This textbook contains all material needed for the Symbolic Machine Learning course exam. The slide sets cross-referenced in Sections 5 and 6 are considered part of this textbook. The teachers are ready to answer your questions. Questions regarding Sections 1-4 should be addressed to Filip Železný , questions regarding Sections 5 and 6 should be addressed to Jiří Kléma.

# Symbolic Machine Learning

Filip Železný and Jiří Kléma

## Contents

# 1 A General Framework

## 1.1 Percepts and Actions

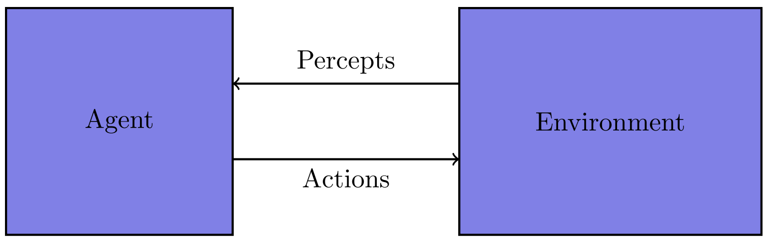



Figure 1: The basic situation under study.

- Discrete *time*
$$k = 1, 2, \ldots$$
- *Percepts*
$$\forall k : x_k \in X$$
- *Actions*
$$\forall k : y_k \in Y$$

$X$ and $Y$ are finite.

A *history* is a sequence of alternating percepts and actions, i.e,

$$x_1, y_1, x_2, y_2, \ldots, x_k, y_k$$

and is denoted as $xy_{\leq k}$. Similarly, $xy_{<k} = x_1, y_1, x_2, y_2, \ldots, x_{k-1}, y_{k-1}$. There is a probability distribution $\mu$ on histories

$$\mu(xy_{\leq k}) = \mu(x_1)\mu(y_1|x_1)\mu(x_2|x_1, y_1)\ldots\mu(x_k|xy_{<k})\mu(y_k|x_k, xy_{<k}) \tag{1}$$

After the initial 'kick-off' $x_1$ from the environment distributed according to $\mu(x_1)$, any percept $x_k$ generated by the environment at time $k$ depends on the entire preceding history $xy_{<k}$ according to

$$\mu(x_k|xy_{<k}) \tag{2}$$

Actions $y_k$ are determined by agent's decision *policy* which also depends on the history as well as the current percept and are distributed according to

$\mu(y_k|x_k, xy_{<k})$. We will assume that the policy is *deterministic*. Thus we identify the policy with function $\pi : (X \times Y)^* \times X \to Y$, so

$$y_k = \pi(xy_{<k}, x_k) \tag{3}$$

This means that $\mu(y_k|xy_{<k}, x_k) = 1$ for $y_k = \pi(xy_{<k}, x_k)$ and 0 otherwise.

The following diagram illustrates the influences between the introduced variables.

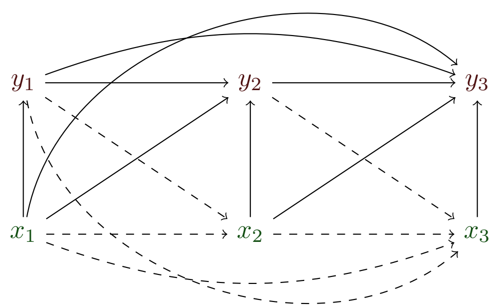



Figure 2: Influence diagram for actions $y_k$ and percepts $x_k$ for $1 \le k \le 3$ with full lines indicating deterministic influences (via $\pi$) and dashed lines showing probabilistic influences (via $\mu$).

While we have yet to define what goals the agent should achieve through interaction with the environment, obviously some histories will be "better" than others in terms of the goal achievement. To maximize the probability (1) of good histories, the agent cannot influence the conditional probability (2), which is inherent to the environment, but it can follow a good policy (3). However, the effect of actions proposed by the policy depends on (2) which is generally not known to the agent. So the agent needs to recognize the environment by experimenting with it. This is formally reflected by (3) where action $y_k$ depends not only on the current percept $x_k$ but also on the history $xy_{<k}$. So the agent will generally make different decisions $y_k \neq y_{k'}$ for $k > k'$ even if $x_k = x_{k'}$ because the experience $xy_{<k}$ at time $k$ is larger than experience $xy_{<k'}$ at time $k'$. This is our first reflection of *learning*.

How does the agent know how well it is doing? This information comes from the environment through a specially distinguished part of the percepts, called *rewards*. The remaining part of each percept contains *observations*. Formally, $X = O \times R$, $o_k \in O$, $r_k \in R \subset \Re$, so

$$x_k = (o_k, r_k) \tag{4}$$

Since $X$ is assumed finite, it follows that rewards have their finite minimum and maximum.

The probability of $x_k$ in (2) can be written in terms of the marginals $\mu_O$ and $\mu_R$

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

which also makes it clear that $o_k$ and $r_k$ are in general not mutually independent, even if conditioned on $xy_{<k}$.

### 1.2 Nonsequential Cases

Scenarios where current percepts depend on the history of previous percepts and actions are called *sequential*. The framework described so far is maximally general in that dependence is assumed on the entire history from $k = 1$ on. On the other extreme are *nonsequential* scenarios. Here, observations are independent of the history as well as the current reward, i.e.

$$\mu_O(o_k|r_k, xy_{<k}) = \mu_O(o_k) \tag{5}$$

and thus $o_1, o_2, \ldots$ are mutually independent random variables sampled from the same distribution $\mu_O$ (they are "i.i.d.").

Rewards in the nonsequential case are assumed to depend only the immediately preceding observation and the action taken on it, i.e.

$$\mu_R(r_k|o_k, xy_{<k}) = \mu_R(r_k|o_{k-1}, y_{k-1}) \tag{6}$$

however, since $y_{k-1}$ is functionally determined by the history $xy_{<k-1}$ and percept $x_{k-1} = (o_{k-1}, r_{k-1})$ through (3), we may rewrite (6) as

$$\mu_R(r_k|o_{k-1}, r_{k-1}, xy_{<k-1}) \tag{7}$$

which makes it clear that reward $r_k$ depends on previous rewards, and thus rewards $r_1, r_2, \ldots$ are not i.i.d.. This is natural since if they were, it would mean the agent never improves its performance.

### 1.3 Batch Learning

We will also consider a specific yet important nonsequential case called *batch learning* consisting of two phases switching right after time $K$

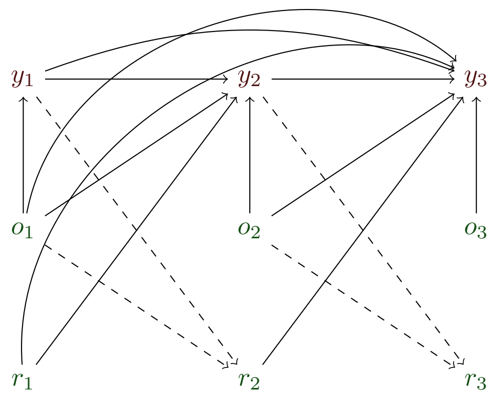



Figure 3: Influence diagram for actions $y_k$, observations $o_k$, and rewards $r_k$ for $1 \leq k \leq 3$ with full lines indicating deterministic influences (via $\pi$) and dashed lines showing probabilistic influences (via $\mu$) in the nonsequential case.

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

In the action phase, the agent no longer changes its decision making policy, so

$$\text{if } k, k' > K \text{ and } x_k = x_{k'} \text{ then } y_k = y_{k'} \tag{8}$$

and ignores rewards. So the action proposed by the policy depends only on the current observation and the history only up to time $K$. So for $k > K$, (3) changes here into

$$y_k = \pi(xy_{\leq K}, o_k) \tag{9}$$

and (6, 7) change into

$$\mu_R(r_k | o_{k-1}, y_{k-1}) = \mu_R(r_k | o_{k-1}, xy_{\leq K}) \tag{10}$$

because due to (9), $y_{k-1}$ is determined by $o_{k-1}$ and $xy_{\leq K}$. The observation $o_{k-1}$ does not depend on rewards due to (5). So reward $r_k$ does not depend on previous rewards $r_{k'}$, $k > k' > K$. Another way to say this is that rewards in the action phase are conditionally independent of each other, given the learning phase history:

$$\mu_R(r_k, r_{k'} | xy_{<K}) = \mu_R(r_k | xy_{<K}) \mu_R(r_{k'} | xy_{<K}) \tag{11}$$

The following figure illustrates the batch-learning situation.

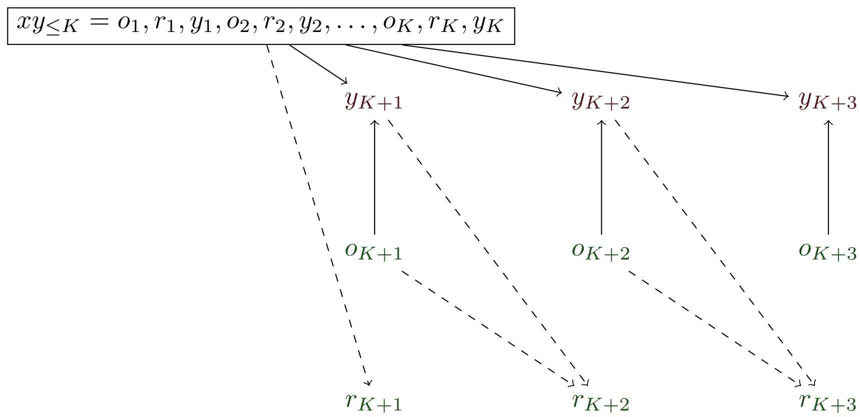



Figure 4: Influence diagram for actions $y_k$, observations $o_k$, and rewards $r_k$ in the action phase ($k > K$) of batch learning with full lines indicating deterministic influences (via $\pi$) and dashed lines showing probabilistic influences (via $\mu$). The top row indicates the influence of the learning phase on the agent's decisions in the action phase.

We can further express the distribution of $r_k$ ($\forall k > K$) without conditioning on the observations, which are i.i.d. by (5)

$$\mu_R(r_k|xy_{\leq K}) = \sum_{o_{k-1} \in O} \mu_O(o_{k-1})\mu_R(r_k|o_{k-1}, xy_{\leq K}) \tag{12}$$

So rewards in the action phase are i.i.d. according to the above distribution conditioned only on the history of the learning phase.

## 1.4 Rewards and Goals

It has been obvious that the agent's goal is to maximize rewards. Here we formalize this goal. Since rewards come at each point of the history, we want the agent to maximize their sum up to a finite time *horizon* $m \in N$

$$r_1 + r_2 + \ldots + r_m$$

or, more generally, maximize the *discounted* sum

$$\sum_{k=1}^{\infty} r_k \gamma_k$$

where $\forall k : \gamma_k \geq 0$ and $\sum_{i=1}^{\infty} \gamma_i < \infty$, so the above sum converges.

But since rewards are probabilistic, the agent should choose a sequence $y_{\leq m}$ of actions leading to a high *expected* cumulative reward

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

or, in the discounted case

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

where the first sum in both cases goes over all possible reward sequences $r_{\leq m}$ (since $R$ and $m$ are finite, there is a finite number of them).

However, for the specific case of *batch learning*, we establish a more appropriate learning goal. First, we do not care about maximizing rewards in the *learning phase* as the purpose of this phase is to probe the environment even at the price of possibly poor rewards. Second, in the *action phase* after time $K$, the rewards $r_k$, $k > K$ are sampled independently from the same distribution (12) so we can simply maximize their expectation with respect to this distribution

$$\sum_{r_k \in R} \mu_R(r_k|xy_{\leq K}) r_k \tag{13}$$

It is again obvious from the formula that the expected reward only depends on the learning phase history $xy_{\leq K}$, after which the agent no longer changes its action policy. Note also that the batch learning scenario allowed us to define an objective (13) without the need to choose the parameters $m$ or $\gamma_k$ $(k = 1, 2, \ldots)$ needed in the sequential scenario.

## 1.5 Environment States

With the exception of the non-sequential scenario, our framework has been very general in that percepts $x_k$ generally depend on entire histories $xy_{<k}$. In the real world, many histories may be equivalent, i.e. leading to the same probabilities of $x_k$ conditioned on action $y_{k-1}$. This can be formalized through the notion of *environment state* $s_k \in S$ at time $k$.

For generality, let us first assume that the state is probabilistically established by the preceding state, the last percept, and the last action through the following state *update* distribution

$$\mathcal{S}(s_k|s_{k-1}, x_{k-1}, y_{k-1}) \tag{14}$$

and that this state generates the current percept

$$\mu(x_k|s_k) \tag{15}$$

This modification does not lessen the generality of the framework if we allow $S$ to be infinite as then there could simply exist a distinct state for each possible history (there is an infinite number of possible histories for unbounded $k$). Indeed, if one instantiates the distribution (14) to the functional dependence

$$s_k = s_{k-1} \parallel (x_{k-1}, y_{k-1}) \tag{16}$$

where $\parallel$ denotes concatenation, $s_k$ will simply collect the entire history and its occurrence in (15) would be just a different name for $xy_{<k}$ in (2). However, we will make an important assumption, which will significantly simplify the framework, that the number of possible states is bounded by a finite constant $S_{\max} \in \Re$ which does not depend on $k$

$$|S| < S_{\max} \tag{17}$$

In practical tasks, there will be far fewer states than possible histories.

We can afford further simplifying assumptions under which the state-based framework will still encompass the learning scenarios we are going to elaborate. First, we will assume that the influence between environment states and the emitted percepts are single-directional. In particular, the percepts depend on states by (15) but not vice versa, so we remove $x_{k-1}$ from (14)

$$\mathcal{S}(s_k|s_{k-1}, x_{k-1}, y_{k-1}) = \mathcal{S}(s_k|s_{k-1}, y_{k-1}) \tag{18}$$

As a consequence, the state cannot collect the history of percepts as in (16) but it can still collect the history of actions

$$s_k = s_{k-1} \parallel y_{k-1} \tag{19}$$

If the state evolves according to (19) then the percept in (15) depends on all historical states $s_{k-1}, s_{k-2}, \ldots, s_1$ as well as all historical actions $y_{k-1}, y_{k-2}, \ldots y_1$ embedded in them, and not on any other factors. So instead of assuming the specific update rule (19), we may equivalently assume that the state evolves in any other way but the state-percept dependencies are preserved, so that percepts are sampled according to

$$\mu(x_k|s_k, s_{k-1}, s_{k-2}, \ldots, s_1, y_{k-1}, y_{k-2}, \ldots, y_1) \tag{20}$$

Our simplification plan is to remove some of the dependencies above. We will do it differently for the two components of the percept, i.e. the observations

$$\mu_o(o_k|r_k, s_k, s_{k-1}, s_{k-2}, \ldots, s_1, y_{k-1}, y_{k-2}, \ldots, y_1) \tag{21}$$

and the rewards

$$\mu_r(r_k|o_k, s_k, s_{k-1}, s_{k-2}, \ldots, s_1, y_{k-1}, y_{k-2}, \ldots, y_1) \tag{22}$$

In particular, the observation will depend only on the current state and the last agent's action

$$\mu_o(o_k|s_k, y_{k-1}) \tag{23}$$

and the reward will depend on the last state and the action taken immediately on it

$$\mu_r(r_k|s_{k-1}, y_{k-1}) \tag{24}$$

## 1.6 Agent Hypotheses

A reasoning similar to the previous section applies to the agent, whose actions generally depend on the entire history as in (3). Again, many histories can lead to the same mapping from percepts to actions, for example because the agent has built the same hypothesis about the environment throughout the different histories. So analogically to the environmental states, we introduce the notion of agent's *hypothesis* $h_k \in H$. Since we work with deterministic agents, we will assume that the hypothesis is updated given the current percept through a functional prescription

$$h_k = \mathcal{H}(h_{k-1}, x_k) \tag{25}$$

and instead of (3), we will assume that actions depend on the (updated) hypothesis rather than the history, and the current observation

$$y_k = \pi(h_k, o_k) \tag{26}$$

Unlike in (3), explicit dependence on $x_k$ is no longer needed in (26) as the latter can always be stored as part of $h_k$ in (25). However, we do keep the $o_k$ component of $x_k$ as an argument of $\pi$ because this will allow us to describe conveniently cases where the agent's hypothesis is kept constant and the actions depend only on their immediately preceding observation. This will in particular include the batch-learning case discussed below in the present context of state- and hypothesis-based descriptions.

Again, we will postulate that

$$|H| < H_{\max} \tag{27}$$

for some constant $H_{\max}$ that does not depend on $k$.

The formalization using enviroment states and agent hypotheses results in the agent and environment structures depicted in Fig. 5. The diagram of variable influences is shown in Fig. 6.

The agent hypothesis $h_k$ has a very natural interpretation as it corresponds to the agent's model of the environment at time $k$, whereas $\pi$ is the the interpreter of the model.[1] For example, $h_k$ may encode a set of logical rules, and $\pi$ may

[1] We might as well call $h_k$ a *model* rather than a *hypothesis* but that would cause terminology clash in cases where the $h_k$ is expressed in the formalism of logic, where the word *model* is already established and has a different meaning.

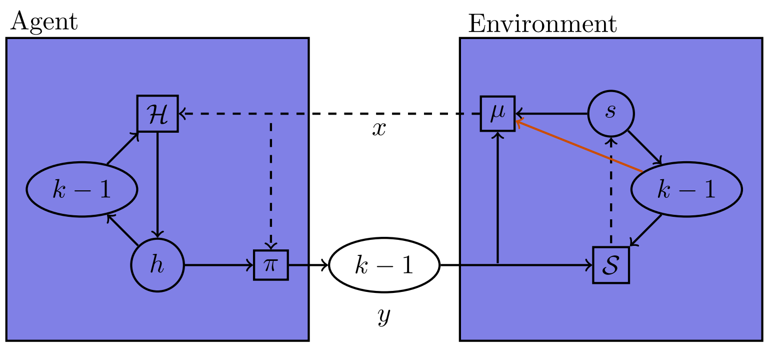



Figure 5: The state-based scheme of agent-environment interaction. Full and dashed lines denote functional and probabilistic influences, respectively. The $k-1$ nodes denote a one-step time lag. The highlighted dependence is only relevant for the reward part $r$ of the percept $x$ generated by $\mu$; if the diagram only captured observations $o$ and actions $y$, it would not contain this dependence and thus would be symmetric.

be a logical prover deriving actions as logical consequences of the rules. Since the hypothesis description has to fit in a finitely bounded memory, there can be only a finite number of different hypotheses. Therefore, the assumption in (27) is well justified.

The history of percepts and actions (in combination with the current percept) is obviously informative for updating the hypothesis so it seems the hypothesis update in (25) should also include previous percepts $x_{k-1}, x_{k-2}, \ldots$ and actions $y_{k-1}, y_{k-2}, \ldots$ as arguments. However, this is not necessary as the update function $\mathcal{H}$ in (25) can always be made to store any finite number of percepts and previous hypotheses in the memory, i.e. as part $h_k$, because they are inputs to the update step (25). But also any historical action $y_{k'}$, $k' < k$ can be retrieved by first retrieving $h_{k'}$ from the memory and then using (25). This is possible because $\pi$ is deterministic and can be simulated by $\mathcal{H}$.

## 1.7 Nonsequential and Batch Cases with States and Hypotheses

Just like in the framework using entire histories, also with the formulation based on states and hypotheses the situation simplifies a lot in the *nonsequential case*. Here, the environment has no memory at all so the conditioning factors in (18) and states are updated by i.i.d. sampling from the marginal distribution

$$\mathcal{S}(s_k) \tag{28}$$

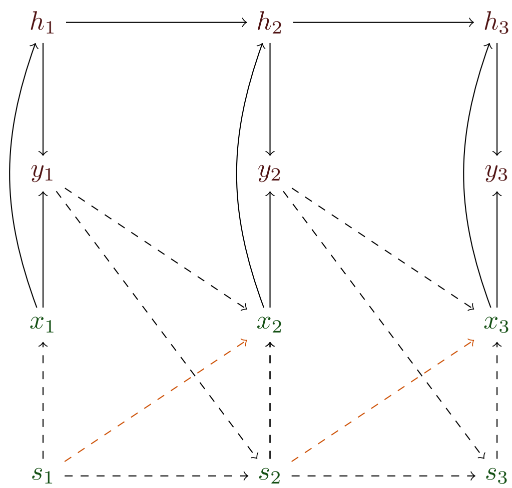



Figure 6: Influence diagram for states $h_k$, actions $y_k$ and percepts $x_k$ for $1 \leq k \leq 3$ with full lines indicating deterministic influences (via $\pi$ and $\mathcal{H}$) and dashed lines showing probabilistic influences (via $\mu$ and $\mathcal{S}$). The highlighted dependencies are only needed for generating the reward part $r_k$ of the percepts $x_k$.

Furthermore, observations $o_k$ no longer depend on agent's last action as in (23) so they are sampled from

$$\mu_o(o_k|s_k) \tag{29}$$

Since $s_k$'s are i.i.d., the $o_k$'s are also i.i.d.

Rewards, given by (24), are however still generally non-i.i.d. as they depend on the agent's actions, which in turn depend on the evolving agent's hypothesis. Fig. 7 shows the complete set of influences in the nonsequential case.

A further simplification comes in the special *batch-learning* scenario of the non-sequential case. While in the learning phase of the latter, the agent uses the update rule (25), in the action phase it no longer updates the hypothesis, so

$$h_k = h_K, \forall k \geq K \tag{30}$$

This is illustrated in Fig. 8. Special attention is needed regarding the variables at time $K$. Reward $r_K$ (part of percept $x_K$) is the last training reward, according to which the last update is conducted towards the final $h_K$. Observation $o_K$ (another part of percept $x_K$) is, however, the first *testing* observation.

For $k > K$, $y_{k-1}$ is fully determined by $o_{k-1}$ and $h_K$ through (26) in which

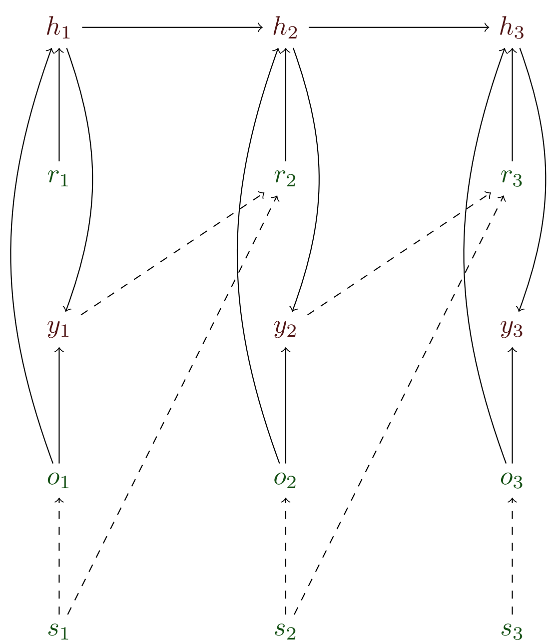



Figure 7: Influence diagram for hypothesis $h_k$, actions $y_k$, observations $o_k$, and rewards $r_k$ for $1 \leq k \leq 3$ with full lines corresponding to deterministic influences (via $\pi$ and $\mathcal{H}$) and dashed lines showing probabilistic influences (via $\mu$ and $\mathcal{S}$) in the nonsequential case.

$h_{k-1} = h_K$. So we can rewrite (24) into

$$\mu_r(r_k|s_{k-1}, o_{k-1}, h_K) \tag{31}$$

and further express

$$\mu_r(r_k|h_K) = \sum_{s_{k-1}\in O} \sum_{o_{k-1}\in O} \mu_r(r_k|h_K, s_{k-1}, o_{k-1})\mu_o(o_{k-1}|s_{k-1})\mathcal{S}(s_{k-1}) \tag{32}$$

where $\mu_o$ and $\mathcal{S}$, i.e. (29) and (28), are independent of $k$. So in the testing phase, rewards $r_k$ are i.i.d. according to the distribution $\mu_r(r_k|h_K)$ depending only on the learned hypothesis $h_K$. This is analogical to the state-free formulation (12). Similarly to (13), an agent operating in the batch-learning scenario with states will be assessed by the expected reward in the testing phase

$$\sum_{r_k\in R} \mu_R(r_k|h_K)r_k \tag{33}$$

and should find a hypothesis $h_K$ maximizing this quantity.

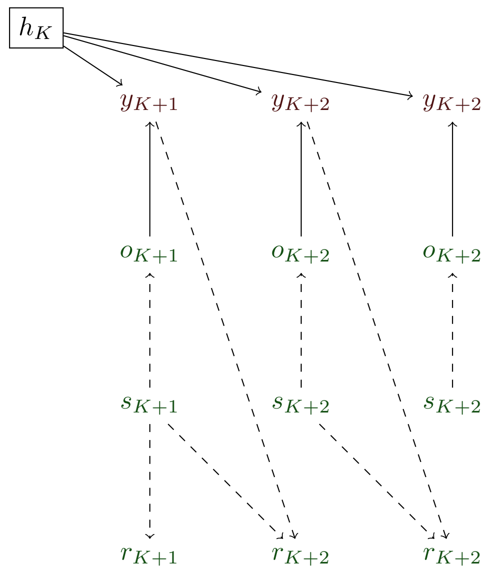



Figure 8: Influence diagram for actions $y_k$, observations $o_k$, states $s_k$, and rewards $r_k$ in the action phase ($k > K$) of batch learning with full lines indicating deterministic influences (via $\pi$) and dashed lines showing probabilistic influences (via $\mu$). The top row indicates the influence of the agent's last hypothesis learned in the learning phase on the action phase. The dependence of $r_{K+1}$ on $s_K$ and $y_K$ is not shown.

## 1.8 Prior Knowledge

- *Implicit*: the setting of $H$ ("hard bias") and $\mathcal{H}$ ("soft bias")
- *Explicit*: the setting of $h_1$ ("background knowledge")

## 1.9 Hypothesis Representations

See Fig. 9.

## 1.10 Learning Scenarios

1. on-line concept learning
2. batch concept learning
3. query-based and active learning (not covered here)

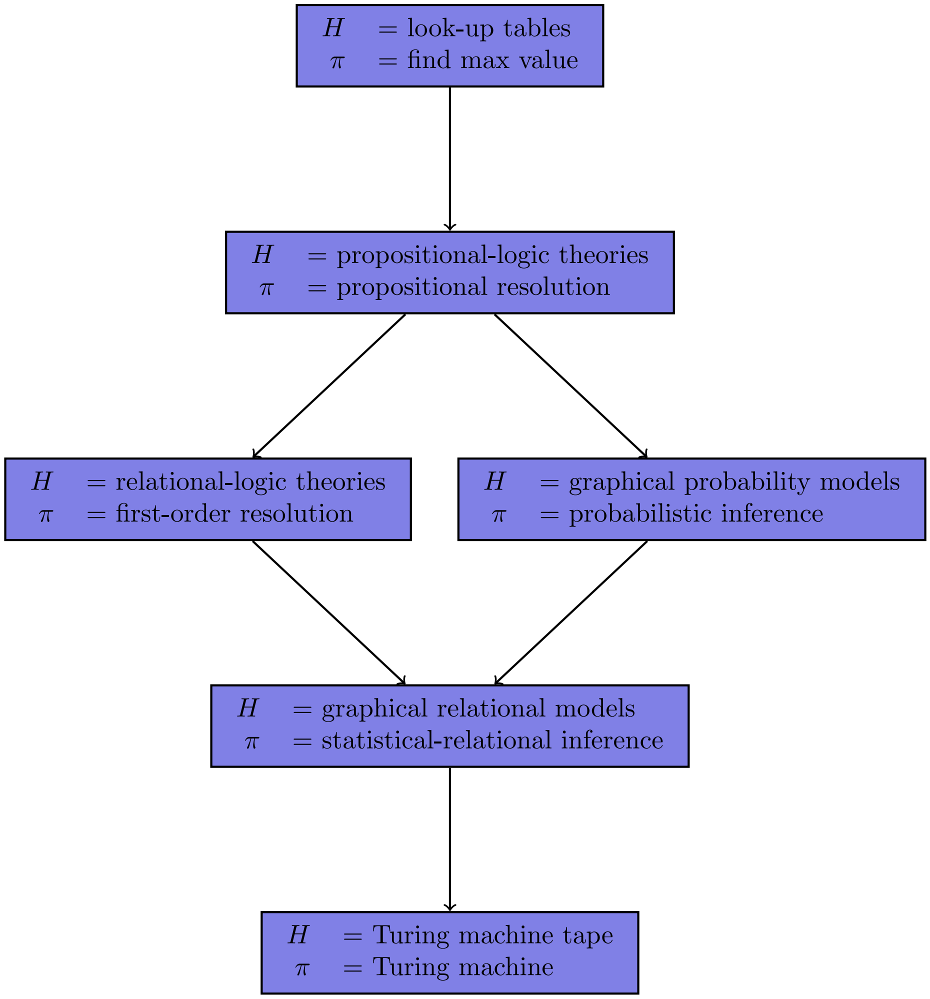



Figure 9: Hypothesis representations and their corresponding policy classes (interpreters) considered in this course. Arrow directions indicate increasing expressiveness. *Note: the bottom box was not covered in SMU 2017 and will not be part of the exam. The box above it was covered in the lectures but is not part of this book or the exam.*

4. reinforcement learning

5. universal learning (not covered here)

# 2 On-line Concept Learning

We first motivate the on-line concept learning scenario with an example, in which the agent is an artificial scientist. The agent conducts repeated experiments with a living cell, which represents the environment. In each experiment, it observes two proteins of interest in the cell. More specifically, the agent detects whether the proteins are present in the cell at all, and it also determines whether they are in an active state (a special spatial conformation of a protein). The agent suspects that these proteins (both or only one of them) initiate apoptosis (cellular suicide). After each observation of the proteins, it tries to predict whether the cell will die or not. If the prediction is incorrect, the agent receives a negative reward. This can be for example a cut-down on the agent's salary by the boss of the lab who is not happy with wrong biological predictions, in which case the boss would be a part of the environment. However, we will simply model such a reward with the number -1 for wrong predictions and with 0 for correct predictions.

| experiment number $k$ | apoptosis initiated $s_k$ | prot. 1 present $o_k^1$ | prot. 1 active $o_k^2$ | prot. 2 present $o_k^3$ | prot. 2 active $o_k^4$ | apoptosis prediction $y_k$ | reward $r_k$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 1 | -1 |
| 3 | 0 | 1 | 0 | 0 | 0 | 1 | -1 |
| 4 | 1 | 1 | 0 | 0 | 0 | 0 | -1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 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.) | | | | | | | |

Table 1: A concept learning experiment.

Table 2 illustrates a history of such agent-environment interaction, in which the agent eventually learns the apoptosis is induced if and only if protein 1 is present and it is in the active form. From time $k = 5$ on, the agent makes correct predictions and is no longer punished with negative reward.

In the sequel, we will see how to model the illustrated scenario in our frameworks and we will see examples of agents able to learn as the agent-scientist has in the story above.

We implement the on-line concept learning scenario as a specific case of the general sequential learning framework. The central assumption of concept learning is that the current observation uniquely determines the current state through function

$$c : O \to S \tag{34}$$

so for (23) it holds

$$\mu_o(o_k|s_k, y_{k-1}) = 0 \text{ if } s_k \neq c(o_k) \tag{35}$$

In other words, the observations are partitioned into *classes* co-inciding with states, and function $c$, which is unknown to the agent, and which *classifies* the observations into these classes.

In the concept learning scenario we will work with two classes only, i.e.

$$S = \{\, 0, 1 \,\} \tag{36}$$

Then function $c$ can be conveniently identified with the subset of observations

$$\underline{c} = \{\, o \in O \;\mid\; c(o) = 1 \,\} \tag{37}$$

and write $o \in \underline{c}$ or $c(o) = 1$ interchangeably. This subset view earns $c$ the name *concept*. In the later text, whenever we speak of a concept $c$, $\underline{c}$ will represent the set given by (37).

In the concept learning scenario, we want the agent to *learn the unknown concept* $c$ by guessing the state $s_k$ at each time $k$ and providing the guess through $y_k = \pi(h_k, o_k)$. The environment will punish the agent by a negative reward for each incorrect guess, and this will make the agent adapt its policy through changing $h_k$. Ideally, these changes should eventually lead to a hypothesis according to which the policy makes only correct guesses. To implement this guessing scenario, we first make sure that the range of actions coincides with the range of states

$$Y = S \tag{38}$$

The rewards should be functionally determined only by the true state and the guess made. So we prescribe it by function $L : S \times Y \to R$ so that (24) takes the specific form (incrementing the time index inconsequentially for shorter notation)

$$\mu_r(r_{k+1}|s_k, y_k) = \begin{cases} 1 \text{ if } r_{k+1} = -L(s_k, y_k) \\ 0 \text{ otherwise} \end{cases} \tag{39}$$

The first reward $r_1$ is immaterial and is still sampled from the marginal $\mu_R(r_1)$.

Function $L$ is called *loss*. The loss should evidently be zero if $s_k = y_k$ and in other cases it quantifies how serious a mistake is made by the wrong guess. Since our goal is just that the agent identifies the concept, we consider all mistakes equally bad and set the loss as[2]

$$L(s_k, y_k) = \begin{cases} 0 \text{ if } s_k = y_k \\ 1 \text{ otherwise} \end{cases} \tag{40}$$

Given (36), and assuming a fixed policy $\pi$, also any hypothesis $h \in H$ can be formally identified with the set

$$\underline{h} = \{\, o \in O \;\mid\; \pi(h, o) = 1 \,\}$$

so that

$$\underline{H} = \{\, \underline{h} \mid h \in H \,\} \tag{41}$$

Again, whenever we speak of a hypothesis $h$ (possibly with the time index, $h_k$), then $\underline{h}$ ($\underline{h}_k$) will automatically mean the set given by (41).

The fact that the agent's hypothesis exactly matches the unknown concept for any observation $o_k$ can now be simply expressed as

$$\underline{h} = \underline{c} \tag{42}$$

Note that it would not be correct to write $h = c$ even if $\underline{h} = \underline{c}$.

Whether or not the agent at some time $k$ learns a hypothesis $\underline{h}_k = \underline{c}$ depends on the agent's update rule (25), and also on whether its hypothesis class[3] $\underline{H}$ contains such a $\underline{h}_k$ at all. To formalize this latter condition, we will assume that the environmental concepts $\underline{c}$ cannot be arbitrary but rather belong to a *concept class* $\underline{C}$. An important property of the particular concept learning scenarios will be whether or not

$$\underline{C} \subseteq \underline{H} \tag{43}$$

Table 2 summarizes the main pieces of notation we use in concept learning.

## 2.1 Generalizing Agent

Here we design an agent that learns an unknown conjunction by starting with the most specific hypothesis (a conjunction of all literals, i.e. all propositional

[2]Defining both a loss function, and a reward as the negative loss seems redundant. Indeed, we could combine (39) and (40) into a single equation without defining the loss at all. We do keep the latter, however, as it is a central established notion of decision theory.

[3]We take the liberty to call *hypothesis class* both $H$, i.e. the set of hypothesis representations, and $\underline{H}$, i.e. the family of sets generated by the representations together with the fixed policy. The word *class* in the terms *hypothesis class* and *concept class* should not be confused with the classes of observations, which are states.

| symbol | meaning |
|---|---|
| $h$ | agent's hypothesis (e.g. a rule) |
| $\pi(h, o)$ | the agent's policy interpreting hypothesis $h$ to produce a binary decision from observation $o$ |
| $H$ | set of possible agent's hypotheses (hypothesis class) |
| $c(o)$ | unknown concept mapping observations to binary states |
| $C$ | set of possible concepts (concept class) |
| $\underline{h}$, $\underline{c}$ | set of all observations $o$ mapped to 1 by $\pi(h, o)$ or $c(o)$ (respectively). Also called a *hypothesis* and a *concept* (respectively), just like the corresponding $h$ and $c$. |
| $\underline{H}$, $\underline{C}$ | set of all $\underline{h}$'s and all $\underline{c}$'s (respectively) following from different choises of $h \in H$ and $c \in C$. |

Table 2: Summary of notation for concept learning

variables as well as their negations) and then deleting all literals inconsistent with the received observations. So the initial hypothesis is gradually *generalized* towards the correct one.

Recall the example in Table 2 and let the propositions “Protein 1 is present” and “Protein 1 is active” be represented by logical symbols $p_1$ and $p_2$, respectively. The analogical assertions for Protein 2 will be represented by symbols $p_3$ and $p_4$. The strategy of the generalizing agent is to start with the initial hypothesis that apoptosis is induced if and only if

$$p_1 \wedge \neg p_1 \wedge p_2 \wedge \neg p_2 \wedge p_3 \wedge \neg p_3 \wedge p_4 \wedge \neg p_4 \tag{44}$$

The is the most specific hypothesis as it conjoins all possible conditions (literals). At the same time, this conjunction can of course never be true as it is self-contradictory. However, the agent's strategy is to successively remove from it all the literals that are inconsistent with the coming observations. Eventually, it should achieve the correct conjunction

$$p_1 \wedge p_2 \tag{45}$$

We will now design such an agent precisely. The main thing we will need to prove is that the successive deletions indeed lead to the correct hypothesis.

Observations are $n$-tuples of binary (truth) values

$$O = \{\, 0, 1 \,\}^n \tag{46}$$

The agent has the hypothesis class

$$H = \Phi \times O \tag{47}$$

where

$$\Phi = \left\{ \bigwedge_{i\in I} p_i \bigwedge_{j\in J} \neg p_j \;\middle|\; I, J \subseteq [1:n] \right\} \tag{48}$$

and $n \in N$. So

$$h_k = (\phi_k, o'_k) \tag{49}$$

consists of a conjunctive formula $\phi_k$ containing at most $2n$ literals, and $o'_k \in O$. The latter has the purpose of memorizing the last observation (example) provided by the environment and will be used only for updating hypotheses.

The formula $\phi_k$ is used to determine decisions through the agent's decision policy (26) $y_k = \pi(h_k, o_k) = \pi((\phi_k, o'_k), o_k)$. Whenever the policy does not depend directly on the memorized example $o'_k$, which will be the typical case, we will afford the shorter notation $\pi(\phi_k, o_k)$. The policy is set to

$$y_k = \pi(\phi_k, o_k) = \begin{cases} 1 \text{ if } o_k \models \phi_k \\ 0 \text{ otherwise} \end{cases} \tag{50}$$

where $o_k \models \phi_k$ means $\phi_k$ is true given the truth-value assignments $o_i$ to variables $p_i$, $1 \le i \le n$. More precisely, we say that positive (negative, respectively) literal $p_i$ ($\neg p_j$) is *consistent* with observation $o_k$ if $o_k^i = 1$ ($o_k^i = 0$). Finally, $o_k \models \phi_k$ holds if and only if all literals of conjunction $\phi_k$ are consistent with $o_k$.

The update rule (25), which we expand by (4) and (49) to

$$(\phi_k, o'_k) = \mathcal{H}\left(\left(\phi_{k-1}, o'_{k-1}\right), (o_k, r_k)\right) \tag{51}$$

is set to

$$o'_k = o_k \tag{52}$$

$$\phi_k = \begin{cases} \phi_{k-1} \text{ if } r_k = 0 \\ \mathsf{delete}(\phi_{k-1}, o'_{k-1}) \text{ otherwise} \end{cases} \tag{53}$$

where

$$\mathsf{delete}\left( \bigwedge_{i\in I} p_i \bigwedge_{j\in J} \neg p_j, (o^1, o^2, \ldots, o^n) \right) = \tag{54}$$

$$\bigwedge_{\substack{i \in I \\ o^i = 1}} p_i \bigwedge_{\substack{i \in I \\ o^j = 0}} \neg p_j \tag{55}$$

So the $\mathsf{delete}$ function keeps exactly those literals from $\phi_{k-1}$ which are consistent with $o'_{k-1}$.

We assume that (43) holds. In particular, there exists a target conjunction $\phi^* \in \Phi$ such that $h^* = (\phi^*, o)$ exactly simulating the unknown concept $c$, i.e.

$$s_k = c(o_k) = \pi(\phi^*_k, o_k) \tag{56}$$

**Lemma 2.1.** *$s_k = 1$ if and only if all literals of $\phi^*$ are consistent with $o_k$.*

The above lemma follows directly from (50) and (56).

**Lemma 2.2.** *Whenever $\mathsf{delete}(\phi_{k-1}, o'_{k-1})$ is called, $s_{k-1} \neq y_{k-1}$, and if $s_{k-1} = 0$, then all literals of $\phi_{k-1}$ are consistent with $o'_{k-1}$.*

*Proof.* To see why Lemma 2.2 is true, note that according to (53), $r_k \neq 0$ when $\mathsf{delete}$ is called. Due to (39) and (40), this means that $s_{k-1} \neq y_{k-1}$. So if $s_{k-1} = 0$ then $y_{k-1} = 1$, but then due to (50), $o'_{k-1} \models \phi_{k-1}$ and so all literals of $\phi_{k-1}$ are indeed consistent with $o'_{k-1}$. ☐

**Lemma 2.3.** *$\mathsf{delete}(\phi_{k-1}, o'_{k-1})$ never removes a literal $l \in \phi_{k-1}$ which is also in $\phi^*$.*

*Proof.* Assume for contradiction that it removes a literal $l \in \phi^*$. First assume $s_{k-1} = 0$. By Lemma 2.2, all literals of $\phi_{k-1}$ are consistent with $o'_{k-1}$. But because $\mathsf{delete}(\phi_{k-1}, o'_{k-1})$ keeps all literals of $\phi_{k-1}$ consistent with $o'_{k-1}$, it does not delete $l$, which is a contradiction. Now consider $s_{k-1} = 1$. Then by Lemma 2.1 all literals of $\phi^*$ including $l$ must be consistent with $o'_{k-1}$. Again, since $\mathsf{delete}$ keeps all consistent literals, it does not delete $l$, which is a contradiction. ☐

The starting hypothesis of the designed agent is set to contain all possible literals

$$\phi_1 = p_1 \wedge \neg p_1 \wedge p_2 \wedge \neg p_2 \wedge \ldots p_n \wedge \neg p_n \tag{57}$$

Thus $\phi_1 \supseteq \phi^*$, where the inclusion is with respect to the sets of literals in $\phi_1$ and $\phi^*$. Furthermore, due to Lemma 2.3, we have

$$\phi_k \supseteq \phi^*, \ \forall k \in N \tag{58}$$

Given the above, the agent makes mistakes only on 'positive examples', and the mistakes are corrected by removing at least one inconsistent literal, as the following lemma formalizes.

**Lemma 2.4.** *Assuming (57), whenever $\mathsf{delete}(\phi_{k-1}, o'_{k-1})$ is called, $s_{k-1} = 1$, and the function deletes at least one literal from $\phi_{k-1}$.*

*Proof.* Due to Lemma 2.2, $s_{k-1} \neq y_{k-1}$. If $s_{k-1} = 0$ and $y_{k-1} = 1$ then by the same lemma, all literals of $\phi_{k-1}$ are consistent with $o'_{k-1}$. According to Lemma 2.1, there would then be a literal in $\phi^*$ inconsistent with $o'_{k-1}$. But due to (58), this inconsistent literal would also be contained in $\phi_{k-1}$, which is a contradiction. So we know that $s_{k-1} = 1$ and $y_{k-1} = 0$. According to (50), this means that $\phi_{k-1}$ contains a literal inconsistent with $o'_{k-1}$. Since delete, by definition, keeps exactly all consistent literals, the inconsistent literal is removed. □

**Theorem 2.5.** *The agent makes at most* $2n$ *mistakes, i.e. the cumulative reward is*

$$\sum_{k=1}^{m} r_k \geq -2n \tag{59}$$

*for an arbitrary horizon* $m \in N$.

*Proof.* Since the first agent's conjunction has $2n$ literals by (57) and upon each mistake, at least one literal is removed from from the conjunction by Lemma 2.4, the maximum number of mistakes is $2n$. □

While the agent's strategy has been designed to learn conjunctions, it can be also made to learn disjunctions due to the equality

$$\neg (p_1 \vee p_2 \vee \ldots \vee p_n) = \neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \tag{60}$$

So the only required change is that the agent replaces observations $o_k$ with $\overline{o_k} = (1 - o_k^1, 1 - o_k^2, \ldots, 1 - o_k^n)$ and its actions $y_k$ with $1 - y_k$.

Other logical classes can also be reduced to conjunction and disjunction learning. Consider e.g. $s$-CNF ($s < \infty$). These are conjunctions of $s$-clauses. An $s$-clause is a disjunction of at most $s$-literals. There is a finite number of $s$-clauses so the agent can simply establish one new propositional variable for each possible $s$-clause a learn a conjunction with these new variables. This reduction would even be efficient if $s$ is a small constant. Indeed, if $n$ is the number of original variables, then the number of possible clauses is $\binom{n}{s}$ which grows exponentially with $s$ and polynomially with $n$. A similar reduction can be used to learn $s$-DNF.

## 2.2 The Subsumption Relation

It is instructive to view the generalization process as a path in the *subsumption lattice* of conjunctions shown for two propositional symbols in Fig. 10. A *lattice* is a partially ordered set where each two elements have their unique least upper bound and the greatest lower bound. The subsumption order is given by the subset relation

$$\phi_1 \subseteq \phi_2 \tag{61}$$

This means that conjunction $\phi_1$ precedes conjunction $\phi_2$ if the latter contains all literals of the former.

Recall from logic that a formula $\phi_1$ *entails* another formula $\phi_2$ if any model of $\phi_1$ is also a model of $\phi_2$. We denote this as

$$\phi_1 \vdash \phi_2 \tag{62}$$

It is obvious that $\phi_1 \subseteq \phi_2$ implies $\phi_2 \vdash \phi_1$ if $\phi_1$ and $\phi_2$ are conjunctions. However, the inverse implication does not hold. For example (observe Fig. 10), we have both $p_1 \wedge \neg p_1 \vdash p_2 \wedge \neg p_2$ and $p_2 \wedge \neg p_2 \vdash p_1 \wedge \neg p_1$ simply because both of the formulas are non-satisfiable and thus neither has a model. However, they do not share any literal so the subset relation does not hold either way. Nevertheless, for satisfiable conjunctions (i.e., conjunctions other than 'contradictions') $\phi_1, \phi_2$, $\phi_1 \subseteq \phi_2$ is equivalent to $\phi_2 \vdash \phi_1$.

While so far, we considered subsumption only conjunctions, the literal subset relation (61) is obviously defined as well for disjunctions, i.e. clauses. However, the relationship to logical entailment becomes inverted. More precisely, for two clauses $\phi_1, \phi_2$, $\phi_1 \subseteq \phi_2$ implies $\phi_1 \vdash \phi_2$. Just like in the case of conjunctions, we cannot claim equivalence between the two latter relations. For example $p_1 \vee \neg p_1 \vdash p_2 \vee \neg p_2$. Again, the problem is with the atoms included both as a positive and a negative literals. While in conjunctions they produced contradictions, their presence in clauses make the latter tautologies, i.e. formulas true in any interpretation. But analogically to conjunctions, $\phi_1 \subseteq \phi_2$ is equivalent to $\phi_2 \vdash \phi_1$ if $\phi_1, \phi_2$ are not tautologies.

Contradictory conjunctions and tautological clauses have one property in common. They contain a positive literal as well as the negation of the same literal. Clauses, which have this property, are called *self-resolving*.[4]

## 2.3 Separating agent

Here we will build an agent with a strategy completely different from the generalization agent. In particular, agent's hypothesis $h$ will define a hyperplane in the $O = \{0, 1\}^n$ space (46) so $\underline{h}$ will be exactly those observations lying above the hyperplane.

[4]As the adjective *self-resolving* originates from the resolution principle, which is applied on clauses and not on conjunctions, it is usually not associated with conjunctions.

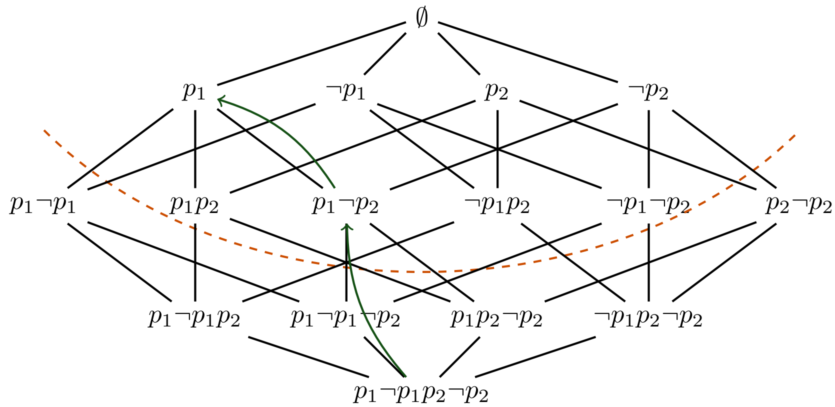



Figure 10: Subsumption lattice for conjunctions. The conjunction symbols $\wedge$ are omitted for brevity. The curved arrows show how the agent generalizes its initial conjunction in two steps following the successive observations $(1,0)$ and $(1,1)$ carrying the respective truth values for $p_1$ and $p_2$. All conjunctions below the dashed line are non-satisfiable.

We will first assume that the concept to be learned corresponds to a disjunction, so

$$C = \{\, c_\phi \mid \phi \in \Phi \,\} \tag{63}$$

where for $s \leq n$

$$\Phi = \{\, p_{i_1} \vee p_{i_2} \vee \ldots \vee p_{i_s} \mid 1 \leq i_1, i_2, \ldots i_s \leq n \,\} \tag{64}$$

and

$$c_\phi(o) = \begin{cases} 1 \text{ if } o \models \phi \\ 0 \text{ otherwise} \end{cases} \tag{65}$$

Although (64) considers only *monotone* disjunctions, i.e. without negated literals, it can be easily generalized to general disjunctions by introducing $2s$ (instead of $s$) propositional variables $p'_i = p_i$, $p'_{2i} = \neg p_i$.

A hypothesis of the separating agent is an $n$-tuple of integer coefficients bounded by some constant $q \in N$ and a memory for the last observation, i.e.

$$h_k = (w_k, o'_k) \tag{66}$$

where $w_k = (w_k^1, w_k^2, \ldots w_k^n)$. So the hypothesis space is

$$H = [0, 1, 2, \ldots, q]^n \times O \tag{67}$$

The agent's decision policy is given by a threshold function applied on the dot product of the agents coefficent tuple with the observation tuple

$$y_k = \pi(w_k, o_k) = \begin{cases} 1 \text{ if } w_k \cdot o_k > n/2 \\ 0 \text{ otherwise} \end{cases} \tag{68}$$

Assume again that $\underline{C} \subseteq \underline{H}$. This can be achieved with a sufficiently large $q$ as disjunctions are linearly separable.

The initial $n$-tuple of coefficients is

$$w_1 = (1, 1, \dots 1) \tag{69}$$

And the hypotheses are updated by the following rule

$$(w_k, o'_k) = \mathcal{H}\left(\left(w_{k-1}, o'_{k-1}\right), (o_k, r_k)\right) \tag{70}$$

where

$$o'_k = o_k \tag{71}$$

$$w_k = \begin{cases} w_{k-1} \text{ if } r_k = 0 \\ \mathsf{update}(2, w_{k-1}, o'_{k-1}) \text{ if } w_{k-1} \cdot o'_{k-1} \leq n/2 \\ \mathsf{update}(0, w_{k-1}, o'_{k-1}) \text{ if } w_{k-1} \cdot o'_{k-1} > n/2 \end{cases} \tag{72}$$

and where the function update is defined such that for $w_k = \mathsf{update}(\theta, w, o)$

$$w^i_k = \begin{cases} \theta \cdot w^i \text{ if } o^i = 1 \\ w^i \text{ otherwise} \end{cases} \tag{73}$$

**Theorem 2.6.** *The agent makes at most $2 + 2s \lg n$ mistakes, , i.e. the cumulative reward is*

$$\sum_{k=1}^{m} r_k \geq -2 - 2s \lg n \tag{74}$$

*for any horizon $m \in N$.*

(proof omitted)

Just like the generalizing agent designed to learn conjunctions could easily be modified to learn disjunctions, $s$-CNF, and $s$-DNF, also the separating agent can be altered to learn conjunctions as well as the latter two classes by means of the same reduction principles.

So the two agents can in principle learn the same concept classes. The difference is in the mistake bound. The latter agent performs better when the number of variables $n$ is larger than the number of relevant variables $s$.

## 2.4 Version Space Agent

How well can we do with *arbitrary* concept classes? Immediate mistake bound for any concept class $C$

$$|C| - 1 \tag{75}$$

Can be improved to $\lg |C|$ using the *version space* strategy.

Assume a set $\Phi$ of *versions*. These may be conjunctions, disjuctions, or other representations. The only assumption is that each version $\phi \in \Phi$ provides a decision $\phi(o)$ for any observation $o \in O$. So this function works similarly to a decision policy $\pi$, however, the version-space agent uses a policy $\pi$ that aggregates multiple versions for a single decision.

The hypothesis class is

$$H = 2^{\Phi} \times O \tag{76}$$

so

$$h_k = (V, o) \tag{77}$$

where $V$ is a set ('space') of versions, and $o$ again stores the last observation. The plan is that $V$ maintains all versions from $\Phi$ consistent with the observations and rewards received so far.

Decisions are determined by voting of all versions in the current version space

$$y_k = \pi(V_k, o_k) = \begin{cases} 1 \text{ if } |\{\, \phi \in V_k \mid \phi(o_k) = 1 \,\}| > |V_k|/2 \\ 0 \text{ otherwise} \end{cases} \tag{78}$$

The initial version space contains all versions from $\Phi$

$$V_1 = \Phi \tag{79}$$

and in the hypothesis update step, the agent deletes from its version set all versions inconsistent with the last observation, i.e.

$$o'_k = o_k \tag{80}$$

$$V_k = \{\, \phi \in V_{k-1} \mid \phi(o_{k-1}) = s_{k-1} \,\} \tag{81}$$

where $s_{k-1}$ is determined as $s_{k-1} = |y_{k-1} - r_{k-1}|$ (check that this is true) and $y_{k-1} = \pi(V_{k-1}, o'_{k-1})$.

Assume that $\Phi$ is rich enough so that it contains $\phi \in \Phi$ so that $\phi(o) = c(o)$ for all $o \in O$ (check that this implies 43). Then the following holds.

**Theorem 2.7.** *The agent makes at most* $\lg|\Phi|$ *mistakes, i.e. the cumulative reward is*

$$\sum_{k=1}^{m} r_k \geq -\lg|\Phi| \tag{82}$$

*for any horizon* $m \in N$.

*Proof.* To see why the theorem holds note that the agent decides by the majority of current versions. So if a mistake is made, at least half of the versions are deleted. In the worst case, the last remaining version is correct. □

The logaritmic bound is good but the computational demands for storing the version space can be prohibitive.

## 2.5 The Mistake Bound Learning Model

The linear mistake bounds we obtained for the generalizing and separating agents indicate that these agents are indeed able to learn well the conjunctive and disjunctive concepts but also other kinds of concepts (namely, $s$-DNF and $s$-CNF) that can be reduced to the latter. We will now generalize the notion of 'good on-line learning.' We say that an agent *learns concept class $C$ on-line* if it makes at most $p(n)$ of mistakes in the on-line scenario with any concept from $C$, where $p$ is a polynomial and $n$ is the size of observations. With our setting (46), the size of observations is the number $n$ of binary values making up the observations.

By Theorem 2.7, the version space algorithm has a mistake bound $\lg|\Phi|$ as long as $\Phi$ contains a version coinciding with the concept. So if $\Phi$ contains a version for any concept from $C$ and $|\Phi|$ is at most exponential in $n$ it necessarily learns $C$ on-line, because the mistake bound $\lg|\Phi|$ would then be polynomial. But note that $|\Phi|$ may be super-exponential. The extreme example of the latter is the space $\Phi$ so rich that it has a $\phi \in \Phi$ for any possible mapping $\phi : O \to S$. There are $2^n$ different possible observations $o \in O = \{0,1\}^n$, each of which is classified in one of the two states $S = \{0,1\}$. Then $|\Phi| = 2^{2^n}$ is super-exponential.

Furthermore, we refine the definition into a stricter form. An agent that learns concept class $C$ on-line is said to learn it *efficiently* if it spends at most polynomial time (in observation size) between the receipt of a percept and the generation of the next action.

What about a *lower bound* on mistakes? We say that a set of observations $O' \subseteq O$ is *shattered* by hypothesis class $\underline{H}$ if

$$\{ O' \cap \underline{h} \mid \underline{h} \in \underline{H} \} = 2^{O'} \tag{83}$$

which means that the set of observations can be partitioned in all possible ways into two classes by the hypotheses from $\underline{H}$.

The *Vapnik-Chervonenkis Dimension* (or VC-dimension) of $\underline{H}$, written $\mathsf{VC}(\underline{H})$, is the cardinality of the largest set $O' \subseteq O$ that can be shattered by $\underline{H}$. The definition extends formally also to $H$ corresponding to $\underline{H}$ by (41), so we will also write $\mathsf{VC}(H)$.

**Theorem 2.8.** *No upper bound on the number of mistakes made by an agent in the concept-learning scenario using hypothesis space $H$ is smaller than $\mathsf{VC}(H)$.*

*Proof.* This is because for any sequence of agent's decisions $y_1, y_2, \ldots, y_{\mathsf{VC}(H)}$ there exists a $h \in H$ according to which all these decisions are wrong. □

# 3 Batch Concept Learning

The batch concept learning situation is defined by the assumptions of batch learning (Section 1.7) combined with the concept-learning requisities which are the same as in on-line concept learning (Section 2). In particular, the latter include the assumption of a target concept determining states from observations (34), the binary range of observations (46) and states (36), and the unit loss function (40) determining rewards (39).

Since rewards are negative losses by (39), the expected reward (33) to be maximized is in $[-1; 0]$. Its negative value, for a given hypothesis and $k > K$, is called the *error* of the hypothesis

$$\mathrm{err}(h_K) = -\sum_{r_k \in R} \mu_R(r_k|h_K) r_k \tag{84}$$

and corresponds to the proportion of misclassified observations in the testing phase, i.e. those observations $o_k$ $(k \geq K)$[5] for which $y_k \neq s_k$. Given (40) and (84), the error can be expressed as the probability of making a mistake, i.e. receiving a -1 reward at an arbitrary time $k > K$

$$\mathrm{err}(h_K) = \mu_R(-1|h_K) \tag{85}$$

A natural question of interest is how the algorithms we designed for on-line concept learning in the sequential scenario would perform in terms of the error (85). Evidently, the bounds on the number of mistakes we established in Theorems 2.5, 2.6, and 2.7 do not translate to any bound on $\mathrm{err}(h_K)$ as there is no guarantee that the mistakes will happen in the learning phase $(k \leq K)$ where the agent still can fix its hypothesis.

[5]Make sure to understand why the inquality is non-strict here.

But unlike in the on-line learning case, the batch case inherits the non-sequential assumptions (28) and (29), meaning that states and observations are sampled i.i.d. according to distributions that do not change with $k$. They prevent the environment from 'adversarial' behavior, for example, one where the training phase would only contain 'easy' examples and the 'hard' ones would be kept for the testing phase. As we will see, in this scenario we can indeed bound $\mathrm{err}(h_K)$ for particular learning agents, although we will be able to do it only with certain probability smaller than 1.

## 3.1 Batch Learning with the Generalizing Agent

Assume the generalizing agent as described in Section 2.1 working in the learning phase ($k \leq K$) of the batch scenario just as it worked in the on-line scenario.

Denote by $\Pr(l)$ the probability that a literal $l$ (i.e., $p_i$ or $\neg p_i$ for $1 \leq i \leq n$) is inconsistent with observation $o_k$. Since observations are now i.i.d., this probability does not depend on $k$. We already know that a hypothesis $h_k$ using conjunction $\phi_k$ with only consistent literals has zero error. So the probability of guessing the wrong class is the probability that some of the literals in $\phi_k$ are inconsistent. Thus we have the bound

$$\mathrm{err}(h_k) \leq \sum_{l \in \phi_k} \Pr(l) \tag{86}$$

We have no more than $2n$ literals in $\phi_k$ so if $\Pr(l) \leq \epsilon/2n$ for each of them then $\mathrm{err}(h_k) \leq \epsilon$. Call a literal *bad* if $\Pr(l) > \epsilon/2n$. The probability that a bad literal $l$ survives $k$ observations is

$$(1 - \Pr(l))^k < \left(1 - \frac{\epsilon}{2n}\right)^k \tag{87}$$

It is important to realize that (87) would not be correct if the observations $o_1, o_2, \ldots o_k$ were not i.i.d. The above equation thus rests fully on the extra assumptions of the nonsequential scenario (of which batch learning is a special case), which we did not adopt for on-line learning.

There are at most $2n$ bad literals so the probability that some of them has survived $k$ steps is at most

$$2n\left(1 - \frac{\epsilon}{2n}\right)^k \tag{88}$$

To work with this upper bound easily, we make use of the inequality $1 - x \leq e^{-x}$ which holds for $x \in [0; 1]$, to obtain

$$2n\left(1 - \frac{\epsilon}{2n}\right)^k \leq 2n e^{-k\frac{\epsilon}{2n}} \tag{89}$$

We now summarize the above inferences into a theorem.

**Theorem 3.1.** *Hypothesis $h_{k+1}$ of the generalizing agent in the learning phase ($k < K$) has* $\text{err}(h_{k+1}) \leq \epsilon$ *with probability at least* $1 - 2ne^{-k\frac{\epsilon}{2n}}$

Note that the $k + 1$ index is due to the fact that $k$ observations are used to learn $h_{k+1}$ ($o_k$ and $r_{k+1}$ are used to create $h_{k+1}$). So at the end of learning, $\text{err}(h_K) < \epsilon$ with probability at least $1 - 2ne^{-(K-1)\frac{\epsilon}{2n}}$.

## 3.2 Batch Learning with General On-line Agents

We define a *standard* on-line agent as one that changes its hypothesis if and only if a mistake has been made by the previous hypothesis. This includes the generalizing and separating agents as follows from the update rules (52) and (72). On the other hand, the version-space agent is not standard as by (81) it updates its hypothesis whenever its current version space contains an inconsistent version, even if the most recent decision determined by the majority vote of versions was correct. For all of the agents designed, we will also assume that their hypothesis spaces include a hypothesis perfectly matching the unknown concept, i.e. (43) holds.

The next lemma will enable us to accommodate any on-line learning agent for the batch learning scenario with a probabilistic bound on the error of the learned hypothesis.

**Lemma 3.2.** *If a standard on-line agent retains a hypothesis $h_k$ for $q$ steps ($h_k = h_{k+1} = \ldots h_{k+q}$), then* $\text{err}(h_k) \leq \epsilon$ *with probability at least* $1 - e^{-q\epsilon}$.

*Proof.* To see why the lemma is correct, we again realize that the probability that the standard agent keeps a bad hypothesis ($\text{err}(h_k) > \epsilon$) on receiving an observation is exactly the probability $1 - \text{err}(h_k)$ that the bad hypothesis produces a correct decision for that observation. Since $\text{err}(h_k) > \epsilon$, the probability is at most $1 - \epsilon$. The probability of keeping the hypothesis over $q$ i.i.d. observations is thus at most $(1 - \epsilon)^q$, and we already know that $(1 - \epsilon)^q \leq e^{-q\epsilon}$. Otherwise, i.e. with probability at least $1 - e^{-q\epsilon}$, the hypothesis was not bad, i.e $\text{err}(h_k) \leq \epsilon$. □

So the rule is: wait until $h_k = h_{k+1} = \ldots h_{k+q}$ happens and then keep $h_k$ with the probabilistic error bound. The question is how to guarantee that the event indeed happens within the learning phase, i.e. $k + q \leq K$. If we have a mistake bound $M$ for the agent, we know that the standard agent makes at most $M$ hypothesis changes. In this case we set the learning phase long enough, in particular $K = Mq$, to guarantee that one of the hypothesis in the learning phase survives at least $q$ observations.

## 3.3 Consistent Agent

Here we design a general agent working with an arbitrary hypothesis space. This is analogical to the version space agent we studied in the on-line setting.

We first adapt the version space agent from on-line learning to batch learning. In the learning phase, the agent works just as in the on-line setting. When the phase ends, i.e. $k = K$, the agent updates the version space for the last time according to (80) and then selects an arbitrary version $\phi_K$ from the version space $V_K$. All other versions are deleted from $V_K$, so $V_K = \{ \phi_K \}$, and $\phi_K$ thus dictates the decision policy for $k \geq K$

$$\pi(V_K, o_k) = \phi_K(o_k) \tag{90}$$

which is because of the majority vote given by (78).

For short notation, we formally extend the error function to versions, so $\mathrm{err}(\phi_K)$ is the error achieved by the above policy. We call a version $\phi$ bad if $\mathrm{err}(\phi) > \epsilon$. The probability that a bad version $\phi$ survives $k$ observations is at most $(1-\epsilon)^k \leq e^{-\epsilon k}$. The probability that *some* bad version from the initial version space (79) survives is at most $|\Phi| e^{-\epsilon k}$. So that probability that no bad version survives and thus $\mathrm{err}(\phi_K) > \epsilon$ whichever $\phi_K$ the agent has picked from the last version space, is at least $1 - |\Phi| e^{-\epsilon k}$.

Maintaining the version set is difficult but an equivalent behavior without the version spaces is achieved as follows. All observationss $o_k$ seen up to $k = K - 1$ are stored in memory along with the true classes $s_k$. The latter are obtained by always making the decision $y_k = 0$ in the training phase so that $s_k = -r_{k+1}$. Then the agent finds any hypothesis $h_K \in H$ consistent with the collected set, i.e. $\pi(h_K, o_k) = s_k$ for all $k < K$. Analogically, to the reasoning above, we have that

**Lemma 3.3.** *The probability that the consistent agent's hypothesis $h_K$ has error* $\mathrm{err}(h_K) \leq \epsilon$ *is at least* $1 - |H| e^{-\epsilon(K-1)}$.

This defines the *consistent agent*. Of course, finding such a hypothesis may be computationally hard.

## 3.4 The PAC Learning Model

Agent *probably approximately learns concept class $C$ (in the batch setting)* if at the end of the training phase it produces $h_K$ such that $\mathrm{err}(h_K) \leq \epsilon$ with probability at least $1 - \delta$, and $K \leq p(n, 1/\delta, 1/\epsilon)$, where $p$ is a polynomial.

"probably approximately learns" = "PAC-learns" (C for correctly)

It PAC-learns the class *efficiently* if it spends at most polynomial (in the same variables) time between the receipt of a percept and the generation of the next action in the training phase.

**Theorem 3.4.** *The generalizing agent efficiently PAC-learns conjunctions.*

*Proof.* Efficiency is obvious: at most $2n$ unit steps (going over literals) for each of $n$ observations. From Theorem 3.1, the probability that $\mathrm{err}(h_{k+1}) > \epsilon$ is at most $2ne^{-k\frac{\epsilon}{2n}}$. It remains to determine how many observations $k$ are needed to make the probability smaller than a given $\delta$ and see if the result is polynomial.

$$\begin{aligned} \delta &> 2ne^{-k\frac{\epsilon}{2n}} \\ \frac{\delta}{2n} &> e^{-k\frac{\epsilon}{2n}} \\ \ln\frac{\delta}{2n} &> -k\frac{\epsilon}{2n} \\ \frac{2n}{\epsilon}\ln\frac{2n}{\delta} &\leq k \end{aligned}$$

So the required $k$ is indeed polynomial in $n$, $1/\epsilon$ and $1/\delta$. □

**Theorem 3.5.** *Any standard agent learning (efficiently) a concept class $C$ on-line, has a counterpart which (efficiently) PAC-learns $C$.*

*Proof.* The agent makes at most $u < p(n)$ updates, i.e. max number of mistakes.

Its batch counterpart works as follows.

Set $q = \frac{1}{\epsilon}\ln(\frac{1}{1-\delta})$

If before $u$ updates have been made, each hypothesis survived for less than $q$ steps, then the last one (which makes no mistakes) is found in at most $uq$ steps, and is kept as $h_K$. Both $u$ and $q$ are polynomial.

If some of them survived for at least $q$ steps, than according to lemma (3.2), its error is less than $\epsilon$ with probability at least $1 - e^{-q\epsilon} = \delta$. This hypothesis found with less than $uq$ (poly) steps, will be kept as $h_K$. □

So a negative batch (PAC) result also means a negative on-line result.

**Theorem 3.6.** *If $\underline{C} \subseteq \underline{H}$ and $|\underline{H}|$ is at most exponential in $n$ then the consistent agent using $H$ PAC-learns $C$.*

*Proof.* By Lemma (3.3), probability $\delta$ that $\mathrm{err}(h) > \epsilon$ is at most $|H|e^{-\epsilon k}$.

$\delta \leq |H|e^{-\epsilon k}$

$\frac{\delta}{|H|} \leq e^{-\epsilon k}$

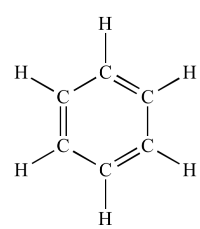
$\frac{1}{\epsilon} \ln \frac{|H|}{\delta} > k$

Since $|\underline{H}|$ is at most exponential in $n$, $\ln |\underline{H}|$ is at most polynomial in it, so $k < p(1/\epsilon, 1/\delta, n)$. $\square$

Also, $s$-CNF and $s$-DNF learnable by poly reduction to conjunctions.

# 4 Learning First-Order Logic Concepts

We now revisit the agent-scientist from Section 2, although the agent will now be in a slightly different situation. In particular, it will investigate chemical compounds, that is, structures such as



and learn to predict for each compound whether it is toxic or safe. The important distinction from the story captured in Table 2 is that there is now no obvious way to encode structure such as the above through tuples $o = (o^1, o^2, \ldots)$ of truth values. Here, observations are graphs and we need a language more expressive than propositional logic to describe such graphs, and also to form hypotheses about them. To this end, we will use the language of first-order predicate logic.

First, we will simplify the situation by abstracting from the parts not important for studying the learning principles. In particular, we will ignore the types of chemical elements in the vertices and also the bond types (single, double). We will simply assume that observations are oriented[6] graphs, that is, directed

[6] Orientation of edges may e.g. correspond to charge distribution along the bonds; we assume oriented edges as their representation is simpler in logic than that of oriented edges.

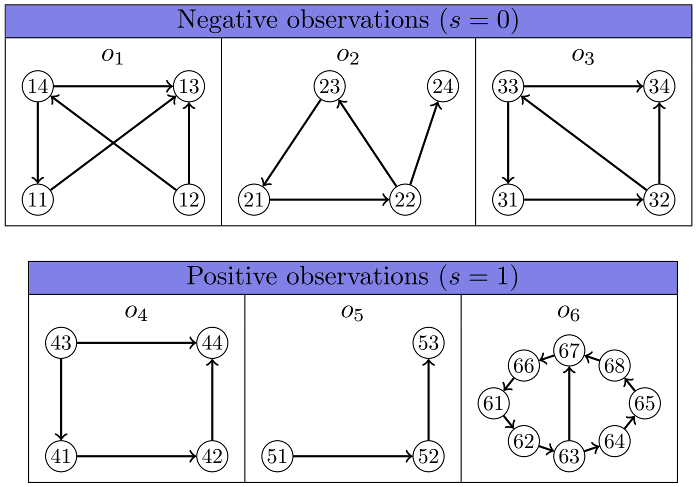



Table 3: Graphical observations from which the agent should learn to classify new observations as negative or positive.

graphs in which no two vertices are connected in both directions. We will assign unique numbers to vertices so that the latter can be addressed later. This is exemplified in Table 3.

Encoding the graphs shown in Table 3 through the language of predicate logic is straightforward. For each graph, we will simply list all of its edges as ground facts of the binary predicate edge. So, for example, the second negative observation will be represented as

$$o_2 = \{\, \mathrm{edge}(21, 22), \mathrm{edge}(22, 23), \mathrm{edge}(23, 21), \mathrm{edge}(23, 24) \,\} \tag{91}$$

and the second positive observed compound will be encoded as

$$o_5 = \{\, \mathrm{edge}(51, 52), \mathrm{edge}(52, 53) \,\} \tag{92}$$

Note that this representation is perfectly analogical to the one used in Section 2.1. In the latter, the observations (46) were truth values assigned to propositional symbols. In (91) and (92) we implicitly assign truth values to all possible ground facts of edge/2 by including exactly those ground facts which hold true, i.e. listing all edges actually in the graph. In both cases, the truth value assignment is called an *interpretation*. In the present first-order context, interpretations such as those shown above, which assign truth values directly to ground atoms of the logical language, are called *Herbrand interpretations*.

Naturally, for more complex problems, the vocabulary of predicates would include more predicates than just edge/2, and an interpretation would define the truth values of all ground facts of all the predicates.

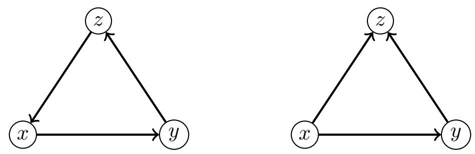



Figure 11: Each negative and no positive observation from Table 3 contains one of these two kinds of triangles.

Returning to the example in Table 3, we would like to design an agent able to learn what patterns are common for only the positive observations (safe compounds) so that such patterns can be used for classifying compounds observed in the future. As we are also intelligent agents, we observe that none of the safe compounds (and each of the toxic ones) contains a triangle. As edges are directed, a triangle may take one of the two forms shown in Fig. 11.

Any other oriented triangle is isomorphic to one of the two shown in the figure. The following formula $\gamma_1$ in predicate logic expresses that a graph does not contain the first (left) kind of triangle

$$\gamma_1 = \forall x, y, z : \neg\text{edge}(x, y) \vee \neg\text{edge}(y, z) \vee \neg\text{edge}(z, x) \tag{93}$$

In the logical representation of graphs, this means that e.g. the negative interpretation $o_2$ above will not be a *model* of this formula. Indeed $o_2 \not\models \gamma_1$ since there exists a substitution $\theta$, namely

$$\theta = \{\, x \mapsto 21, y \mapsto 22, z \mapsto 23 \,\} \tag{94}$$

making all the literals in $\gamma\theta$ false with respect to $o_2$, since all of edge$(21, 22)$, edge$(22, 23)$, and edge$(23, 21)$ are in $o_2$. Analogically, $o_3 \not\models \gamma_1$, so the third negative example is also 'eliminated' by $\gamma_1$. However, the formula does not eliminate $o_1$ – indeed $o_1 \models \gamma_1$. This observation does not contain the first kind of triangle shown in Fig. 11. However, it contains the second one, which is in turn eliminated by the formula

$$\gamma_2 = \forall x, y, z : \neg\text{edge}(x, y) \vee \neg\text{edge}(z, x) \vee \neg\text{edge}(z, x) \tag{95}$$

As before, we can check easily that $o_1 \not\models \gamma_2$ using either the substitution $\theta = \{\, x \mapsto 14, y \mapsto 11, z \mapsto 13 \,\}$ or the substitution $\theta = \{\, x \mapsto 12, y \mapsto 14, z \mapsto 13 \,\}$.

In summary, the conjunction

$$\gamma_1 \wedge \gamma_2 \tag{96}$$

eliminates all negative observations. On the other hand, both $\gamma_1$ and $\gamma_2$ are true in all the positive observations (there is no substitution to $x, y, z$ making all literals of $\gamma_1$ or $\gamma_2$ false in them), so indeed the above conjunction perfectly discriminates between the positive and negative observations.

Note that in a substitution, different variables can map to the same term. So e.g. $\gamma_1$ would also be false in interpretations which we did not intend such as

$$o' = \{ \text{edge}(1,1), \text{edge}(1,2), \text{edge}(2,1) \} \tag{97}$$

as with the substitution

$$\theta = \{ x \mapsto 1, y \mapsto 1, z \mapsto 2 \} \tag{98}$$

all literals $\gamma_1\theta = \neg\text{edge}(1,1) \vee \neg\text{edge}(1,2) \vee \neg\text{edge}(2,1)$ are false with respect to $o'$. Similarly, one can check easily that $\gamma_2$ is false e.g. in interpretation $o'' = \{ \text{edge}(1,2), \text{edge}(2,1) \}$. This would be a problem if $o'$ or $o''$ were positive observations, since they would be eliminated incorrectly by $\gamma_1 \wedge \gamma_2$. There is an apparent reason why $o'$ or $o''$ should be classified as positive (safe): these observations do not contain a proper triangle. However, note that neither $o'$ or $o''$ represent an *oriented* graph as they contain mutually inverse edges and thus they cannot come as observations at all.

## 4.1 Generalizing Agent

In the illustrative example above, the formulas $\gamma_1$ and $\gamma_2$ were first-order logic clauses, that is, universally quantified disjunctions of first-order logic literals. Thus (96) is a first-order logic CNF. We will design an agent able to learn such CNF's in the on-line scenario.

We will keep the basic concept-learning assumptions, i.e. observations uniquely map (34) to binary states (36). Note that these assumptions fit well the illustrative example above. Also, a reward will punish an incorrect prediction with a unit loss as dictated by (39) and (40).

Our new agent will be similar to the generalizing agent from Section 2.1. It will also learn conjunctions following the generalization strategy, except that first-order clauses will be conjoined rather than propositional literals.

Observations will no longer be tuples of truth values as in (46). They will take the form of (Herbrand) interpretations of maximum cardinality $o_{\max}$. In the illustrative graph example above, $o_{\max}$ would be the number of edges in the largest observed graph.

As will become clear later, to guarantee on-line learnability of first-order CNF's, we need to restrict the expressiveness of our language to so called *range-restricted st-clauses*, rather than general first-order clauses. An *st*-clause contains at most $s$ literals and each of them contains at most $t$ occurrences of predicate, variable and function symbols. A clause is range-restricted if any variable occurring in a positive literal of it, also occurs in a negative literal of it. So clauses $\gamma_1$

and $\gamma_2$ from the previous section are examples of range-restricted $3,3$-clauses. The range restriction here follows simply from the fact that the clauses have no positive literals at all.

In Sec. 2.1 we considered the vocabulary of $n$ propositional symbols $p_1, p_2, \ldots, p_n$ out of which the agent constructed conjunctions. With the present first-order logic language, the language vocabulary will be more complex. In particular, $P$ and $F$ are finite sets of predicate and function symbols (respectively) and we denote by

$$\Gamma = \{\ \gamma_1, \gamma_2, \ldots, \gamma_{|\Gamma|}\ \} \tag{99}$$

the set of all range-restricted $st$-clauses made out using only predicate symbols in $P$ and function symbols in $F$. Note that constants are a special case of functions, namely functions of arity 0.

The hypothesis class of the agent will have the same structure as in (47), consisting of the current conjunction and memory for the last observation seen. However, unlike (48), we have

$$\Phi = \left\{ \bigwedge_{i \subseteq I} \gamma_i \ \middle|\ I \subseteq [0 : |\Gamma|] \right\} \tag{100}$$

So $\Phi$ collects all possible conjunctions of clauses from $\Gamma$.

In the propositional-logic setting, the *size of the learning task* was simply $n$, the number of propositional symbols (see Sec. 2.5). Analogically to $n$, we now consider $o_{\max}$. But besides $o_{\max}$, the size of the task is also given by the complexity of the representation vocabulary, that is, the values $|P|$ and $|F|$. So when we speak of a quantity polynomial in the size of the problem in the context of the present agent, it has to be polynomial in all of the three variables.

The agent's decision policy is as defined in (50).

Similarly to the propositional generalizing agent, the agent's first hypothesized CNF is the most specific one. Now this means it is a conjunction of all clauses from $\Gamma$.

$$\phi_1 = \bigwedge_{\gamma \in \Gamma} \gamma \tag{101}$$

The hypothesis update function as in (51-53), except the delete function is different:

$$\mathsf{delete}\left( \bigwedge_{i \in I} \gamma_i, o \right) = \bigwedge_{\substack{i \in I \\ o \models \gamma_i}} \gamma_i \tag{102}$$

so it keeps exactly all clauses consistent with $o$.

Does this agent learn conjunctions of range restricted $st$-clauses on-line according to Section 2.5? Since the agent follows the same strategy as the generalizing agent for propositional conjunctions (Section 2.1), i.e., deleting inconsistent clauses from the curent conjunction, starting with the maximal conjunction, Lemma 2.4 applies. That is to say, the agent makes errors only on positive observations. Also, using Theorem 2.5 (in which we replace the number $2n$ of literals to the number $|\Gamma|$ of clauses in the conjunction), we get that the agent makes at most $|\Gamma|$ mistakes. Now the question is whether $|\Gamma|$ is polynomial in the size of the problem, i.e, in $|P|, |F|, s$, and $t$.

To determine $|\Gamma|$, we first examine the number of different atoms and literals, and then the number $|\Gamma|$ of different $st$-clauses.

Each atom has exactly 1 predicate symbol chosen out of $|P|$ symbols, at at most $t-1$ other symbols in arguments. Each can be chosen out of $|F|$ function symbols or it can be a variable. An $st$-clause can have at most $st$ variables. So there are at most $|P|(|F|+st)^{t-1}$ different atoms, and $2|P|(|F|+st)^{t-1}$ literals. A clause combines at most $s$ literals, so the number of all $st$-clauses can be estimated as

$$\sum_{i=1}^{s} \binom{2|P|(|F|+st)^{t-1}}{i} \leq \sum_{i=1}^{s} \left[2|P|(|F|+st)^{t-1}\right]^{i} = p(|P|,|F|) \qquad (103)$$

and so is polynomial in the size of the learning task. The number $|\Gamma|$ of all range-restricted $st$-clauses is at most the above number and so is also polynomial. So indeed, the agent learns conjunctions of range-restricted $st$-clauses on-line.

Does it also learn efficiently, i.e. does it spend at most polynomial time at each hypothesis update step? That depends on whether the relation $o \models \gamma$ tested for each clause $\gamma = \gamma_i$ in the current conjunction in (102), can be determined efficiently.

We recall from elementary first-order logic that $o \models \gamma$ does *not* hold if and only if there is a ground instance[7] $\gamma\theta$ of $\gamma$ such that

1. atoms of all negative literals of $\gamma\theta$ are in $o$, and
2. no positive literal of $\gamma\theta$ is in $o$

Consider for example a clause $\gamma = \neg\text{edge}(x, y) \vee \neg\text{edge}(y, z) \vee \text{path}(x, z)$, which

[7]that is, a clause obtained from $\gamma$ by substituting each of its variables to some ground term. A ground term does not contain variables. The substitution $\theta$ for which $\gamma\theta$ is ground, is called a *grounding substitution*.

can be rewritten as

$$\text{edge}(x, y) \wedge \text{edge}(y, z) \rightarrow \text{path}(x, z) \tag{104}$$

Note that we did not write the quantification $\forall x, y, z$ in the prefix of $\gamma$. In first-order clauses, *all* variables are universally quantified, and therefore the quantifier part need not be marked explicitly. We will not indicate quantifiers in clauses further in the text, assuming their implicit universal quantification.

Interpretation $o = \{ \text{edge}(a, b), \text{edge}(b, c) \}$ is not a model of $\gamma$ ($o \not\models \gamma$). Indeed the following ground instance $\gamma\theta$ of $\gamma$

$$\text{edge}(a, b) \wedge \text{edge}(b, c) \rightarrow \text{path}(a, c) \tag{105}$$

where $\theta = \{ x \mapsto \mathsf{a}, y \mapsto \mathsf{b}, z \mapsto \mathsf{c} \}$ has both the atoms corresponding to its negative literals, i.e. $\text{edge}(a, b)$ and $\text{edge}(b, c)$, in $o$ but the positive literal $\text{path}(a, c)$ is not in $o$. Another interpretation, $o = \{ \text{edge}(a, b), \text{edge}(a, c) \}$ is neither a model of $\gamma$, this time because there is no substitution $\theta$ that would satisfy condition 1 above. However, the interpretation

$$o = \{ \text{edge}(a, b), \text{edge}(b, c), \text{path}(a, c) \} \tag{106}$$

is a model of $\gamma$.

So to determine $o \models \gamma$, the agent first finds all substitutions $\theta$ satisfying condition 1, and then checks if each of them satisfies condition 2.

The first stage can be arranged as a tree search. The agent starts with the set $A = \{ a_1, a_2, \ldots \}$ of atoms corresponding to $\gamma$'s negative literals. At level $i$ of the tree, atom $a_i$ is unified with some element of $o$ in multiple possible ways corresponding to branches leading to level $i + 1$. The unification grounds a subset of variables present in $A$. When all variables in $A$ have been grounded, the corresponding search node is a successful leaf representing a grounding substitution $\theta$.

We illustrate this with the clause (104) and interpretation (106). The $\gamma$'s negative literals $A = \{ \text{edge}(x, y), \text{edge}(y, z) \}$ are unified with $o$ as shown in Fig. 4.1. In this example, only one grounding substitution exists and is found. We observe that the search tree has, in general, at most $s$ levels and branching factor at most $o_{\max}$, so it has at most $o_{\max}^s$ vertices. The atom in each vertex has at most $t$ arguments so the tree can be searched in at most $to_{\max}^s$ time units, which is polynomial in $o_{\max}$.

The second stage is easy due to range-restriction. In particular, any substitution $\theta$ resulting from the tree search, which makes all negative literals ground, also makes all positive literals ground. Thus checking for each ground positive literal whether or not it is in $o$ (i.e. verifying condition 2) can be done in in at most $so_{\max}$ unit steps, which is therefore polynomial.

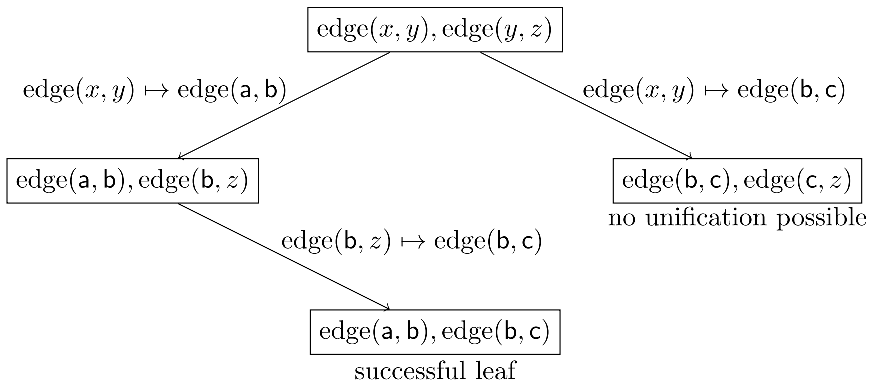



Figure 12: By searching this tree, the substitution $\{ x \mapsto a, y \mapsto b, z \mapsto c \}$ is found unifying all the negative literals of (104) with elements of (106).

So the agent *efficiently* learns conjunctions of range-restricted $st$-clauses on-line.

Due to Theorem 3.5, the agent also efficiently PAC-learn this hypothesis class.

## 4.2 Generalization of Clauses

So far, observations $o \in O$ the agent received from the environment were ground data encoded through logical interpretations. The latter were simply truth-assignment to propositional variables (which can be perceived as data features), or, in the first-order case, sets of observed ground facts. Now we consider a more interesting situation where observations are akin more to *knowledge* than *data*. Informally, the difference is that knowledge captures patterns from which multiple ground data can be inferred.

A natural way to encode pieces of knowledge in logic is through formulas. In the previous section it was convenient to adhere to clauses as a specific type of formulas, to represent learned knowledge. To represent observations through formulas, we will again stick to clauses.

To exemplify the situation through a simple example, consider that the environment tells the agent two principles through two positive observations ($s_1 = s_2 = 1$)

$$o_1 = \text{male}(x) \wedge \text{female}(y) \wedge \text{parent}(x, y) \rightarrow \text{daughter}(y, x) \tag{107}$$

$$o_2 = \text{female}(x) \wedge \text{parent}(\mathsf{ann}, x) \rightarrow \text{daughter}(x, \mathsf{ann}) \tag{108}$$

Both observations represent a rule that is true in the natural real-world interpretation. But neither of them reflects the entire truth — the rules are not general enough. The agent's goal is to find a joint generalization of the observations, which in this case would be

$$\gamma = \text{female}(x) \wedge \text{parent}(y, x) \rightarrow \text{daughter}(x, y) \tag{109}$$

The fact that $\gamma$ is indeed more general than both $o_1, o_2$ are a logical consequence of $\gamma$, i.e. they are entailed by $\gamma$, which we write as

$$\gamma \vdash o_1 \tag{110}$$

$$\gamma \vdash o_2 \tag{111}$$

Obviously, there are multiple clauses $\gamma$ satisfying (110,111). Another option would be e.g.

$$\gamma' = \text{parent}(y, x) \rightarrow \text{daughter}(x, y)$$

which in the real-world is an over-generalization. To prevent it, the agent should make sure that the hypothesized clause $\gamma$ does not entail *negative observations*. The environment could e.g. provide the negative example ($s_3 = 0$),

$$o_3 = \text{parent}(\mathsf{jack}, \mathsf{john}) \rightarrow \text{daughter}(\mathsf{john}, \mathsf{jack}) \tag{112}$$

It is indeed entailed by $\gamma'$, indicating the latter is an over-generalization.

There is however another way to prevent over-generalization. Recall that in Sections 2.1 and 4.1 we devised agents that did not need negative observations as they made the smallest possible generalization steps starting from the most specific hypothesis. If the smallest possible generalization had already been over-general, it simply meant that the target concept had no exact match in the agent's hypothesis space, i.e. (43) was not true. Here we will also design an agent making the smallest possible generalization steps.

To build the agent, we would like to adopt a policy based on entailment between clauses such as used in (110,111), which would thus be $\pi(\gamma, o) = 1$ if and only if $\gamma \vdash o$. Unfortunately, $\vdash$ is *undecidable* for general clauses $\gamma, o$.

Recall that in Section 2.2 we discussed syntactic subsumption between propositional clauses as an 'approximation' of the entailment relation. Syntactic subsumption is also defined for first-order clauses, is decidable, but is more involved than in the propositional case.

In the first-order case, clause $\gamma_1$ is said to *$\theta$-subsume* clause $\gamma_2$ if there is a substitution $\theta$ such that $\gamma_1\theta \subseteq \gamma_2$ where $\subseteq$ is with respect to the sets of literals on either side. This is denoted as

$$\gamma_1 \subseteq_\theta \gamma_2$$

For example, $\gamma \subseteq_\theta o_1$ with $\theta = \{ x \mapsto y, y \mapsto x \}$ as indeed

$$\{ \neg\text{female}(y), \neg\text{parent}(x, y), \text{daughter}(x, y) \} \\ \subseteq \\ \{ \neg\text{male}(x), \neg\text{female}(y), \neg\text{parent}(x, y), \text{daughter}(x, y) \}$$

Similarly, $\gamma \subseteq_\theta o_2$ with $\theta = \{ y \mapsto \mathsf{ann} \}$.

Two clauses $\gamma_1, \gamma_2$ are said to be *subsume-equivalent*, denoted $\gamma_1 \approx_\theta \gamma_2$ if $\gamma_1 \subseteq_\theta \gamma_2$ and $\gamma_2 \subseteq_\theta \gamma_1$. Clause $\gamma_1$ *strictly subsumes* clause $\gamma_2$, written as $\gamma_1 \subset_\theta \gamma_2$ if $\gamma_1 \subseteq_\theta \gamma_2$ but $\gamma_2 \not\subseteq_\theta \gamma_1$.

**Lemma 4.1.** *The relations $\subseteq_\theta, \approx_\theta, \subset_\theta$ are transitive, i.e. if $\gamma_1 \subseteq_\theta \gamma_2$ and $\gamma_2 \subseteq_\theta \gamma_3$, then $\gamma_1 \subseteq_\theta \gamma_3$ (and analogically for the other two relations).*

As in the propositional setting, subsumption implies entailment, that is to say $\gamma_1 \subseteq_\theta \gamma_2$ implies $\gamma_1 \vdash \gamma_2$, as discussed in Section 2.2. But again, the reverse implication holds only if $\gamma_1$ and $\gamma_2$ are not self-resolving. An important difference from the propositinal setting is that a first-order self-resolving clause need not be a tautology. For instance, consider the non-tautological clause

$$\gamma_1 = \text{natural}(x) \rightarrow \text{natural}(\mathsf{s}(x))$$

which can be interpreted to express that a successor of a natural number is also a natural number. Evidently, $\gamma_1$ logically entails

$$\gamma_2 = \text{natural}(x) \rightarrow \text{natural}(\mathsf{s}(\mathsf{s}(x)))$$

That is, $\gamma_1 \vdash \gamma_2$, however, $\gamma_1 \subseteq_\theta \gamma_2$ does not hold.

Note that in the above example, none of $o_1, o_2, \gamma$ are self-resolving so the decidable relation $\subseteq_\theta$ is indeed equivalent to $\vdash$ for any pair of these clauses.

Clause $\gamma_3$ is a *generalization* of clauses $\gamma_1$ and $\gamma_2$ if $\gamma_3 \subseteq_\theta \gamma_1$ and $\gamma_3 \subseteq_\theta \gamma_2$.

Clause $\gamma_3$ is a *least general generalization* of clauses $\gamma_1$ and $\gamma_2$ if it is their generalization and there is no other generalization $\gamma_4$ of the same clauses such that $\gamma_3 \subset_\theta \gamma_4$.

There may be multiple least general generalizations of one pair of clauses, but such generalizations are mutually subsume-equivalent. For example $\text{p}(x)$ and $\text{p}(x) \vee \text{p}(y)$ are both least general generalizations of $\text{p}(a)$ and $\text{p}(b)$, and indeed $\text{p}(x) \approx_\theta \text{p}(x) \vee \text{p}(y)$.

We shall now discuss how to compute a least general generalization of two clauses. The core of the procedure is the *anti-unification* procedure which can be seen as a complement to the unification procedure known from the first-order

**Algorithm 1** Anti-unification of two compatible atoms

**Require:** Atoms $a, b$ compatible with each other

$i = 0$; $\theta := \emptyset$; $\sigma := \emptyset$ ▷ *a counter and two substitutions*
$v_1, v_2, \ldots$ : variables not appearing in $a$ or $b$
**while** $a \neq b$ **do**
  Let $p$ be the leftmost position where $a$ and $b$ differ and $s$ and $t$ be the terms at this position in $a$ and $b$, respectively.
  **if** for some $j$ $(1 \leq j \leq i)$, $v_j\theta = s$ and $v_j\sigma = t$ **then** ▷ *variable already assigned*
    put $v_j$ to position $p$ in both $a$ and $b$ ▷ *replace the terms with that variable*
  **else**
    $i := i + 1$
    put $v_i$ to position $p$ in both $a$ and $b$ ▷ *replace the terms with a new variable*
    $\theta := \theta \cup \{ v_i \mapsto s \}$, $\sigma := \sigma \cup \{ v_i \mapsto t \}$ ▷ *store assignment of $v_i$*
  **end if**
**end while**
**return** $a$

logic resolution algorithm. Before exposing the algorithm, we need to establish a few notions.

Two *atoms* are *compatible* if they have the same predicate symbol and arity. Two *literals* are *compatible* if they have the same sign, predicate symbol, and arity.

Let $a$ be an atom of arity $n$. The *position* [i] in $a$ is the $i$-th argument place in $a$. The position $[i_1, i_2, \ldots k, l]$ is the $l$-th argument place in the term occurring at position $[i_1, i_2, \ldots k]$ in $a$. For example, the variable $x$ occurs at positions $[1]$ and $[2, 2, 1]$ in atom $a = \mathrm{p}(x, \mathsf{f}(\mathsf{c}, \mathsf{g}(x, \mathsf{d})))$. By putting variable $y$ to position $[2, 2]$ of $a$, we change $a$ into $\mathrm{p}(x, \mathsf{f}(\mathsf{c}, y))$.

Position $u$ is *left of* position $v$ if $u$ precedes $v$ in the lexical order, e.g. $[1, 3, 5]$ is left of $[1, 4]$.

The *anti-unification* of two compatible atoms is an atom produced by Algorithm 1. An example of the steps conducted by the Algorithm is shown in Tab. 4.

Now we shall see how to use the anti-unification algorithm defined for atoms to get a least general generalization defined for clauses. We first define the *selection set*, which picks all pairs of compatible literals from two input clauses.

The *selection set* $\mathsf{Sel}(\gamma_1, \gamma_2)$ of two clauses $\gamma_1, \gamma_2$ is thus defined as

$$\mathsf{Sel}(\gamma_1, \gamma_2) = \{ (l, m) \mid l \in \gamma_1, m \in \gamma_2, l \text{ is compatible with } m \}$$

In the next theorem, clauses are converted into atoms using the selection set and using $\bigvee$ formally as a predicate symbol so that the anti-unification procedure can be applied on such atoms. The resulting atom is then converted back to a clause.

| $i$ | $a$ | $\theta$ | $b$ | $\sigma$ |
|---|---|---|---|---|
| 0 | $\mathrm{p}(x, \mathsf{f}(\mathsf{a}, \mathsf{b}, \mathsf{g}(\mathsf{b}, \mathsf{a})), \mathsf{h}(\mathsf{a}))$ | $\emptyset$ | $\mathrm{p}(y, \mathsf{f}(\mathsf{b}, \mathsf{a}, \mathsf{g}(\mathsf{a}, \mathsf{a})), \mathsf{s}(\mathsf{a}))$ | $\emptyset$ |
| 1 | $\mathrm{p}(v_1, \mathsf{f}(\mathsf{a}, \mathsf{b}, \mathsf{g}(\mathsf{b}, \mathsf{a})), \mathsf{h}(\mathsf{a}))$ | $\{v_1 \mapsto x\}$ | $\mathrm{p}(v_1, \mathsf{f}(\mathsf{b}, \mathsf{a}, \mathsf{g}(\mathsf{a}, \mathsf{a})), \mathsf{s}(\mathsf{a}))$ | $\{v_1 \mapsto y\}$ |
| 2 | $\mathrm{p}(v_1, \mathsf{f}(v_2, \mathsf{b}, \mathsf{g}(\mathsf{b}, \mathsf{a})), \mathsf{h}(\mathsf{a}))$ | $\{v_1 \mapsto x,$ $v_2 \mapsto \mathsf{a}\}$ | $\mathrm{p}(v_1, \mathsf{f}(v_2, \mathsf{a}, \mathsf{g}(\mathsf{a}, \mathsf{a})), \mathsf{s}(\mathsf{a}))$ | $\{v_1 \mapsto y,$ $v_2 \mapsto \mathsf{b}\}$ |
| 3 | $\mathrm{p}(v_1, \mathsf{f}(v_2, v_3, \mathsf{g}(\mathsf{b}, \mathsf{a})), \mathsf{h}(\mathsf{a}))$ | $\{v_1 \mapsto x,$ $v_2 \mapsto \mathsf{a},$ $v_3 \mapsto \mathsf{b}\}$ | $\mathrm{p}(v_1, \mathsf{f}(v_2, v_3, \mathsf{g}(\mathsf{a}, \mathsf{a})), \mathsf{s}(\mathsf{a}))$ | $\{v_1 \mapsto y,$ $v_2 \mapsto \mathsf{b},$ $v_3 \mapsto \mathsf{a}\}$ |
| 4 | $\mathrm{p}(v_1, \mathsf{f}(v_2, v_3, \mathsf{g}(v_3, \mathsf{a})), \mathsf{h}(\mathsf{a}))$ | $\{v_1 \mapsto x,$ $v_2 \mapsto \mathsf{a},$ $v_3 \mapsto \mathsf{b}\}$ | $\mathrm{p}(v_1, \mathsf{f}(v_2, v_3, \mathsf{g}(v_3, \mathsf{a})), \mathsf{s}(\mathsf{a}))$ | $\{v_1 \mapsto y,$ $v_2 \mapsto \mathsf{b},$ $v_3 \mapsto \mathsf{a}\}$ |
| 5 | $\mathrm{p}(v_1, \mathsf{f}(v_2, v_3, \mathsf{g}(v_3, \mathsf{a})), v_4)$ | $\{v_1 \mapsto x,$ $v_2 \mapsto \mathsf{a},$ $v_3 \mapsto \mathsf{b},$ $v_4 \mapsto \mathsf{h}(\mathsf{a}))\}$ | $\mathrm{p}(v_1, \mathsf{f}(v_2, v_3, \mathsf{g}(v_3, \mathsf{a})), v_4)$ | $\{v_1 \mapsto y,$ $v_2 \mapsto \mathsf{b},$ $v_3 \mapsto \mathsf{a}$ $v_4 \mapsto \mathsf{s}(\mathsf{a})\}$ |

Table 4: An example of the anti-unification steps as conducted by Alg. 1.

**Theorem 4.2.** *Let $\gamma_1, \gamma_2$ be clauses and let $\bigvee(a_1, a_2, \ldots, a_n)$ be the anti-unification of atoms $\bigvee(l_1, l_2, \ldots, l_n)$ and $\bigvee(m_1, m_2, \ldots, m_n)$ where*

$$\{(l_1, m_1), (l_2, m_2), \ldots (l_n, m_n)\} = \mathsf{Sel}(\gamma_1, \gamma_2)$$

*Then $a_1 \vee a_2 \vee \ldots \vee a_n$, denoted $\mathsf{lgg}(\gamma_1, \gamma_2)$, is a least general generalization of $\gamma_1$ and $\gamma_2$.*

(Proof omitted)

Consider an example using $o_1, o_2$ from the beginning of this section. Here, the selection set is

$$\begin{aligned}\mathsf{Sel}(o_1, o_2) = \quad & \{(\neg\text{female}(y), \neg\text{female}(x)),\\ & (\neg\text{parent}(x, y), \neg\text{parent}(\mathsf{ann}, x)),\\ & (\text{daughter}(y, x), \text{daughter}(x, \mathsf{ann}))\}\end{aligned}$$

The anti-unification of

$$\bigvee(\neg\text{female}(y), \neg\text{parent}(x, y), \text{daughter}(y, x))$$

and

$$\bigvee(\neg\text{female}(x), \neg\text{parent}(\mathsf{ann}, x), \text{daughter}(x, \mathsf{ann})$$

is

$$\bigvee(\neg\text{female}(v_1), \neg\text{parent}(v_2, v_1), \text{daughter}(v_1, v_2))$$

So the $\mathsf{lgg}(o_1, o_2)$ is

$$\neg\text{female}(v_1) \vee \neg\text{parent}(v_2, v_1) \vee \text{daughter}(v_1, v_2)$$

| | $\neg\text{female}(x)$ | $\neg\text{parent}(\mathsf{ann}, x)$ | $\text{daughter}(x, \mathsf{ann})$ |
|---|---|---|---|
| $\neg\text{male}(x)$ | | | |
| $\neg\text{female}(y)$ | $\neg\text{female}(v_1)$ | | |
| $\neg\text{parent}(x, y)$ | | $\neg\text{parent}(v_2, y)$ | |
| $\text{daughter}(y, x)$ | | | $\text{daughter}(v_1, v_2)$ |

| $\theta$ | $\sigma$ | |
|---|---|---|
| $y$ | $x$ | $v_1$ |
| $x$ | $\mathsf{ann}$ | $v_2$ |

Table 5: Visualizing the $\mathsf{lgg}$ algorithm in a matrix form. The coordinates of the non-empty matrix entries correspond to the selections of compatible literals and the entries represent the anti-unified literals. The table to the right stores the substitutions established by successive anti-unifications of literals.

which can be transcribed into an equivalent but nicer form

$$\text{female}(x) \wedge \text{parent}(y, x) \rightarrow \text{daughter}(x, y)$$

The exemplified process can be visualized in a matrix form as in Table 5.

We state some basic properties of $\mathsf{lgg}$ without a proof.

**Lemma 4.3.** *Let $\gamma_1, \gamma_2, \gamma_3$ be clauses. Ten*

1. *If $\gamma_1 \subseteq_\theta \gamma_2$, then* $\mathsf{lgg}(\gamma_1, \gamma_2) \approx_\theta \gamma_1$
2. *(commutativity)* $\mathsf{lgg}(\gamma_1, \gamma_2) \approx_\theta \mathsf{lgg}(\gamma_2, \gamma_1)$
3. *(associativity)* $\mathsf{lgg}(\mathsf{lgg}(\gamma_1, \gamma_2), \gamma_3) = \mathsf{lgg}(\gamma_1, \mathsf{lgg}(\gamma_2, \gamma_3))$

Property 1 means simply that the least general generalization of a clause with a more general clause is just the more general clause or its equivalent.

Due to the commutative and associtative properties, an $\mathsf{lgg}$ of a finite set of clauses can be obtained by a repeated application of $\mathsf{lgg}$ to arbitrary clause-pairs from the set, always replacing the chosen pair with its $\mathsf{lgg}$, until the set has only one element. Since the order of such $\mathsf{lgg}$ applications is irrelevant, we can design an agent for learning from clauses which does not need to collect all clausal observations before jointly generalizing them. In other words, we can design an *on-line learning* agent for clausal observations, and need not resort to the batch-learning setting.

The difference from the generalizing agent in Section 4.1 is not only that the current agent's observations $o \in O$ are first-order logic clauses, rather than first-order logic interpretations. Also the hypothesis space of the current agent differs from the former one.

The current agent's hypothesis consists of a clause used for decisions and, as usually, a memory for the last-seen observation. So it is a pair $(\gamma, o)$ and the hypothesis space is

$$H = \Gamma \times O$$

The agent from Section 4.1 used a conjunction of clauses $\Phi$ (100) whereas the present agent decides by a single clause from $\Gamma$ only. On the other hand, the set (99) from which the former agent could form conjunctions was constrained to range-restricted *st*-clauses, whereas the present agent works with an unconstrained set $\Gamma$ of all clauses that can be constructed with given respective sets of predicate and function symbols, without assuming any bounds on the size of these sets or on the size or structure of the clauses in $\Gamma$.

The agent's decision policy makes a positive decision for an observation subsumed by its hypothesized clause $\gamma$,

$$y_k = \pi(\gamma, o_k) = \begin{cases} 1 \text{ if } \gamma \subseteq_\theta o_k \\ 0 \text{ otherwise} \end{cases} \tag{113}$$

So e.g. if (109) is the hypothesized clause $\gamma$, then observations (107) and (108) are decided positively, whereas $o_3$ (112) or another negative observation such as

$$o_4 = \text{male}(x) \wedge \text{female}(y) \rightarrow \text{daughter}(y, x)$$

would be decided negatively since $\gamma \not\subseteq_\theta o_3$, $\gamma \not\subseteq_\theta o_4$.

The agent starts with the initial clause that just copies the first positive observation. For simplicity and without loss of generality we assume that the first observation $o_1$ is positive (otherwise we would just let the agent discard observations until the first positive one comes). So

$$\gamma_1 = o_1 \tag{114}$$

The hypothesis update rule is similar to those used by the generalizing agents considered in Sections 2.1 and 4.1, prescribed by (51-53). However, the former two agents generalized by deleting literals or clauses (respectively) from their hypothesized conjunction. The present agent will instead generalize the hypothesized clause using the $\mathsf{lgg}$ operator. So

$$(\gamma, o'_k) = \mathcal{H}\left(\left(\gamma_{k-1}, o'_{k-1}\right), (o_k, r_k)\right) \tag{115}$$

is set to

$$o'_k = o_k \tag{116}$$

$$\gamma_k = \begin{cases} \gamma_{k-1} \text{ if } r_k = 0 \\ \mathsf{lgg}(\gamma_{k-1}, o'_{k-1}) \text{ otherwise} \end{cases} \tag{117}$$

For the current agent, we can show that the hypothesized clause is changed only after a making a wrong decision on a positive observation. In other words, the agent—just like both of the generalizing agents we have seen so far—never makes mistakes on negative observations.

To show this, we need to reiterate the basic assumption (43), which in the present context means that there is a clause $\gamma^* \in \Gamma$ exactly matching the concept $c$, i.e. $c(o) = 1$ ($o$ is a positive observation) if and only if

$$\gamma^* \subseteq_\theta o \tag{118}$$

Naturally, $\gamma^*$ is unknown to the agent. We carry on this assumption without mentioning it explicitly in the rest of the section.

Now we realize a fact analogical to Lemma 2.3.

**Lemma 4.4.** *Assuming (114), the clause $\gamma_k$ produced by $\mathsf{lgg}(\gamma_{k-1}, o'_{k-1})$ in (116) satisfies for all $k = 1, 2, \ldots$*

$$\gamma^* \subseteq_\theta \gamma_k \tag{119}$$

The lemma says that the agent never 'over-generalizes' by skipping over the target clause in terms of generality. This can be shown by mathematical induction.

*Proof.* In the inductive step, we assume that for $\gamma_{k-1}$ in (116) it holds $\gamma^* \subseteq_\theta \gamma_{k-1}$, and will show that this implies $\gamma^* \subseteq_\theta \gamma_k$.

First consider that $s_{k-1} = 0$. From (117) we know that $r_k \neq 0$, so $y_{k-1} \neq s_{k-1}$ (in plain words, since $\mathsf{lgg}$ was called, the previous decision must have been wrong), which means $y_{k-1} = 1$. By (113), this in turn implies that $\gamma_{k-1} \subseteq_\theta o'_{k-1}$. By Lemma 4.3 (Item 1) we have that $\gamma_k = \mathsf{lgg}(\gamma_{k-1}, o'_{k-1}) \approx_\theta \gamma_{k-1}$, in other words $\gamma_k$ and $\gamma_{k-1}$ are subsume-equivalent. But then the assumption $\gamma^* \subseteq_\theta \gamma_{k-1}$ implies $\gamma^* \subseteq_\theta \gamma_k$.

Next consider that $s_{k-1} = 1$. Considering (118), this implies $\gamma^* \subseteq_\theta o'_{k-1}$. Recall that for induction we assumed $\gamma^* \subseteq_\theta \gamma_{k-1}$. Assume now for contradiction that $\gamma^* \not\subseteq_\theta \gamma_k = \mathsf{lgg}(\gamma_{k-1}, o'_{k-1})$. But then $\mathsf{lgg}$ would not be a least general generalization, as follows from the definition of the latter on page 43 by identifying $\gamma_{k-1}, o'_{k-1}, \gamma_k, \gamma^*$ (in this order) with $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ in the definition. But that contradicts Theorem 4.2. So we have again that $\gamma^* \subseteq_\theta \gamma_k$.

We have proven the inductive step and it remains to prove the base case, i.e. that $\gamma^* \subseteq_\theta \gamma_1$. But this follows immediately from (114), the fact that $o_1$ is assumed to be positive, and (118). □

Now we are ready to show a lemma analogical to Lemma 2.4. In particular the current agent makes wrong decisions only for positive observations, and each such error leads to a new hypothesis that is strictly more general than (i.e., strictly subsumes) the previous one.

**Lemma 4.5.** *Assuming (114), if* $r_k \neq 0$*, then* $s_{k-1} = 1$ *and* $\gamma_k \subset_\theta \gamma_{k-1}$.

*Proof.* To see that $s_{k-1} = 1$, assume for contradiction that $s_{k-1} = 0$ (i.e., $o_{k-1}$ was a negative observation). If $r_k \neq 0$ (i.e. $y_{k-1}$ was a wrong decision) then $s_{k-1} \neq y_{k-1} = 1$. Then by (113), $\gamma_{k-1} \subseteq_\theta o_{k-1}$. From Lemma 4.4 (Equation 119), we also have $\gamma^* \subseteq_\theta \gamma_{k-1}$. Due to transitivity of subsumption (Lemma 4.1), this means $\gamma^* \subseteq_\theta o_{k-1}$. But then according to (118), $o_{k-1}$ was a positive observation, which is a contradiction.

Since $s_{k-1} = 1$ and $r_k \neq 0$, we have $y_{k-1} = 0$, so by (113), $\gamma_{k-1} \not\subseteq_\theta o_{k-1}$. Because $\gamma_k = \mathsf{lgg}(\gamma_{k-1}, o_{k-1})$ is a generalization of both of its arguments, $\gamma_k \subseteq_\theta \gamma_{k-1}$ and $\gamma_k \subseteq_\theta o_{k-1}$. Given that $\gamma_{k-1} \not\subseteq_\theta o_{k-1}$ and $\gamma_k \subseteq_\theta o_{k-1}$, it cannot be that $\gamma_{k-1} \approx_\theta \gamma_k$. So the subsumption $\gamma_k \subseteq_\theta \gamma_{k-1}$ must be strict, i.e. $\gamma_k \subset_\theta \gamma_{k-1}$. □

Lemma 4.4 established that the agent makes a strict generalization upon each mistake yet it never over-generalizes. It was exactly this reasoning that made us able to prove a mistake bound (Theorem 2.5) for the agent in Section 2.1, which also applied to the agent in 4.1. For these agents, the maximum number of generalization steps, and correspondingly the mistake bound, was $2n$ and $|\Gamma|$, respectively. These numbers were finite. Unfortunately, we cannot apply the same reasoning for the current agent, as the number of strict generalization steps from the initial clause towards the more general target clause does not have such a general finite bound. Indeed, consider for example the following infinite series of clauses[8] for $n = 2, 3, \ldots$

$$\gamma_n = \bigvee_{1 \leq i,j \leq n, i \neq j} \mathrm{p}(x_i, x_j)$$

So e.g.

$$\gamma_2 = \mathrm{p}(x_1, x_2) \vee \mathrm{p}(x_2, x_1)$$
$$\gamma_3 = \mathrm{p}(x_1, x_2) \vee \mathrm{p}(x_2, x_1) \vee \mathrm{p}(x_1, x_3) \vee \mathrm{p}(x_3, x_1) \vee \mathrm{p}(x_2, x_3) \vee \mathrm{p}(x_3, x_2)$$

and so on. We leave it to the reader to verify that $\gamma_2 \subset_\theta \gamma_3 \subset_\theta \ldots$.

Now let the target clause $\gamma^*$ be $\gamma^* = \gamma_2$. For any finite number $M \in N$, the environment can present a sequence $o_1 = \gamma_{M+3}, o_2 = \gamma_{M+2}, o_3 = \gamma_{M+1}, \ldots o_{M+1} = \gamma_3$ of $M + 1$ positive observations to the agent, causing it to generalize after each observation, therefore making $M + 1$ mistakes. Thus no finite number $M$ is a mistake bound for the current agent.

[8] Example adopted from Nienhuys-Cheng, de Wolf: Foundation of Inductive Logic Programming, Springer 1998

The example above follows from the fact that we do not bound the maximum size of clauses included in the lattice. The simple lattice we encountered earlier (Fig. 10) could also contain infinite paths if we alleviated the size bound $2n$ on the conjunctive elements in it. An intricacy distinguishing the subsumption lattice of size-unbounded first-order clauses from the latter lattice is that the infinite path $\gamma_2 \subset_\theta \gamma_3 \subset_\theta \ldots$, which grows in size (number of literals in $\gamma_n$), in fact connects two *small* elements in the lattice. More precisely

$$\mathrm{p}(x_1, x_2) = \gamma_2 \subset_\theta \gamma_3 \subset_\theta \ldots \subset_\theta \mathrm{p}(x_1, x_1)$$

Again, we leave it to the reader to verify that indeed $\gamma_n \subset_\theta \mathrm{p}(x_1, x_1)$ for any $n \geq 2$. So, speaking informally, the clause subsumption lattice is not just infinitely large but also infinitely *dense*.

Finally we will explore how the clausal formalism allows to build an agent that does not start learning from 'scratch', i.e. zero initial knowledge, but rather possesses some prior ('background') knowledge that just needs to be extended for making correct decisions. We motivate this situation through the following example where the environment provides two positive observations

$$o_1 = \mathrm{female}(x) \wedge \mathrm{father}(y, x) \rightarrow \mathrm{daughter}(x, y) \tag{120}$$

$$o_2 = \mathrm{female}(x) \wedge \mathrm{mother}(y, x) \rightarrow \mathrm{daughter}(x, y) \tag{121}$$

Because $\mathrm{father}(y, x)$ and $\mathrm{mother}(y, x)$ are not mutually compatible, the present agent would generalize $o_1, o_2$ using $\mathsf{lgg}$ into

$$\mathrm{female}(x) \rightarrow \mathrm{daughter}(x, y)$$

which is clearly unsatisfactory. Consider, however, that the agent has background knowledge in the form of a set of clauses (i.e., a *clausal theory*) $B$:

$$\mathrm{father}(x, y) \rightarrow \mathrm{parent}(x, y) \tag{122}$$

$$\mathrm{mother}(x, y) \rightarrow \mathrm{parent}(x, y) \tag{123}$$

Knowing $B$, the agent should be able to generalize $o_1, o_2$ into (109).

To formalize this idea, note that if $\gamma_1 \subseteq_\theta \gamma_2$ then the formula $\gamma_1\theta \rightarrow \gamma_2$ is a tautology, i.e.

$$\vdash \gamma_1\theta \rightarrow \gamma_2 \tag{124}$$

For example $\mathrm{p}(x) \subseteq_\theta \mathrm{p}(a) \vee \mathrm{q}(y)$ so $\vdash \mathrm{p}(a) \rightarrow (\forall y : \mathrm{p}(a) \vee \mathrm{q}(y))$. We shall account for background knowledge $B$ by making (124) relative to it, i.e.

$$B \vdash \gamma_1\theta \rightarrow \gamma_2 \tag{125}$$

So here $\gamma_1\theta \rightarrow \gamma_2$ is a tautological consequence of (is entailed by) $B$ rather than being a tautology.

This leads us to the following definition. We say that clause $\gamma_1$ *theta-subsumes* clause $\gamma_2$ *relative to* clause set $B$, written $\gamma_1 \subseteq_\theta^B \gamma_2$ if there is a substitution $\theta$ such that (125) holds. Clauses $\gamma_1, \gamma_2$ are *subsume-equivalent relative to B* if $\gamma_1 \subseteq_\theta^B \gamma_2$ and $\gamma_2 \subseteq_\theta^B \gamma_1$; we denote this as $\gamma_1 \approx_\theta^B \gamma_2$.

Furthermore, we define *relative least general generalization* (with respect to $B$) just as least general generalization (page 43), except that we replace the relation $\subseteq_\theta$ with $\subseteq_\theta^B$ in the definition.

Unfortunately, a relative least general generalization of two clauses with respect to an arbitrary clause set $B$ generally does not exist. However, it can be shown to exist for the special case that $B$ is finite and all clauses in it are just ground facts. This is summarized by the following theorem.

**Theorem 4.6.** *Let $\gamma_1, \gamma_2$ be clauses and $B$ a finite set of ground facts. Then $\mathsf{rlgg}_B(\gamma_1, \gamma_2)$ is a relative least general generalization of $\gamma_1, \gamma_2$ with respect to $B$, where*

$$\mathsf{rlgg}_B(\gamma_1, \gamma_2) = \mathsf{lgg}(\gamma_1 \vee_{l \in B} \neg l, \gamma_2 \vee_{l \in B} \neg l)$$

(Proof omitted)

Note that this theorem excludes, for example, the clauses (122, 123) we used as background knowledge in the motivating example as they are not ground facts. Instead, we shall exemplify the theorem with simpler background knowledge describing ground family relationships for persons we identify for brevity with constants $\mathsf{a}, \mathsf{b}, \mathsf{c}$, using predicates $\mathrm{m}/1, \mathrm{f}/1, \mathrm{p}/1$ with the informal meaning *male*, *female*, and *parent of* (respectively).

$$B = \{\ \mathrm{m}(\mathsf{a}), \mathrm{p}(\mathsf{a}, \mathsf{b}), \mathrm{f}(\mathsf{b}), \mathrm{p}(\mathsf{b}, \mathsf{c}), \mathrm{f}(\mathsf{c})\ \} \tag{126}$$

The agent should again learn the definition of the daughter relation. Equipped with background knowledge $B$, it receives the first positive observation of the daughter relation expressed through predicate $\mathrm{d}/2$

$$o_1 = \mathrm{d}(\mathsf{b}, \mathsf{a}) \tag{127}$$

and thus forms its first hypothesized clause $\gamma_1 = o_1 = \mathrm{d}(b, a)$. Once the second positive observation

$$o_2 = \mathrm{d}(\mathsf{c}, \mathsf{b}) \tag{128}$$

has been received, the agent should update its hypothesis with $\gamma_2 = \mathsf{rlgg}_B(\gamma_1, o_2)$. According to Theorem 4.6, this can be computed as the $\mathsf{lgg}$ of the clauses

$$\mathrm{d}(\mathsf{b}, \mathsf{a}) \vee \neg\mathrm{m}(\mathsf{a}) \vee \neg\mathrm{p}(\mathsf{a}, \mathsf{b}) \vee \neg\mathrm{f}(\mathsf{b}) \vee \neg\mathrm{p}(\mathsf{b}, \mathsf{c}) \vee \neg\mathrm{f}(\mathsf{c}) \tag{129}$$

$$\mathrm{d}(\mathsf{c}, \mathsf{b}) \vee \neg\mathrm{m}(\mathsf{a}) \vee \neg\mathrm{p}(\mathsf{a}, \mathsf{b}) \vee \neg\mathrm{f}(\mathsf{b}) \vee \neg\mathrm{p}(\mathsf{b}, \mathsf{c}) \vee \neg\mathrm{f}(\mathsf{c}) \tag{130}$$

| | d(c, b) | ¬m(a) | ¬p(a, b) | ¬f(b) | ¬p(b, c) | ¬f(c) |
|---|---|---|---|---|---|---|
| d(b, a) | $d(v_1, v_2)$ | | | | | |
| ¬m(a) | | ¬m(a) | | | | |
| ¬p(a, b) | | | ¬p(a, b) | | $\neg p(v_2, v_1)$ | |
| ¬f(b) | | | | ¬f(b) | | $\neg f(v_1)$ |
| ¬p(b, c) | | | $\neg p(v_3, v_4)$ | | ¬p(b, c) | |
| ¬f(c) | | | | $\neg f(v_4)$ | | ¬f(c) |

Table 6: Computation of the lgg of clauses (129,130) visualized as in Table 5.

| $\theta$ | $\sigma$ | new variable |
|---|---|---|
| b | c | $v_1$ |
| a | b | $v_2$ |
| b | a | $v_3$ |
| c | b | $v_4$ |

Table 7: Assignment of new variables to terms in clause (129) ($\theta$ column) and terms in clause (130) ($\sigma$ column) to new variables during the lgg computation as shown in Table 6.

The lgg computation is shown in Table 6 in the same way as was done in Table 5. Table 7 shows the two substitutions created in the process. The resulting clause is

$$\gamma_2 = d(v_1, v_2) \leftarrow m(\mathsf{a}) \wedge p(\mathsf{a}, \mathsf{b}) \wedge p(v_2, v_1) \wedge f(\mathsf{b}) \wedge \\ f(v_1) \wedge p(v_3, v_4) \wedge p(\mathsf{b}, \mathsf{c}) \wedge f(v_4) \wedge f(\mathsf{c})$$

While this is a correct relative least general generalization, it is evidently redundant. Informally, the ground facts appearing in the body of the clause (the conjunctive part to the right of $\leftarrow$) can be deleted as they are true due to $B$. Also, the body literals whose all variables do not appear in the head of the clause (to the left of $\leftarrow$) are redundant. Speaking precisely, $\gamma_2$ is subsume-equivalent (relative to $B$) to the clause

$$\gamma_2' = d(v_1, v_2) \leftarrow p(v_2, v_1) \wedge f(v_1) \tag{131}$$

which indeed represents the desired learned hypothesis. In formal notation, $\gamma_2 \approx_\theta^B \gamma_2'$. To prove this equivalence relation, we need to prove $\gamma_2 \subseteq_\theta^B \gamma_2'$ and $\gamma_2' \subseteq_\theta^B \gamma_2$.

The latter relation, which by definition (page 51) transcribes into

$$B \vdash (\gamma_2'\theta \rightarrow \gamma_2)$$

is evident, because $\gamma_2' \subset \gamma_2$, implying $\gamma_2' \vdash \gamma_2$. So $\gamma_2'\theta \rightarrow \gamma_2$ is a tautology for $\theta = \{\}$, meaning it is true in any model, not just in any model of $B$.

To demonstrate $\gamma_2 \subseteq_\theta^B \gamma_2'$, i.e.

$$B \vdash (\gamma_2\theta \rightarrow \gamma_2')$$

we set $\theta = \{ v_3 \mapsto v_1, v_4 \mapsto v_1 \}$. Then

$$\gamma_2\theta = \mathrm{d}(v_1, v_2) \leftarrow \mathrm{m}(\mathsf{a}) \wedge \mathrm{p}(\mathsf{a}, \mathsf{b}) \wedge \mathrm{p}(v_2, v_1) \wedge \mathrm{f}(\mathsf{b}) \wedge \mathrm{f}(v_1) \wedge \mathrm{p}(\mathsf{b}, \mathsf{c}) \wedge \mathrm{f}(\mathsf{c})$$

In any model of $B$, all of the ground literals of $\gamma_2\theta$ are true so by deleting them, we get a clause logically equivalent to $\gamma_2\theta$. But such a clause is exactly $\gamma_2'$ so $\gamma_2\theta \rightarrow \gamma_2'$ becomes $\gamma_2' \rightarrow \gamma_2'$ which is satisfied trivially.

The conversion of $\gamma_2$ into $\gamma_2'$ as shown above is an example of *clause reduction*.

We say that a clause $\gamma$ is *reduced* if for no clause $\gamma'$, $\gamma' \subset \gamma$, $\gamma' \approx_\theta \gamma$. A reduced clause $\gamma'$ is a *reduction* of a clause $\gamma$ if $\gamma' \approx_\theta \gamma$.

Similarly, a clause $\gamma$ is *reduced with respect to B* if for no clause $\gamma'$, $\gamma' \subset \gamma$, $\gamma' \approx_\theta^B \gamma$. A clause $\gamma'$ which is reduced with respect to $B$ is a *reduction* of $\gamma$ *with respect to B* if $\gamma' \approx_\theta^B \gamma$.

# 5 Learning Probabilistic Graphical Models

See the slide set on the CourseWare page.

# 6 Reinforcement Learning

See the slide set on the CourseWare page.