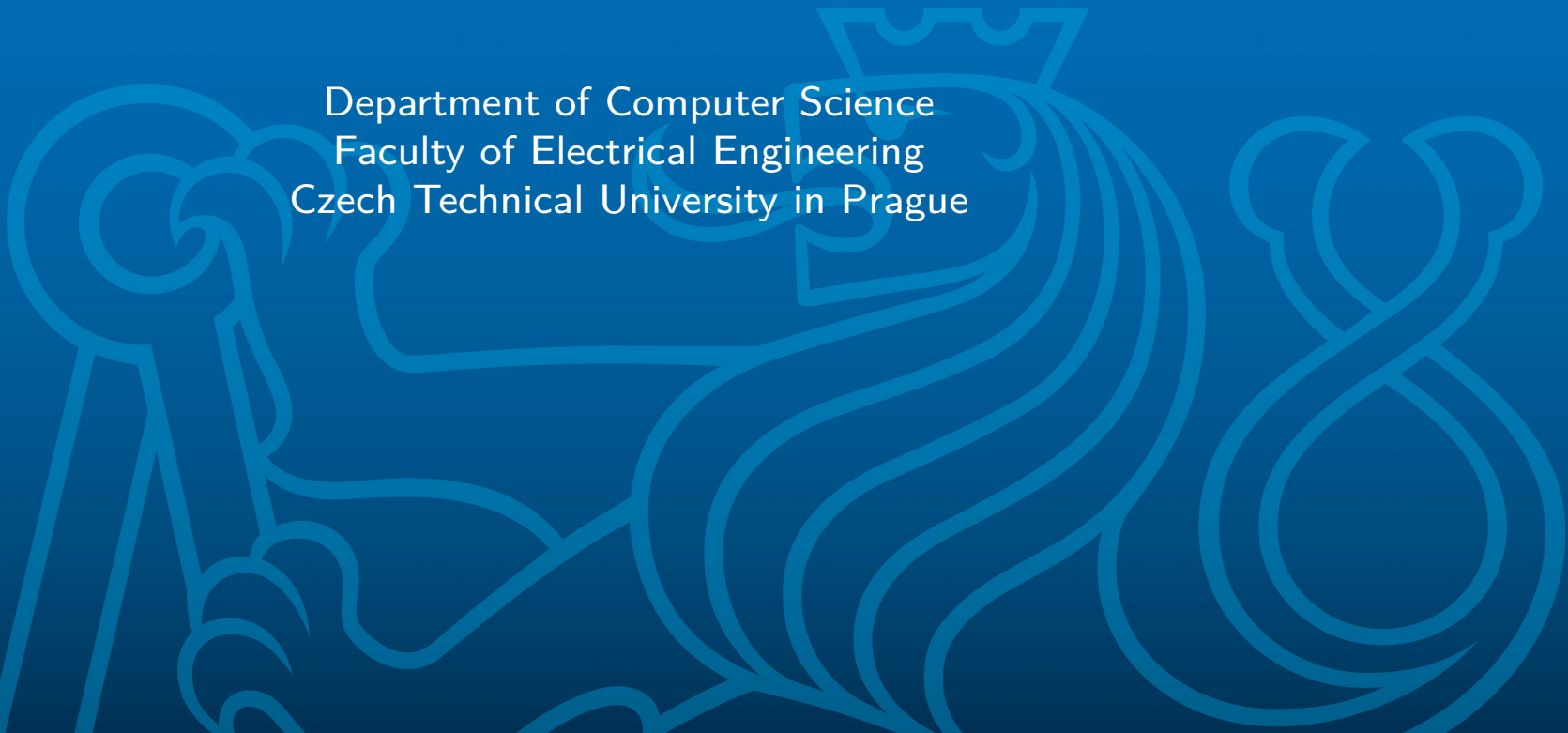
Symbolic Machine Learning

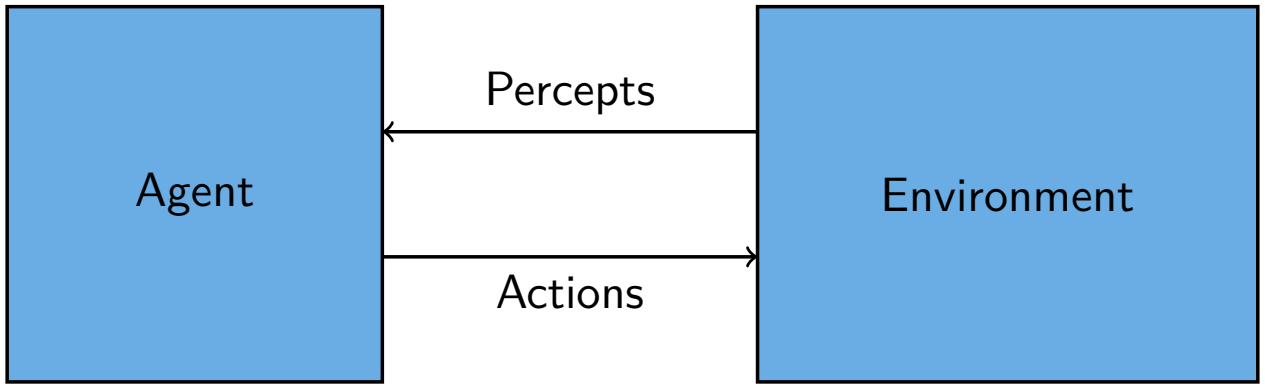
Lecture Slides

Filip Železný

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague



Agent Interacts with Environment



- Discrete *time*

$$k = 1, 2, \ldots$$

- *Percepts*

$$\forall k : x_k \in X$$

- *Actions*

$$\forall k : y_k \in Y$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Histories, Probability, Agent's Policy

History

$$xy_{\leq k} = x_1, y_1, x_2, y_2, \ldots, x_k, y_k$$

has probability

$$\mu(xy_{\leq k}) = \mu(x_1)\mu(y_1|x_1)\mu(x_2|x_1, y_1)\ldots\mu(x_k|xy_{<k})\mu(y_k|x_k, xy_{<k})$$

History influenced by agent's deterministic *policy*

$$y_k = \pi(xy_{<k}, x_k)$$

so

$$\mu(y_k|xy_{<k}, x_k) = \begin{cases} 1 \text{ if } y_k = \pi(xy_{<k}, x_k) \\ 0 \text{ otherwise} \end{cases}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Rewards

Rewards $r_k \in R \subset \mathbf{R}$ are a distinguished part of percepts

$$x_k = (o_k, r_k)$$

Everything else in the percepts are *observations* $o_k \in O$.

Note: r_1 is immaterial.

Define marginals on O and R so that

$$\mu(x_k|xy_{<k}) = \mu(o_k, r_k|xy_{<k}) = \mu_O(o_k|r_k, xy_{<k})\mu_R(r_k|xy_{<k})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Nonsequential Interaction

Nonsequential: a special, simplified case of interaction.

- Observations are i.i.d.:

$$\mu_O(o_k|r_k, xy_{<k}) = \mu_O(o_k)$$

  Note: very strong assumption!

- Reward given by previous observation and the action taken on it:

$$\mu_R(r_k|o_k, xy_{<k}) = \mu_R(r_k|o_{k-1}, y_{k-1})$$

Note that through y_{k-1}, r_k still depends on the entire preceding history.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Batch Learning

Batch: a non-sequential interaction with two successive phases:

- *learning (training, exploration)* at $k = 1, 2, \ldots K$
- *action (testing, exploitation)* at $k = K + 1, K + 2, \ldots$

Assumption: no change of policy in the action phase, so for $k > K$

$$y_k = \pi(xy_{\leq K}, o_k)$$

A model for most "data-mining" scenarios.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Rewards in Batch Learning

In the action phase ($k > K$)

$$\mu_R(r_k|o_{k-1}, y_{k-1}) = \mu_R(r_k|o_{k-1}, xy_{\leq K})$$

Batch case being non-sequential, o_{k-1} is i.i.d. from $\mu_O(o_{k-1})$ that does not depend on k. So r_k is i.i.d from

$$\mu_R(r_k|xy_{\leq K})$$

where

$$\mu_R(r_k|xy_{\leq K}) = \sum_{o_{k-1} \in O} \mu_O(o_{k-1}) \mu_R(r_k|o_{k-1}, xy_{\leq K})$$

So a learning history $xy_{\leq K}$ determines the reward distribution (and the expected reward) in the action phase.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Agent's Goal

Given a *finite* time horizon $k' \in N$ of interest, choose $y_1, y_2, \ldots, y_{k'-1}$ to maximize expected cumulative reward

$$\sum_{r_{\leq k'}} \mu_R(r_{\leq k'}|y_{\leq k'})(r_1 + r_2 + \ldots + r_{k'})$$

If horizon *infinite*, use the discount sequence δ_k such that $\sum_{i=1}^{\infty} \delta_i < \infty$ and maximize

$$\lim_{k' \to \infty} \sum_{r_{\leq k'}} \mu_R(r_{\leq k'}|y_{\leq k'}) \sum_{k=1}^{k'} r_k \delta_k$$

In the *batch* scenario, simply maximize the expected reward in the action phase

$$\sum_{r_k \in R} \mu_R(r_k|xy_{\leq K}) r_k$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Environment: State-Based Description

Assumption: environment fully described by its *state* $e_k \in E$ at each k.

Generation of percepts $x_k = (o_k, r_k)$:

- observations: $\mu_o(o_k|e_k, y_{k-1})$
- rewards: $\mu_r(r_k|e_{k-1}, y_{k-1})$

Assume $|E| <$ some constant that does not depend on k. So the environment has a *finite memory* and percepts do not depend on the entire history.

State gets *updated stochastically* at each $k > 1$ according to distribution

$$\mathcal{E}(e_k|e_{k-1}, y_{k-1})$$

Note: x_{k-1} not assumed among the conditions.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Agent: State-Based Description

Assumption: agent fully described by its *state* $a_k \in A$ at each k.

Generation of actions y_k:

$$y_k = \pi(a_k, o_k)$$

Assume $|A| <$ some constant that does not depend on k. So the agent has a *finite memory* and percepts do not depend on the entire history.

State gets *updated deterministically* at each $k > 1$ according to function

$$a_k = \mathcal{A}(a_{k-1}, x_k)$$

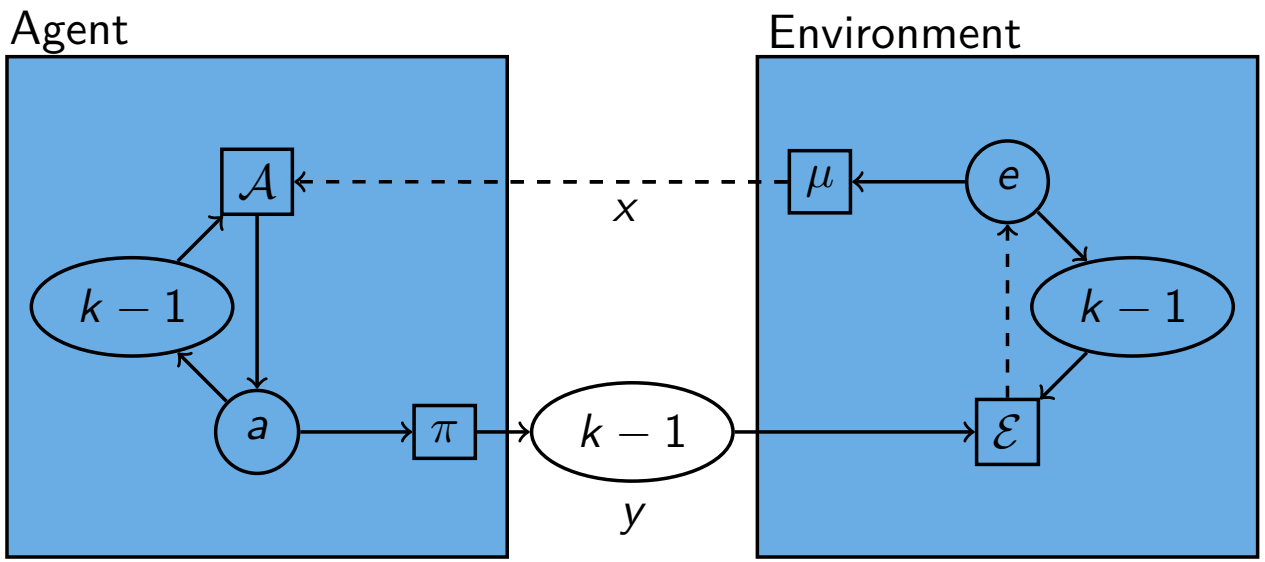
Note: x_k in the argument, not x_{k-1}. Recall the 'clock' ticks after agent's action.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

State-Based Interaction Model

Although we will not do it, o_k (resp. e_{k-1} and y_{k-1}) can be formally remembered as part of a_k (resp. e_k), allowing a simplified notation:

Agent		Environment	
actions:	$y = \pi(a_k)$	percepts:	$x_k \sim \mu(x_k \mid e_k)$
state update:	$a_k = \mathcal{A}(a_{k-1}, x_k)$	state update:	$e_k \sim \mathcal{E}(e_k \mid e_{k-1}, y_{k-1})$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Non-sequential and Batch with States

Non-sequential assumption: e_k sampled i.i.d. from

$$\mathcal{E}(e_k)$$

that does not depend on k, so observations are also i.i.d from

$$\mu_O(o_k) = \sum_{e_k \in E} \mu_O(o_k|e_k)\mathcal{E}(e_k)$$

Rewards in the action phase of the *batch scenario* are also i.i.d. from distribution $\mu_R(r_k|a_K)$ depending only on the last agent's state a_K in the learning phase (see textbook for derivation).

Agent should reach state maximizing the expected reward

$$\sum_{r_k \in R} \mu_R(r_k|a_K)r_k$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

On-line Concept Learning: Example

experiment number k	apoptosis initiated e_k	protein 1 present o_k^1	protein 1 active o_k^2	protein 2 present o_k^3	protein 2 active o_k^4	prediction y_k	reward r_k
1	0	0	0	0	0	0	0
2	0	0	0	1	0	1	0
3	0	1	0	0	0	1	-1
4	1	1	1	0	0	0	-1
5	0	1	0	1	1	0	-1
6	1	1	1	1	0	1	0
7	1	1	1	0	0	1	0
8	0	1	0	1	1	0	0
(etc.)							

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Learning Assumptions

Assumption: "*observation identifies state*", i.e. for any observation o, there is at most one state e such that

$$\mu_O(o|e) > 0$$

Further simplifying assumptions (for convenience):

- Binary environment states: $E = \{0, 1\}$
- Observations are binary tuples $O = \{0, 1\}^n$ $(n \in N)$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Decisions and Rewards in Concept Learning

- Decisions are guesses about states, so $Y = E$
- Agent is punished for the wrong guess

$$r_{k+1} = \begin{cases} 0 \text{ if } e_k = y_k \\ -1 \text{ otherwise} \end{cases}$$

- Note: this is a special case of the *loss function*

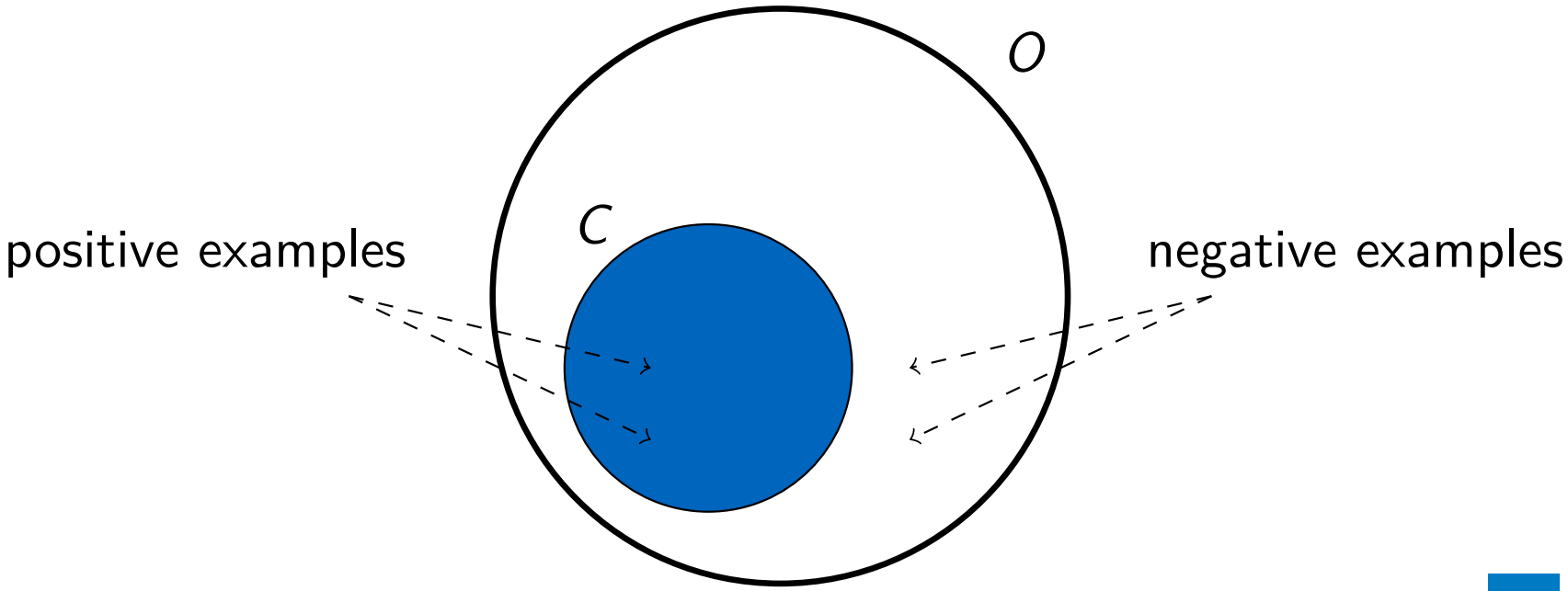
$$r_{k+1} = -L(e_k, y_k)$$

called *unit loss*. We will only work with unit loss.

Concept

- Environment (target) concept:

$$C = \{ o \in O \mid \mu_O(o|1) > 0 \}$$



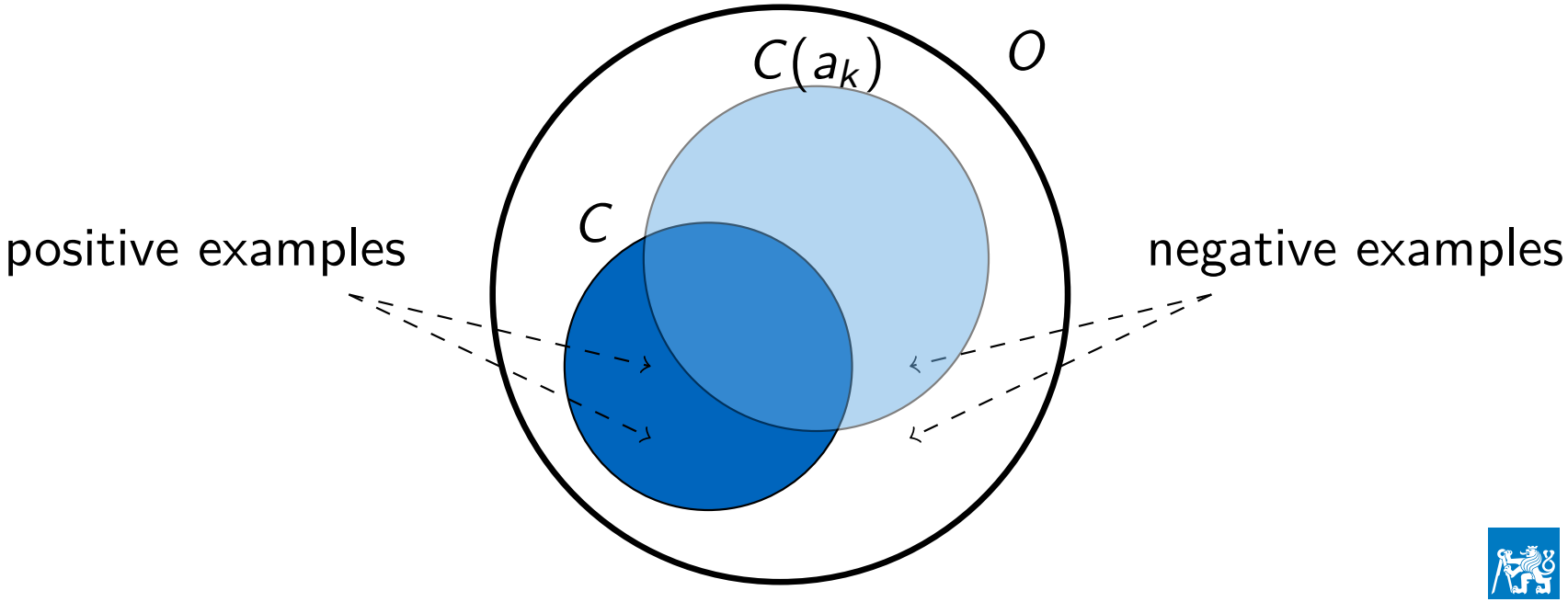
Concept

- Environment (target) concept:

$$C = \{ o \in O \mid \mu_O(o|1) > 0 \}$$

- Agent's (hypothesized) concept at time k:

$$C(a_k) = \{ o \in O \mid \pi(a_k, o_k) = 1 \}$$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Designing a Concept Learning Agent

Note on concept-learning terminology:

- C - *positive class*, $O \setminus C$ - *negative class*
- e_k is the *class label* of o_k.
- "observations" $\sim$ "examples"

First thought:

- At $k+1$, agent knows the class of o_k (it is $|y_k + r_{k+1}|$).
- It can remember a finite set of labeled examples in its state.
- If state size overflown, e.g. forget the oldest example.
- For a unseen observation, give the prevailing class among the most similar (e.g. Hamming distance) observations remembered.
- This is known as nearest-neighbor classification and is barely learning.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Agent with a Hypothesis

Just like a human, the agent should learn a *hypothesis* by which it will decide.

At time k, agent has hypothesis h_k stored in its state a_k.

As h_k is the only part of a_k influencing the decisions, we may write $\pi(h_k, o_k)$ and $C(h_k)$ instead of $\pi(a_k, o_k)$ and $C(a_k)$.

h_k can be 'anything' (a set of rules, equations, a graph, ...) as long as π can produce a decision out of it.

Updating a Hypothesis

When $r_k = -1$, the agent knows it made a wrong decision at $k-1$ and should change its hypothesis.

Most of the times, it needs to know o_{k-1} to do it. So the last observation will be memorized as part of the state

$$a_k = (o_k, h_k)$$

and the update function $\mathcal{A}(a_{k-1}, x_k)$ will be instantiated to

$$\mathcal{A}\left(\left(o_{k-1}, h_{k-1}\right), \left(o_k, r_k\right)\right) = (o_k, h_k)$$

Agent's intelligence rests in the way it updates towards h_k given h_{k-1} and o_{k-1}, whenever $r_k = -1$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent

Let us design an agent whose hypothesis is a propositional *logic formula*.

$$y_k = \pi(h_k, o_k) = \begin{cases} 1 \text{ if } o_k \models h_k \\ 0 \text{ otherwise} \end{cases}$$

h_k is made of propositional variables $p_1, p_2, \ldots p_n$ and the binary tuple o_k is a truth-value assignment to them.

More specifically, we assume h_k to be a *conjunction*.

The agent will start with the conjunction of all possible literals and on each error, it will delete all inconsistent literals.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent: Update Step Example

For $n = 4$:

$$
\begin{aligned}
h_1 = \quad & p_1 \wedge \neg p_1 \quad \wedge p_2 \wedge \neg p_2 \quad \wedge p_3 \wedge \neg p_3 \quad \wedge p_4 \wedge \neg p_4 \\
o_1 = (\quad & 1, \quad 1, \quad 0, \quad 0) \\
y_1 = \quad & 0
\end{aligned}
$$

$$
\begin{aligned}
r_2 &= -1 \\
h_2 = \quad & p_1 \quad \wedge p_2 \quad \wedge \neg p_3 \quad \wedge \neg p_4
\end{aligned}
$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent: Update Step

Formally:

$$h_k = \begin{cases} h_{k-1} \text{ if } r_k = 0 \\ \text{delete}(h_{k-1}, o_{k-1}) \text{ otherwise} \end{cases}$$

where

$$\text{delete}\left(\bigwedge_{i \in I} p_i \bigwedge_{j \in J} \neg p_j, (o^1, o^2, \ldots, o^n)\right) = \bigwedge_{\substack{i \in I \\ o^i = 1}} p_i \bigwedge_{\substack{i \in I \\ o^j = 0}} \neg p_j$$

Generalizing Agent: How Well Does it Learn?

Assume a perfect conjunction h^* exists: $C(h^*) = C$, i.e. the target concept can be expressed through a conjunction.

Observations (see textbook for more rigor):

1. All literals of h^* are consistent with an example iff the example is positive.
2. When a negative example is misclassified, the hypothesis has all literals consistent with it.
3. No literal that is in h^* is removed from the agent's hypothesis (from 1 and 2). Since h_1 contains *all* literals, we have $h_k \supseteq h^*$, $\forall k \in N$. This means that mistakes are made only on positive examples.
4. From 3: on each mistake, at least one literal in the hypothesis is inconsistent and gets deleted.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent: Mistake Bound

Since at least one literals is deleted on each mistake, and the initial hypothesis has $2n$ literals, *no more than $2n$ mistakes* are made before $h_k = h^*$.

So the cumulative reward is

$$\sum_{k=1}^{k'} r_k \geq -2n$$

for an arbitrary horizon $k' \in N$.

(Also a lower bound on the expected cumulative reward for any horizon)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent for Disjunctions

Any disjunction can be converted to a conjunction of same size (up to additive/multiplicative factor):

$$p_1 \vee p_2 \vee \ldots \vee p_n = \neg\left(\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n\right)$$

So just learn a conjunction (as in the parentheses) with the same algorithm except with inverted actions ($1 - y_k$ instead of y_k).

Finally, produce a disjunction by going from the RHS to the LHS.

Generalizing Agent for s-CNF, s-DNF

An *s-clause* ($s \in N$) is a disjunction of at most s literals. An *s-CNF* is a conjunction of s-clauses.

With n variables, $n' = \binom{2n}{s}$ different s-clauses can be made. Assign a new propositional variable p'_i $(1 \leq i \leq n')$ to each of them.

Learn a conjunction with the new variables, determining the truth value of each p'_i as that of the assigned clause whose variables are valuated by the examples.

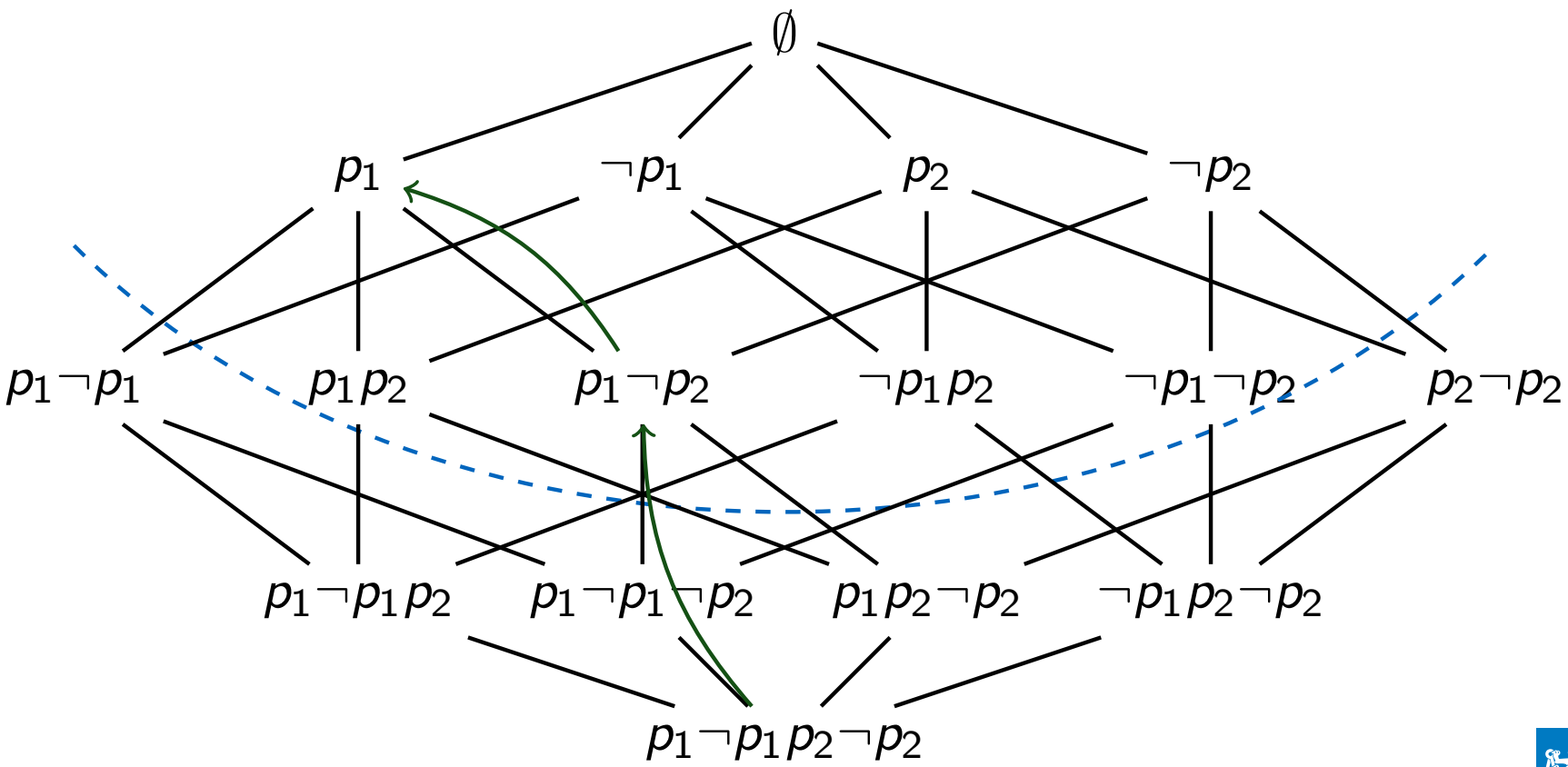
Finally produce the learned s-CNF by replacing the variables in the conjunction with their assigned clauses.

Method efficient if s is a small constant. s-DNF (disjunctions of *s-terms*) learnable analogically.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Subsumption Lattice

We call the agent generalizing because $\forall k,\ h_{k+1} \subseteq h_k$. We say that conjunction h_{k+1} *subsumes* conjunction h_k and learning can be viewed as search in a *subsumption lattice*.



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Subsumption and Implication

For *conjunctions*, subsumption implies logical entailment:

$$\text{if } h \subseteq h' \text{ then } h' \vdash h$$

But the reverse implication is not true for *self-resolving* conjunctions on the LHS. E.g. $p_1 \wedge \neg p_1 \vdash p_2$. In propositional logic, self-resolving conjunctions are *contradictions*.

For *clauses* (i.e. disjunctions), the implication is different:

$$\text{if } h \subseteq h' \text{ then } h \vdash h'$$

Again, the reverse implication is not true for *self-resolving* clauses on the RHS. E.g. $p_2 \vdash p_1 \vee \neg p_1$. In propositional logic, self-resolving clauses are *tautologies*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE