Groupe de Recherche en Informatique,
Image, Automatique et Instrumentation de Caen
Group "Models, Agents, Decision"

Habilitation thesis

Computational Aspects of Learning, Reasoning, and Deciding

Bruno Zanuttini

June 2011

Committee:

Michèle Sebag	Senior Scientist, CNRS, Université Paris Sud	reviewer
Craig Boutilier	Professor, University of Toronto, Canada	reviewer
Peter Jonsson	Professor, Linköpings Universitet, Sweden	reviewer
Jérôme Lang	Senior Scientist, CNRS, Université Paris Dauphine	
Patrice Perny	Professor, Université Pierre et Marie Curie	
Olivier Sigaud	Professor, Université Pierre et Marie Curie	
Abdel-Illah Mouaddib	Professor, Université de Caen Basse-Normandie	advisor

Bruno Zanuttini, https://www.greyc.fr/user/40
Group "Models, Agents, Decision", GREYC
Université de Caen Basse-Normandie
CNRS UMR 6072, ENSICAEN, France

Foreword and Acknowledgements

The work presented in this thesis summarizes some of the research I have being doing at the University of Caen for ten years, as a PhD student and then as an associate professor. It reports the aspects of this work which are closest to Artificial Intelligence and to the design of autonomous agents. This thesis however mostly focuses on ongoing research and middle-term projects, on which I hope to be able to work with Master and PhD students in the forthcoming years.

This work is at the crossroads of several subfields of research in computer science, in particular computational complexity, decision-making, and computational learning. I would not be able to present such a project without many colleagues, who accepted to work with me on these topics despite my initial ignorance of them. In that respect, my warmest thanks go to Jean-Jacques Hébrard, Nadia Creignou, Pierre Marquis, Abdel-Illah Mouaddib, Miki Hermann, Christian Bessière, Jérôme Lang, Gustav Nordh, Yann Chevaleyre, and Frédéric Koriche. I especially want to thank Abdel-Illah for my position and for being so trustful to me, and Fred for so many rich discussions, and for being always so enthusiastic.

I am also very grateful to the committee members for this habilitation. I am very honoured to have the occasion to share my thoughts and projects with you. Many thanks in particular to the reviewers for their careful reading, their suggestions, and their constructive critics.

An important part of this work owes a lot to Boris Lesner, who is such a smart and tireless PhD student. Almost all results in Chapters 5 and 6, but also my knowledge of the literature on factored decision processes, are essentially *his* work.

Let me however keep the most important part of my thanks for my wife, Gaëlle, and my son, Mano, who constantly support me in my work.

Contents

Chapter 1

Introduction

Contents

We start this thesis by settling the general context and motivation of our study. Section 1.5 also presents the contents and organization of the document.

1.1 Autonomous Agents and their Intellectual Abilities

This thesis is concerned with Artificial Intelligence, and its important subfield consisting in the design of *autonomous agents*. A prototypical example of an autonomous agent is a rover missioned to explore an unknown planet. Apart from physical abilities related to perceiving the environment, moving, storing data, communicating with Earth, etc., such a rover must be endowed with the ability to *decide* what to do at any moment. Such an ability distinguishes an autonomous rover from a remote controlled one, or from one with a predetermined sequence of actions to execute (may them be conditioned on many contingencies). Many situations indeed preclude remote control or predetermination of action sequences. In the case of another planet, communication, in particular remote control, is typically possible only during a short time window each day: the one where the planet and Earth "face each other". As concerns predetermination of actions to execute, contingencies must be known in advance. This is by definition unfeasible if the planet is unknown.

This typical application calls for methods which allow artificial agents to decide what to do per themselves. What we usually want is to design *rational* agents, that is, agents which make decisions so as to optimize some *utility function*. In the example of a rover, the utility function typically takes into account resource consumption (e.g., the more fuel remaining, the better), the amount of data sent to Earth (the more the better), broken pieces (the less the better), etc. The utility function is often known to the agent once and for all, but it may also have to be discovered during the mission. For instance, the utility of getting here or there may depend on the probability of finding new information about the planet at each precise location, which is typically not known in advance.

Clearly, for the decisions made to be optimal, they must take into account the precise state and dynamics of the environment of the agent, or at least some best possible approximation of them. When they are not known in advance, this calls for the ability of *learning* and *reasoning*. Typical things to be learnt by a rover on an unknown planet are a map of the planet, the accuracy of its sensors and actuators (how far does the camera see?, how slippery is the ground?, etc.), a model of exogenous events (how often does a meteorit fall?), possible or impossible configurations (is it

possible to face both the Earth and the Sun — so as to reload batteries while sending collected data?), etc. Reasoning typically takes place when evaluating the current state of the environment: observing smoke infer that there is certainly some fire burning, facing the Sun deduce that Earth is unreachable, etc.

To summarize, this thesis will focus on what can be called the "intellectual abilities" of autonomous agents (as opposed to their "physical abilities"): learning, reasoning, and deciding.

1.2 Interacting with Humans

An important part of the work and project presented here is motivated by scenarios in which an autonomous agent interacts with a human being. More precisely, we will depart from the illustrative case of an isolated rover on some unknown planet, and keep in mind scenarios where an agent serves a person, especially by accomplishing tasks for her[1].

An illustrative scenario We will use the following illustrative scenario in several parts of the thesis. Our agent is a software program, running as a daemon on a user's laptop, mobile phone, etc. Its role is to help the user scheduling its various obligations, such as meetings, administrative charges with or without deadlines, etc. For this, it may:

- (sensors) monitor the requests which the user receives for participating to meetings and choosing dates and times for them, monitor the user's answers to these requests (if any), access the user's todo-list, detect her logging in/out, etc.
- (actuators) ask questions to the user (e.g., "do you prefer meetings on weekdays or during the week-end?"), suggest answers to requests, suggest tasks to do at precise moments, communicate with other agents, etc.

Ideally, in organizations such as, say, a computer science department, every person would have such an agent running, and most of the time the agents would arrange schedules so that every user is happy with their decisions. But another desirable feature of such a system is that agents need not be calibrated for each person. What we want is initially homogeneous agents, each of which adapts its behaviour to a user from the moment it is attached to her (installed on her machine).

Another important point in this example is the interaction between agents (attached to different persons). Typically, such interactions are studied in the paradigm of multiagent systems. As concerns this example, models of negociation, argumentation, notions from game theory and collective decision theory would be particularly relevant. However we will not consider such issues in this thesis, and rather focus on one agent, modelling other agents in the environment. Said otherwise, we will not assume that other persons involved in meetings also have an agent acting on their behalf, but rather than they are part of an (initially) unknown and unmodelled environment.

Problems raised by human-agent interaction What is specific to interacting with a user? Several points are important to be raised here.

First, humans cannot be assumed to be rational. Even knowing the precise actions available to them and their (current) utility function, it is not reasonable to assume that they will always choose the optimal actions. From the viewpoint of an agent, this makes part of its environment highly unpredictable. This may be the case because the user is indeed irrational, but also, simply because her actions and utility function are not fully known to the agent, which makes her *appear* as being irrational. In our scenario, typically the agent will not be told that the user is in a bad mood today, though this will certainly impact her (even optimal) decisions[2].

Second and in the same vein, it is unreasonable to assume that the user can be assigned a static model. Humans tend to change their mind, their preferences, and their abilities. Having a

[1] Throughout the document, we consistently use "the user", "she", and "her" to refer to the human being, and "it" to refer to the agent.

[2] Ideally, the agent will be able to *learn* to recognize when the user is in a bad mood.

child born typically changes a user's priorities forever; having her car unusable typically changes her ability to be at work at such and such moment until the problem is fixed, etc. This problem is particularly important in our scenario where the same agent is intended to serve the same person for a long time, say, over years, and it precludes the possibility to spend some time learning a model of the user once and for all, then acting optimally with respect to this model. It calls for being able to continuously adapt to her.

Third, it is widely admitted that humans are not at ease with numerical utilities. It is rather easy for a user to decide between, say, a meeting on Monday and one on Tuesday, but difficult if not impossible to give a value of 10 to Monday and 5 to Tuesday. Similarly, it is difficult for a user to understand a statement like "other members of the department want to organize a meeting with you on some odd day between 1 and 5"; it is easier for her to understand "... on Monday, Wednesday, or Friday". From the viewpoint of the agent, this calls for the ability to communicate with the user in a symbolic rather than numerical language. Note that this does not preclude the agent to use numbers internally, only to rely on them for interacting with the user.

Fourth, a user is typically interested in knowing why the agent behaved so and so. This will especially be the case if the agent made a decision on her behalf, and the user does not agree with it, or if the agent is unable to suggest, say, a date for a particular meeting. In such cases the agent must be able to communicate with her about the reason why it chose such solutions, or why it could not find any.

Lastly, users cannot be queried again and again, just as the model of a system could be simulated. Users will typically be reluctant to answer queries originated by the agent. At least, they will choose the moments when they are indeed prone to answer them. This calls for agents which are able to learn mostly from passive observations rather than being taught the user's preferences.

1.3 Logic and Algorithms

In this thesis we mainly focus on *propositional* representations of the environment, of the user's preferences, etc. That is, we assume that what the agent perceives from its environment is described through a set of 0/1 variables (or attributes, descriptors). This is clearly a restrictive description language. In particular, continuous values such as positions, velocities, temperatures, etc. cannot be faithfully represented. The typical way to handle such values is to *discretize* them into a finite set of values. Finite sets themselves can then be encoded by propositional variables, as is commonly done for search and optimization problems (Walsh, 2000). Maybe more importantly, in propositional logic statements about unknown, unbounded sets of objets cannot be expressed, contrary to first-order logic. A typical domain of interest where this is restrictive is planning: though planning problems are naturally represented in first-order logic, as in the PPDDL language (Younes and Littman, 2004), propositional representations must rely on *grounding*, that is, they must instanciate the problem with the objects at hand and find a plan specific to them.

We acknowledge these restrictions and the important areas of AI which are concerned with handling more expressive languages. Nevertheless, we stick to propositional representations because they provide a convenient framework for studying algorithms, whereas continuous variables raise representation issues, and first-order logic often leads to undecidable problems. Moreover, in practice there are a lot of algorithms which are able to handle huge propositional representations, as is the case for the satisfiability problem and variants, for planning, etc. (Biere et al., 2009). This leaves hope that nontrivial computational problems can be solved with grounded representations. Finally, results about the properties and algorithmics of grounded languages typically settle bases for studying the algorithmics of higher-level languages. A good example of this is given by the recent excitement about solving planning problems directly at the relational level: most techniques introduced are more or less direct generalizations of techniques elaborated for grounded representations (Wang et al., 2008; Sanner and Boutilier, 2009). In our opinion, such techniques may not have been developed directly at the first-order level, simply because the propositional setting allows one to focus only on combinatorial concerns (without having, for instance, decidability as another subject of focus).

Indeed, we are interested here in a complexity-theoretic and algorithmical point of view. This thesis will study computational problems and the existence of algorithms for solving them efficiently, mainly in terms of complexity but also in practice. When such algorithms will fail to show up, we will be interested in the likelihood of their existence, and will investigate whether the problems are hard in terms of complexity (e.g., NP-hard, PSPACE-hard, etc.).

In that respect, a specific framework will prove interesting for studying problems and providing efficient algorithms, namely the framework of tractable fragments. The tractable fragments of propositional logic are well-studied for many problems, especially reasoning problems. In particular, the theory of clones, developed by Post in the first half of the XX$^{\text{th}}$ century (Post, 1941) and revisited in the mid 1990's in the context of computational complexity (Jeavons et al., 1997), provides a wide, and in some sense complete, list of interesting fragments for most tasks in AI. Other representational frameworks used in this thesis are those of ordinal preferences, in particular Conditional Preference Networks (CP-nets, Boutilier et al. 2004), and of planning languages, especially the Probabilistic Planning Domain Description Language (PPDDL, Younes and Littman 2004).

1.4 Putting it All Together

Summarizing, we will be concerned here with providing agents with algorithms for learning, deciding, and reasoning. Our agents will be assumed to perceive their environment through propositional variables, representing the outputs of their sensors, or dicretizations and Boolean encodings of these. They will also be allowed to control actuators, and again, controls will be assumed to take their values in finite, known sets. The environment will often contain a particular object, namely a human being, and communication with her will be possible through observation and through specific actuators allowing to ask questions.

We will not impose *a priori* restrictions on the languages in which our agents will be allowed to store and manipulate their knowledge. Nevertheless, most representations which we will consider will build on propositional logic and probabilities. Symbolic preferences will play a special role in knowledge about the human.

Hence we will investigate:

- concept learning issues in the sense of computational learning theory,
- preference elicitation issues,
- decision-making issues, in the sense of policy computation for Markov decision processes and reinforcement learning,
- more fundamental issues about the fragments of propositional logic.

Why these investigations are important for the design of autonomous agents, and how they fit together, is discussed in a dedicated chapter of this thesis.

1.5 Contents and Organization of the Document

This document presents a research project rather than a compilation of existing results. We chose to settle the bases for studying general issues and more precise questions which are currently open, or unsatisfactorily addressed in the literature. These questions will of course derive from work already accomplished, by the author or by others, and we give surveys of existing work in the related areas. For each research direction suggested we precisely state the issues, questions, and motivation for them, and give first results or examples of results which in our opinion make the study promising.

Some contributions are nevertheless reported from published or submitted articles, mainly because they serve the purpose of settling bases for the corresponding research project. These are:

- an original algorithm for decision-making under uncertainty, using an extended version of action-value functions, which in turn opens perspectives for policy revision (joint work with Boris Lesner in the context of his PhD (Lesner and Zanuttini, 2011), described in Chapter 5),
- a theoretical study of the problem of learning actions from indirect observations of their effects, through the possibly ambiguous transitions which they provoke in the environment (ongoing joint work with Boris Lesner, described in Chapter 6),
- negative results and near-optimal algorithms for learning ordinal preferences on combinatorial domains, using observation and possibly queries to the user (joint work with Frédéric Koriche (Koriche and Zanuttini, 2009, 2010), described in Chapter 7),
- the introduction and study of new theoretical tools for studying the complexity of problems with Boolean formulas as instances, when reductions are desired which preserve the number of variables and hence, the "exact complexity" of problems (joint work with Gustav Nordh (Nordh and Zanuttini, 2009), described in Chapter 8).

Building on this work and others, we propose several research projects, the main of which are:

- the study of reinforcement learning problems in settings where the agent has no other prior knowledge, nor feedback from the environment, than the transitions which its trials-and-errors provoke (Chapter 6),
- the study of decision-making problems for an agent which serves a human being, when this human being has ordinal preferences on the agent's realizations and these are not known *a priori* to the agent (Chapter 7),
- the development of new theoretical tools for studying the complexity of problems (Chapter 8),
- the study of algorithms for learning Boolean concepts which are able to automatically change bias when needed, and the corresponding formalizations (Chapter 9).

Each of these studies addresses a specific problem, but all together they serve the general purpose of designing agents which are able to decide in attribute-value environments. We describe this point of view in details in Chapter 4.

Organization To summarize the organization of this document, Chapters 2 and 3 give the necessary preliminaries on symbolic representations and computational problems. Because the relevant notions are briefly recalled at the beginning of each other chapter, they may serve as references instead of being read as a whole.

Chapter 4 places all the components of this thesis (representations, problems, and solutions) in the context of the intellectual abilities of an agent as described in Section 1.1. This chapter is necessary to understand how the different studies presented and suggested are articulated with each other. Then Chapters 5–9 present our contributions and research projects. These chapters are essentially self-contained and independent from each other, in particular they can be read in any order. Finally, Chapter 10 concludes and gives further directions of research.

Throughout the thesis, we point at various open questions, which are labelled as such. These are minor research questions, the answers to which have few chances to yield significant advances in Artificial Intelligence (at least, in our opinion). On the other hand, at the end of each chapter one or several sections (labelled "Perspective") point at wider research questions, which constitute real research projects with the potential to have a more significant impact. In our opinion, each of these would deserve a master's thesis and/or a PhD. We summarize the corresponding subjects in Chapter 10.

Chapter 2

Preliminaries on Representations

Contents

This chapter gives the necessary background on the main representation languages which we will use throughout this thesis, namely propositional logic and its tractable fragments (Sections 2.1 and 2.2), conditional preference networks (Section 2.3), and Markov decision processes and their factored representations (Section 2.4). All these representation languages have in common to be based on a set of propositional variables as their main component.

2.1 Propositional Logic

We assume a countable set of *propositional variables*, that is, variables which can take values in the Boolean domain $D = \{0, 1\}$. We denote variables by $x, x_1, x_i \ldots$. Throughout the thesis, variables are consistently interpreted as *descriptors* of the environment or *attributes* of objects. In a particular situation, for instance a precise state of the environment as perceived by an agent, the number of variables manipulated will always be finite and known to the agent: for instance, they are the outputs of its sensors. Countable infinity is used to allow the agent (and, in a more abstract setting, reductions between computational problems) to introduce an unbounded number of fresh variables whenever needed.

An *assignment* to a set of propositional variables X is a mapping from X to $\{0, 1\}$, usually denoted by $\mu, \mu_1, \mu_i \ldots$. We write 2^X for the set of all assignments to X, and $\mu[x_i]$ or $\mu[i]$ for the

value assigned by μ to the variable x_i. We also write $\mu[x/v]$ for the assignment μ' equal to μ on all variables but x, and which assigns v to x; for instance, $11111[x_4/0]$ denotes the assignment 11101. Given an assignment μ to X and a subset X' of X, we write $\mu_{|X'}$ for the assignment to X' defined by $\mu_{|X'}[x] = \mu[x]$ for all $x \in X'$. Finally, given two assignments μ, μ' to disjoint sets of variables X, X' (respectively), we write $\mu.\mu'$ for the assignment to $X \cup X'$ which extends both.

If X is the set of all descriptors which the agent uses for its environment, an assignment to X represents a precise state of this environment, as perceived by the agent. Precisely, such an assignment μ represents the *equivalence class* of all states of the environment which the agent perceives as μ. To emphasize this interpretation as states, we write $s, s', s_i \ldots$ instead of $\mu, \mu', \mu_i \ldots$. Similarly, if X is the set of attributes used by the agent to describe objects, an assignment to X denotes a particular object, as perceived by the agent. We distinguish two assignments, written $\mu_{\vec{0}}$ and $\mu_{\vec{1}}$, which assign 0, respectively 1, to all the variables in X (the set X will always be clear from the context).

2.1.1 Formulas

The syntax of propositional logic is given by *propositional formulas*. We will use the connectives $\neg, \vee, \wedge, \rightarrow, \exists$ with their usual interpretation. Variables x and negated variables $\neg x$ (usually denoted by $\overline{x}$) are called *positive literals* and *negative literals*, respectively. Literals are usually denoted by $\ell, \ell_1, \ell_i \ldots$, and the complementary literal of ℓ is denoted by $\overline{\ell}$, that is, $\overline{\ell}$ is $\overline{x}$ for $\ell = x$, and $\overline{\ell}$ is x for $\ell = \overline{x}$.

A *clause* is a finite disjunction of literals of the form $(\ell_1 \vee \cdots \vee \ell_k)$, usually written $c, c_1, c_i \ldots$. Observe that a clause of the form $(\overline{x}_1 \vee \cdots \vee \overline{x}_p \vee x_{p+1} \vee \cdots \vee x_n)$ is logically equivalent to the *rule* $(x_1 \wedge \cdots \wedge \cdots x_p \rightarrow x_{p+1} \vee \cdots \vee x_n)$. A *term* is a finite conjunction of literals of the form $(\ell_1 \wedge \cdots \wedge \ell_k)$, usually written $t, t_1, t_i \ldots$. Abusing notation, we write 3^X for the set of all terms t with variables from X. A propositional formula is any well-formed formula on the variables, connectives, and parentheses, denoted by $\varphi, \varphi_1, \varphi_i \ldots$.

A propositional formula φ is said to be in *conjunctive normal form* (CNF) if it is written as a finite conjunction of clauses, of the form $c_1 \wedge \cdots \wedge c_k$. Dually, it is said to be in *disjunctive normal form* (DNF) if it is written as a finite disjunction of terms, of the form $t_1 \vee \cdots \vee t_k$. Intuitively, the CNF is best suited for expressing constraints and expert knowledge, because these are usually given as rules all of which are true (or must be true for search problems). Dually, the DNF is typically best suited for expressing *desires*, because it allows a user to specify directly a list of desired (groups of) objects.

Example 2.1 *The formula* $\varphi = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (x_2 \vee x_3)$ *is in CNF and contains* 3 *clauses. Its first clause can be interpreted as the rule* $x_3 \rightarrow x_1 \vee x_2$. *Now* $\varphi' = \exists x_2 (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (x_2 \vee x_3)$ *is also a formula but it is not in CNF.*

2.1.2 Semantics

If φ is a propositional formula, we write $\mathit{Vars}(\varphi)$ for the set of all variables which occur in φ. Note that this is syntax-dependent, hence φ may not depend on all the variables in $\mathit{Vars}(\varphi)$. For instance, $\varphi = (x_1) \wedge (\neg x_1)$ is such that $\mathit{Vars}(\varphi) = \{x_1\}$ but depends on no variable, since it is unsatisfiable. Given a propositional formula φ and some set of variables $X \supseteq \mathit{Vars}(\varphi)$, a *model of* φ *over* X is an assignment μ to X which makes φ evaluate to true under the standard semantics of the connectives. We say that μ *satisfies* φ, and write $\mu \models \varphi$. We usually consider the models of φ over $\mathit{Vars}(\varphi)$, and we write $M(\varphi)$ for the set of all such models.

A formula φ is said to be *satisfiable* if it has at least one model ($M(\varphi) \neq \emptyset$), and *unsatisfiable* otherwise. It is said to *entail* a formula φ', written $\varphi \models \varphi'$, if each model of φ over $\mathit{Vars}(\varphi) \cup \mathit{Vars}(\varphi')$ satisfies φ'. We say that φ and φ' are *logically equivalent* (or simply *equivalent*), written $\varphi \equiv \varphi'$, if φ entails φ' and φ' entails φ.

Example 2.2 (continued) *The formula* φ *in Example 2.1 satisfies* $\mathit{Vars}(\varphi) = \{x_1, x_2, x_3\}$, *and has* $M(\varphi) = \{010, 011, 101\}$ *as its set of models, where, e.g.,* 010 *is interpreted as the assignment*

of 0 *to* x_1, 1 *to* x_2, *and* 0 *to* x_3. *It also has* $\{0100, 0101, 0110, 0111, 1010, 1011\}$ *as its set of models over* $\{x_1, x_2, x_3, x_4\}$. *The formula* φ' *has* $M(\varphi') = \{00, 01, 11\}$ *as its set of models (over* $Vars(\varphi') = \{x_1, x_3\}$*). We have* $\varphi \models (x_1 \vee x_2)$, *but* $\varphi \not\equiv (x_1 \vee x_2)$ *(the assignment* $\mu = 100$ *satisfies* $(x_1 \vee x_2)$ *but not* φ*).*

2.2 Tractable Fragments and Post's Lattice

A natural generalization of the notion of a clause is the notion of a *constraint*, as used in constraint satisfaction problems (Rossi et al., 2006). Considering the class of formulas which use constraints from a fixed language yields a now well-developed theory which allows for classifying the complexity of many problems in an exhaustive manner.

2.2.1 Relations, Languages, and Formulas

If we abstract from the particular variables used, a clause can be defined by the sign of the literals in it and by its arity. For instance, the clause $(\overline{x}_1 \vee x_2 \vee x_3)$ can be defined as the ternary *relation* $\{000, 001, 010, 011, 101, 110, 111\}$ *applied* to the sequence of variables (x_1, x_2, x_3).

In general, a (Boolean) *relation* R of arity $k \in \mathbb{N}$ is any subset of $\{0,1\}^k$. The elements in a relation are called (k-ary) *tuples*, usually written $t, t_1, t_i \ldots$. We write $t[i]$ for the ith coordinate of tuple t. A k-ary relation R and a sequence of variables $(x_{i_1}, \ldots, x_{i_k})$ together define the *constraint* $R(x_{i_1}, \ldots, x_{i_k})$, of which R is called the *relation* and $(x_{i_1}, \ldots, x_{i_k})$ the *scope*. Constraints are usually denoted like clauses, by $c, c_1, c_i \ldots$, with the context making the distinction from clauses clear. Importantly, nothing in the definition requires that the variables are distinct, that is, $i_j = i_{j'}$ can hold for some $j \neq j'$.

Example 2.3 *An example binary relation is* $R_\rightarrow = \{00, 01, 11\}$, *and an example ternary relation is* $R_\oplus = \{001, 010, 100, 111\}$. *Example constraints built on them are* $R_\rightarrow(x_2, x_3)$ *and* $R_\oplus(x_1, x_2, x_3)$, *which are logically equvialent to* $(x_2 \rightarrow x_3)$ *and* $x_1 \oplus x_2 \oplus x_3 = 1$, *respectively. Another example constraint is* $R_\oplus(x_1, x_2, x_1)$, *which is logically equivalent to* x_2.

Some specific relations will be useful at many places. The unary relations R_T and R_F are defined to be $\{1\}$ and $\{0\}$, respectively. In words, the constraint $R_T(x)$ (resp. $R_F(x)$) forces the variable x to be assigned true (resp. false). The binary relation $R_=$ is defined to be $\{00, 11\}$, that is, the constraint $R_=(x_1, x_2)$ forces the variables x_1 and x_2 to take the same value. We usually write $x_1 = x_2$ for such a constraint.

A set of (Boolean) relations, possibly of different arities, and possibly infinite, is called a (Boolean) *language*, and is usually denoted by $\Gamma, \Gamma_1, \Gamma_i \ldots$. Given a language Γ, a Γ-*formula* is a finite conjunction of constraints of the form $c_1 \wedge \cdots \wedge c_k$, where for $i = 1, \ldots, k$ the relation of c_i is in Γ or is the binary relation $R_=$. Why the relation $R_=$ is always allowed in Γ-formulas will be made clear a little later in this section, but for the moment it is enough to observe that for most computational problems, two variables x_1, x_2 in the scope of a constraint $x_1 = x_2$ in some Γ-formula φ can be managed by replacing one with the other everywhere in φ.

Finally, an assignment μ is said to be a *model* (or to *satisfy*) a Γ-formula φ if for all constraints $R(x_{i_1}, \ldots, x_{i_k})$ in φ, the tuple $(\mu[i_1], \ldots, \mu[i_k])$ is in R. The definitions of satisfiability, entailment, and logical equivalence are naturally extended from propositional formulas, as well as entailment and logical equivalence between propositional and Γ-formulas.

Example 2.4 (continued) *An example language is* $\Gamma = \{R_\rightarrow, R_\oplus\}$. *The formula*

$$\varphi = R_\rightarrow(x_2, x_3) \wedge R_\oplus(x_1, x_2, x_3) \wedge x_3 = x_4 \wedge R_\oplus(x_5, x_6, x_5)$$

is a Γ*-formula, with set of models* $M(\varphi) = \{001101, 001111, 100001, 100011, 111101, 111111\}$.

The distinction between finite and infinite languages Γ will be important in several places, for at least two reasons. The first of these is the problem of encoding Γ and recognizing Γ-formulas.

When Γ is finite, many reasonable presentations of it may be used, and in particular the one giving the set of all relations, in some arbitrary order, with each relation encoded by the set of its tuples (in *extension*). Moreover, for finite Γ any Γ-formula can be encoded by the set of its constraints, where the relation of each constraint is given in extension, or by a relation name associated to it once and for all. Finally, given a formula with relations presented in extension, deciding whether this formula is a Γ-formula for a given, finite language Γ can be decided straightforwardly.

Things are more complex for infinite languages Γ. Any formula can still be encoded with relations given in extension (because formulas are defined to contain a finite number of constraints, the relations of which have finite arities), but it is not obvious to decide whether an arbitrary formula is a Γ-formula if Γ is infinite. What we need is an (efficient) algorithm for Γ, which takes a relation given in extension as input and decides whether this relation is in Γ. As we will see however, for most languages of interest such an efficient algorithm indeed exists.

The second problem with infinite languages pertains to reductions between problems. We elaborate on this in the next paragraph, but we will anyway make explicit whether we are considering a finite or infinite language Γ whenever needed throughout the document.

2.2.2 Implementations and Co-Clones

For many computational problems, and for the problem of deciding satisfiability of formulas in the first place, it can be shown that finite languages can be grouped into *equivalence classes*, in such a way that for two languages Γ_1, Γ_2 in the same class, the computational problem at hand has the same complexity with its inputs restricted to Γ_1-formulas or to Γ_2-formulas. The term "same complexity" is intended here modulo logspace, and hence polynomial-time equivalence.

The equivalence class of a language consists of those languages which generate the same *co-clone* (or *relational clone*). This notion is defined using the following binary relation on finite languages.

Definition 2.5 (implementation) *A finite language Γ_1 is said to* implement *a finite language Γ_2 if for all relations R of arity k in Γ_2 and for any sequence of k variables X, there is a set of variables X' disjoint from X and a Γ_1-formula φ over $X \cup X'$ such that the constraint $R(X)$ is logically equivalent to the formula $\exists X'\varphi$. The formula $\exists X'\varphi$ is usually called a* primitive positive definition *of $R(X)$.*

Example 2.6 (continued) *Let $R_\vee$ be the binary relation $\{01, 10, 11\}$. Then $\Gamma = \{R_\to, R_\oplus\}$ implements $R_\vee$, as witnessed by the equivalence $R_\vee(x_1, x_3) \equiv \exists x_2 x_4 x_5 x_6 \varphi$ with φ defined in Example 2.4. It also implements R_T as witnessed by $R_T(x_1) \equiv R_\oplus(x_2, x_1, x_2)$.*

In words, Γ_1 implements R if it is able to express it using conjunction and existential quantification (and possibly the equality relation and duplicated variables in the scope of constraints). Hence, if Γ_1 implements Γ_2, any Γ_2-formula φ_2 can be replaced with a logically equivalent formula of the form $\exists X'\varphi_1$, where φ_1 is a Γ_1-formula (this can be done by conjuncting the implementations of all relations used in Γ_2 and rearranging the resulting formula). Clearly enough, if Γ_2 is finite, implementations of each relation in it by Γ_1 can be stored in a lookup table, so that for fixed, finite Γ_2 this transformation is feasible in time linear in the size of φ_2, with one lookup per constraint. Importantly, while doing this we have introduced an existential quantifier. But for the purpose of satisfiability testing, it is clear that the formula $\exists X'\varphi_1$ is satisfiable if and only if so is φ_1, so that the transformation above gives a (logspace) reduction from satisfiability testing of Γ_2-formulas to satisfiability testing of Γ_1-formulas.

Example 2.7 (continued) *The language $\Gamma = \{R_\to, R_\oplus\}$ implements $R_\vee$, R_T (see Example 2.6), and obviously it implements $R_\oplus$. Hence with $\Gamma' = \{R_\oplus, R_\vee, R_T\}$, the problem $\mathrm{SAT}(\Gamma')$ is reducible in logarithmic space to the problem $\mathrm{SAT}(\Gamma)$. As an example, the formula $R_\oplus(x_7, x_8, x_9) \wedge R_T(x_{11}) \wedge R_\vee(x_1, x_3)$ is logically equivalent to the formula $\exists x_2 x_4 x_5 x_6 x_{10} R_\oplus(x_7, x_8, x_9) \wedge R_\oplus(x_{10}, x_{11}, x_{10}) \wedge \varphi$ with φ defined in Example 2.4. Hence the former formula is satisfiable if and only if the formula $R_\oplus(x_7, x_8, x_9) \wedge R_\oplus(x_{10}, x_{11}, x_{10}) \wedge \varphi$ also is.*

This reduction is the basic idea behind the use of implementations in analyzing the complexity of problems. Observe that introducing the auxiliary variables in X' and/or the existential quantifier is not harmless for all problems. We will come back to this issue in depth in Chapter 8.

In can be shown rather easily that implementations compose together into implementations, and obviously any finite language implements itself. This is enough for showing that implementations group finite languages together in closed classes.

Definition 2.8 (co-clone) *A* co-clone C *is any set of relations* Γ *which is closed under implementations. The* co-clone of Γ*, written* $\langle\Gamma\rangle$*, is defined to be the set of all relations which* Γ *implements.*

Observe that due to the definition of implementations, the equality relation $R_=$ is in all co-clones.

What makes co-clones easily usable for classifying the complexity of problems is a very nice Galois connection between co-clones and classes of Boolean functions. Precisely, it can be seen rather easily that co-clones form a lattice, ordered by inclusion, and with meet and join operations defined respectively by $C_1 \sqcap C_2 = C_1 \cap C_2$ and $C_1 \sqcup C_2 = \langle C_1 \cup C_2\rangle$. This lattice is in Galois connection through the operations *Inv* and *Pol* (to be defined soon), with a lattice of classes of Boolean functions defined as follows.

A k-ary Boolean function $f : \{0,1\}^k \to \{0,1\}$ is said to be a *projection* if there is an index $i \in \{1,\ldots,k\}$ such that $f(x_1,\ldots,x_k)$ is x_i for all $x_1,\ldots,x_k$. Moreover, a set of Boolean functions C_f is said to be *closed under superposition* if for any function $f \in C_f$, the functions obtained from f by permuting its arguments, identifying one of its arguments to another, or adding irrelevant arguments, are also in C_f, and if for any two functions f, f', their composition $f \circ f'$, where f' takes the place of any argument of f, is also in C_f. Then C_f is said to be a *clone* if it contains all projections and is closed under superposition.

Example 2.9 *The class of all monotone Boolean functions, i.e., of all* f *such that* $f(x_1,\ldots,x_k) \geq f(x'_1,\ldots,x'_k)$ *holds if* $x_1 \geq x'_1,\ldots,x_k \geq x'_k$ *all hold, is a clone. As examples, the first* k*-ary projection defined by* $f(x_1,\ldots,x_k) = x_1$ *is clearly monotone. Now if* f, f' *are monotone, then clearly* $f \circ f'$ *defined by* $(f \circ f')(x_1,\ldots,x_{k+k'-1}) = f(f'(x_1,\ldots,x_{k'}), x_{k'+1},\ldots,x_{k'+k-1})$ *is monotone as well.*

It can be shown that clones can be organized in a lattice, just as co-clones. These two lattices are in Galois correspondence through the notion of *polymorphism*. For a k-ary Boolean function $f : \{0,1\}^k \to \{0,1\}$ and k tuples $t_1,\ldots,t_k$ of the same size n, write $f(t_1,\ldots,t_k)$ for the tuple obtained by componentwise application of f, that is, $f(t_1,\ldots,t_k)$ is the tuple t of size n defined by $t[i] = f(t_1[i],\ldots,t_k[i])$ for $i = 1,\ldots,n$.

Definition 2.10 (polymorphism) *Let* R *be a relation, and* $f : \{0,1\}^k \to \{0,1\}$ *be a* k*-ary Boolean function. Then* f *is said to be a* polymorphism *of* R *if for any sequence of* k *tuples* $t_1,\ldots,t_k \in R$ *(not necessarily all distinct), the tuple* $t = f(t_1,\ldots,t_k)$ *is in* R*. We also say that* f preserves R*. The set of all polymorphisms of* R *(of any arity) is written* $Pol(R)$*.*

Example 2.11 *Let* R *be the relation* $\{0011, 1010, 0010\}$ *and* f *be the binary function defined by* $f(x_1,x_2) = x_1 \wedge x_2$*. Then* f *is a polymorphism of* R *because for all tuples* $t \in R$*,* $f(t,t) = t$ *is in* R *(*f *is idempotent), and* $f(0011,1010) = f(1010,0011) = 0010 \in R$*,* $f(0011,0010) = f(0010,0011) = 0010 \in R$*, and* $f(1010,0010) = f(0010,1010) = 0010 \in R$ *all hold. Contrastingly,* f' *defined by* $f'(x_1,x_2) = x_1 \vee x_2$ *is not a polymorphism of* R*, because* $f'(0011,1010) = 1011$ *is not in* R*.*

The Galois correspondence between the lattice of co-clones and that of clones is then defined by the operations *Pol* and *Inv*, defined by $Pol(\Gamma) = \{f \mid \forall R \in \Gamma, f \in Pol(R)\}$ and $Inv(C_f) = \{R \mid \forall f \in C_f, f \in Pol(R)\}$. In words, each language Γ is in correspondence with the clone of all functions which preserve all its relations, and conversely, each clone C_f is in correspondence with the co-clone of all relations which all its functions preserve.

This correspondence has many applications in complexity-theoretic studies of constraint satisfaction problems. A great number of them are surveyed in a recent collection of articles (Creignou

et al., 2008a). For our matters however, what is important is that the lattice of Boolean clones has been well-known for a long time (Post, 1941), and that it is surprisingly simple (at least, as opposed to that for any finite domain of size at least 3, which contains an uncountable number of clones). It follows directly that the lattice of Boolean co-clones is also known and simple. This lattice is depicted on Figure 2.1, and is known as *Post's lattice* (of *co*-clones). The solid lines depict inclusion relations (the bottommost co-clone being included in the topmost one), and the dashed lines depict a countable number of co-clones of the form, e.g., $\mathrm{IS}_0^2, \mathrm{IS}_0^3, \mathrm{IS}_0^4, \ldots$ for the rightmost one. We will not go into the details of the co-clones, but simply give some examples of them.

Example 2.12 *At the top of the lattice,* BR *is the co-clone containing all Boolean relations. Now* IN *is the co-clone containing all complementive relations, i.e., the relations R such that $(t[1], \ldots, t[k]) \in R$ entails $(1 - t[1], \ldots, 1 - t[k]) \in R$ (which is equivalent to the relation being preserved by the unary function $\neg$). The co-clone of all Horn relations is* IE_2*, where a relation R is defined to be Horn if it is preserved by the binary function $\wedge$ or, equivalently, if $R(X)$ is equivalent to a CNF φ over X with at most one positive literal per clause.*

How clones can be used for classifying the complexity of problems is illustrated by the following elegant proof of Schaefer's result (Schaefer, 1978). For any finite language Γ, let $\textsc{sat}(\Gamma)$ be the problem of deciding whether a given Γ-formula is satisfiable, that is, the satisfiability problem with inputs restricted to Γ-formulas (with some convenient presentation, e.g., with relations given in extension).

A formula is *Horn* (resp. *reverse Horn*, *bijunctive*) if it is written in CNF with at most one positive literal per clause (resp. at most one negative literal, at most two literals), and it is *affine* if it is written as a conjunction of linear equations over $\mathbb{F}_2$, of the form $(x_1 \oplus \cdots \oplus x_k = v)$. A language Γ is Horn if for all relations $R \in \Gamma$, there is a Horn CNF φ such that $R(X) \equiv \varphi$, and similarly for the other definitions. We studied in details how to translate efficiently from relations given in extension to such formulas, and how to decide whether there are such formulas at all for a given relation (Zanuttini and Hébrard, 2002; Hébrard and Zanuttini, 2003). Finally, a language is 0-*valid* (resp. 1-*valid*) if all its relations contain the all-0 tuple (resp. the all-1 tuple).

Proposition 2.13 (Schaefer 1978) *Let Γ be a finite Boolean language. Then* $\textsc{sat}(\Gamma)$ *is solvable in polynomial time if Γ is Horn, reverse Horn, bijunctive, affine,* 0*-valid, or* 1*-valid, and* NP*-complete in all other cases.*

The short proof goes as follows (Böhler et al., 2004). The problem is trivial for 0-valid languages (any instance is satisfied by the all-0 assignment $\mu_{\vec{0}}$) and for 1-valid languages. Now the case when Γ is Horn is a special case of the satisfiability problem for Horn CNF formulas (any constraint can be cast into such a formula using a lookup table, since Γ is finite and fixed). Since the latter is known to be solvable in polynomial-time, we have the result. The result for reverse Horn formulas is dual. Similarly, if Γ is bijunctive, then we have a special case of the satisfiability problem for 2CNF formulas, and if it is affine then we can decide satisfiability using Gaussian elimination. Now (Figure 2.1) the co-clone of all Horn (resp. reverse Horn, bijunctive, affine, 0-valid, 1-valid) languages is IE_2 (resp. IV_2, ID_2, IL_2, II_0, II_1). It follows that for any finite language Γ in one of these co-clones, $\textsc{sat}(\Gamma)$ is solvable in polynomial time. Because the lattice is ordered by inclusion, this is obviously also the case for any co-clone below one of these in the lattice.

Hence we are left with IN_2 and BR. But it turns out that the language $\{\{0,1\}^3 \setminus \{000, 111\}\}$ (containing only one relation) is in the co-clone IN_2, and allows to encode NOT-ALL-EQUAL-3-SAT, a well-known NP-complete problem. It follows that $\textsc{sat}(\Gamma)$ is NP-complete for all finite languages Γ which determine IN_2 ($\langle\Gamma\rangle = \mathrm{IN}_2$) or some co-clone above (BR), hence the classification is complete.

2.3 Conditional Preference Networks

We now present our language of choice for representing a user's preferences over a set of states or objects (assignments to variables), namely, conditional preference networks (CP-nets, Boutilier et al. 2004).

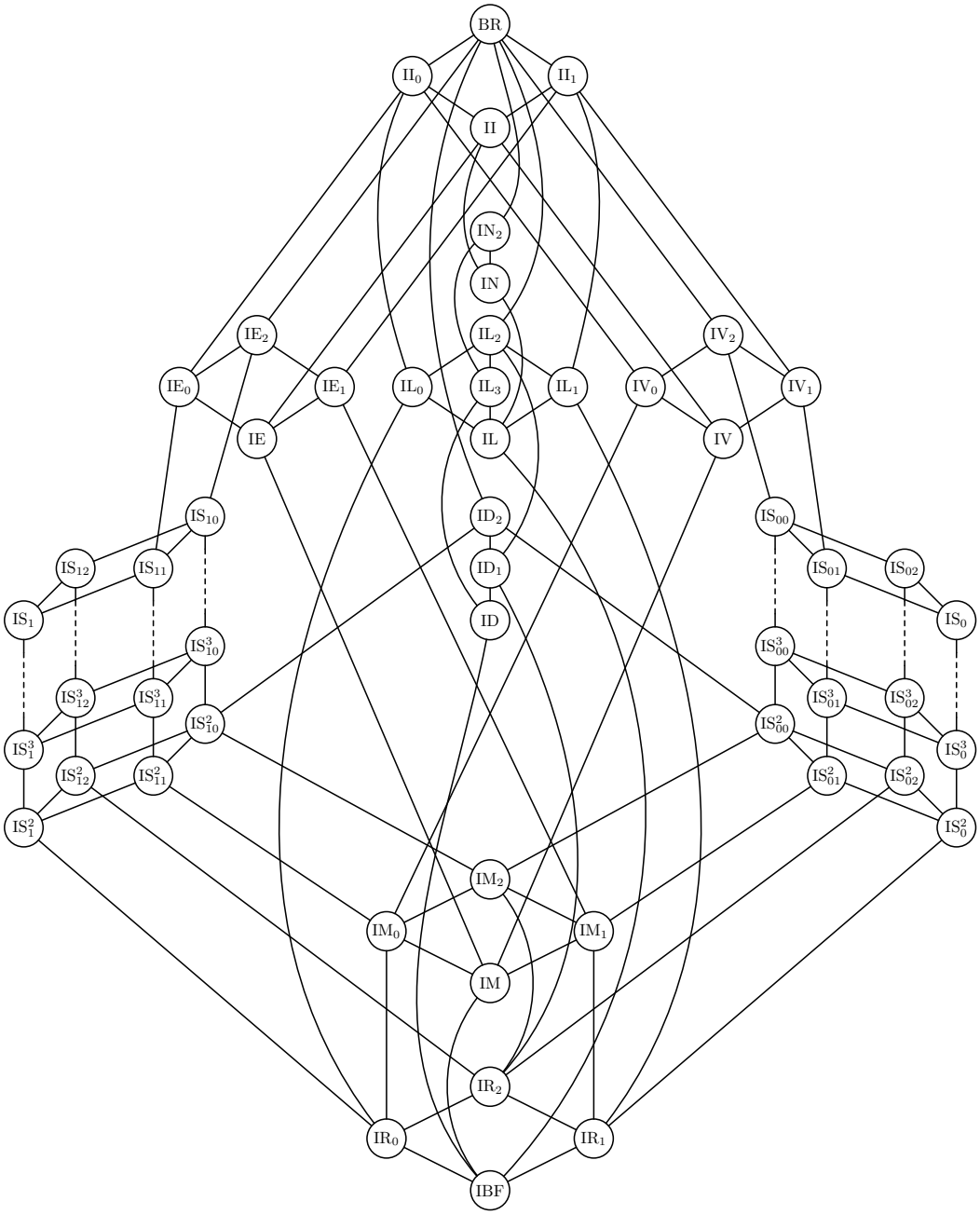
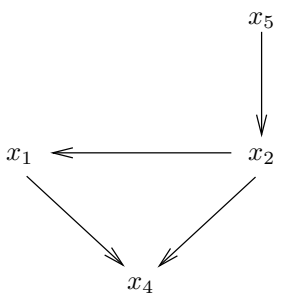


Figure 2.1: Post's lattice of co-clones



$x_2 : x_1 > \overline{x}_1$
$\overline{x}_2 : \overline{x}_1 > x_1$

$x_5 : x_2 > \overline{x}_2$
$\overline{x}_5 : \overline{x}_2 > x_2$

$x_1 \wedge \overline{x}_2 : \overline{x}_4 > x_4$
$\overline{x}_1 \wedge \overline{x}_2 : x_4 > \overline{x}_4$
$x_1 \wedge x_2 : \overline{x}_4 > x_4$

$x_5 > \overline{x}_5$

Figure 2.2: An example CP-net

2.3.1 Syntax

A *conditional preference rule* (CP-rule) on a variable x is an expression of the form $t : \ell > \overline{\ell}$, where ℓ is a literal on the variable x and t is a term not mentioning x. Such a rule captures the statement "if t is true of the objects under consideration, the value ℓ is preferred to the value $\overline{\ell}$ for variable x, provided all other things are equal".

A *conditional preference table* (CP-table) on a variable x with respect to a set of *parents* $Pa(x)$ for x (with $x \notin Pa(x)$) is a set of CP-rules on x which associates at most one rule of the form $t : \ell > \overline{\ell}$ to each term t with $Vars(t) = Pa(x)$. The CP-table is said to be *complete* if *exactly* one such rule is associated to each term t over $Pa(x)$, and *incomplete* otherwise.

Example 2.14 *An example CP-rule is $x_1 \wedge \overline{x}_2 : \overline{x}_4 > x_4$, meaning that value 0 is preferred to value 1 for x_4 in objects satisfying $x_1 \wedge \overline{x}_2$ (and all other things being equal). With $Pa(x_4) = \{x_1, x_2\}$, an example CP-table on x_4 is $\{x_1 \wedge \overline{x}_2 : \overline{x}_4 > x_4, \overline{x}_1 \wedge \overline{x}_2 : x_4 > \overline{x}_4, x_1 \wedge x_2 : \overline{x}_4 > x_4\}$. This table is incomplete because there is no rule for the term $\overline{x}_1 \wedge x_2$ on $Pa(x_2)$.*

A *conditional preference network* (CP-net) over a set of variables X is merely a set of CP-tables, at most one on each variable $x \in X$. Formally, it is given by a directed graph G whose vertice set is a subset X' of X, and a CP-table on each $x \in X'$ with respect to its set of parents $Pa(x)$ in the graph G.

The variables in X' are called *relevant variables*, which distinguishes them among all the variables on which the objects to be compared are described, a lot of which may be irrelevant to the relation encoded by the CP-net. The CP-net is said to be *complete* if $X' = X$ holds, that is, all variables are relevant, and all its CP-tables are complete. Otherwise, it is said to be *incomplete*, which encompasses cases when some variables are irrelevant or when the preference on the values of some variables are not given for all contexts, as well as combinations of these.

Example 2.15 (continued) *Let $X = \{x_1, \ldots, x_5\}$. Figure 2.2 depicts a CP-net N on X. The relevant variables in N are $X' = \{x_1, x_2, x_4, x_5\}$. The tables on x_1, x_2, x_5 are complete.*

We will be interested in two syntactic restrictions of CP-nets. A CP-net is said to be *acyclic* if its graph G is (directed) acyclic, and *tree-structured* if its graph G is a forest, in particular, each relevant variable $x \in X'$ has at most one parent. For instance, the CP-net N on Figure 2.2 is acyclic, but not tree-structured (x_4 has two parents).

2.3.2 Semantics

Call two assignments μ, μ' *swaps* (of each other, on variable x) if they assign the same value to all variables but x. The semantics of a CP-net N over X is defined starting with the following relation on swaps.

Let μ, μ' be swaps on some variable x. Then μ is said to be *preferred* to μ' according to N, written $\mu \succ_N \mu'$, if there is a CP-rule $t : \ell > \overline{\ell}$ on x such that μ, μ' satisfy t, μ satisfies ℓ, and μ' satisfies $\overline{\ell}$. Note that because μ, μ' are swaps on x, either both or none satisfy t, and they coincide on the variables in $X \setminus (Pa(x) \cup \{x\})$. This translates the "all other things being equal" interpretation of CP-rules.

The relation $\succ_N$ induced by a CP-net N on the set of assignments 2^X to X is simply defined to be the transitive closure of the relation which it induces on swaps[1].

Example 2.16 (continued) *The assignments* $10001, 10011$ *are swaps on* x_4*. Writing* N *for the CP-net of Figure 2.2, we have* $10001 \succ_N 10011$*, due to the rule* $x_1 \wedge \overline{x}_2 : \overline{x}_4 > x_4$*. On the other hand, the assignments* $10001, 10010$ *are not comparable directly by any rule, because they are not swaps (hence they cannot satisfy the "all other things being equal" property). Still, we have* $10001 \succ_N 10010$*, as witnessed by* $10001 \succ_N 10011 \succ_N 10010$ *(using the rules* $x_1 \wedge \overline{x}_2 : \overline{x}_4 > x_4$ *and* $x_5 > \overline{x}_5$*). Finally, it can be checked that the assignments* 10001 *and* 00000 *are incomparable to each other, and so are* 00000 *and* 11111*, because they differ on the irrelevant variable* x_3*.*

2.3.3 Preference Relations

A *preference relation* $\succ$ on a set of assignments 2^X is defined to be a transitive and irreflexive binary relation on 2^X. Intuitively, this corresponds to assuming that if the user prefers μ to μ' and μ' to μ'', then she prefers μ to μ'' (transitivity), and that she never (strictly) prefers μ to itself (irreflexivity). On the other hand, we do *not* assume that any two assignments μ, μ' are comparable to each other, that is, it may be the case that the user does not prefer μ to μ', nor μ' to μ.

There may be at least two reasons why two assignments would not be comparable. The first case is when the user really does not want to compare them, as is the case for dilemmas ("which of your children do you want to sacrifice?"). The second case is when the user indeed prefers one to another, but this preference is not known to the agent. This may be because it has not learnt it, or because there are some differences between the assignments which the agent does not perceive (for instance, they differ on variables which the agent believes to be irrelevant).

An important point to note is that incomparability of μ, μ' is not the same as *indifference* to μ, μ', which would translate into μ being "preferred or equal" to μ', and vice-versa. Encoding indifference in (satisfiable) CP-nets requires to extend the syntax to include atomic statements of the form $t : \ell \sim \overline{\ell}$. We do not pursue this direction in this thesis, but note that results on CP-nets *without* indifference cannot always be straightforwardly extended to results on CP-nets *with* indifference.

As it turns out, not all CP-nets encode a preference relation. The point is that though the relation on swaps is guaranteed to be irreflexive by construction, its transitive closure needs not be so. However, a sufficient condition for the relation induced by a CP-net to be a preference relation, that is, to be transitive and irreflexive, is for the CP-net to be acyclic (Boutilier et al., 2004, Theorem 1). If the relation is indeed a preference relation, then the CP-net is said to be *satisfiable*, or *consistent*.

2.4 Representations of Actions

The last representational framework which we will use in this thesis is that of factored representations of actions, especially in the setting of Markov Decision Processes.

[1] Using the same symbol for the relation on swaps and its transitive closure is well-founded for acyclic CP-nets (Koriche and Zanuttini, 2010, Lemma 1), but not in the general case. For instance, for $N = \{\overline{x}_1 : x_2 > \overline{x}_2, x_2 : x_1 > \overline{x}_1, \overline{x}_2 : \overline{x}_1 > x_1\}$ we have $11 \succ_N 10$ only by transitivity. However, the confusion will be harmless for all the work presented in this document.

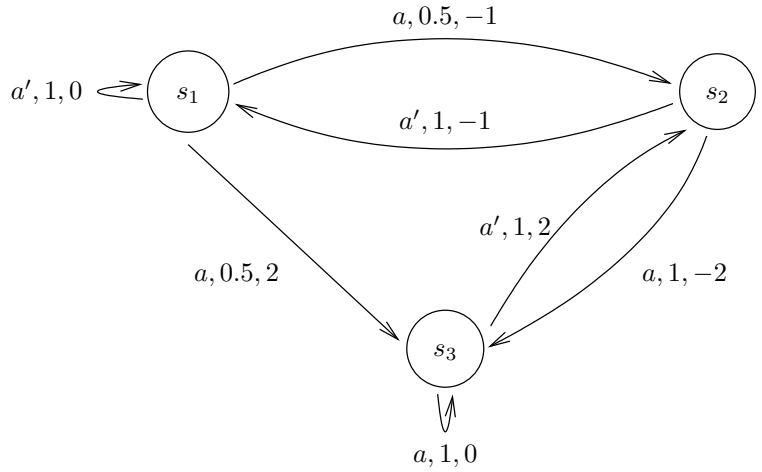


Figure 2.3: An example Markov Decision Process

2.4.1 Markov Decision Processes

A *Markov Decision Process* is a mathematical abstraction of stochastic processes which evolve depending on their current state only (Markov assumption), rather than depending on their complete history or even on a portion of it. Formally, an MDP is defined to be a tuple (S, A, T, R), in which S is a set of states, A is a set of actions, $T : S \times A \times S \to [0, 1]$ gives the probability $T(s, a, s')$ for the process to evolve to state s' given that action a was performed in state s, and $R : S \times A \times S \to \mathbb{R}$ assigns a *reward* (or a *cost*) to each such *transition*.

The transition function T is required to satisfy $\sum_{s' \in S} T(s, a, s') = 1$ for all states $s \in S$ and actions $a \in A$. We view it as a probability distribution on the *next-states* s' of state s through action a. We write $T(\cdot|s, a)$ for this distribution and, accordingly, $T(s'|s, a)$ for $T(s, a, s')$.

Example 2.17 *Figure 2.3 depicts a Markov Decision Process, where the transitions between states (arrows from s to s') are labelled with triples giving the action a, probability of transition $T(s'|s, a)$, and reward $R(s, a, s')$. For instance, the next-states of s_1 through a are s_2, s_3. Action a' provokes a deterministic transition from s_3 to s_2, and this transition gives a reward of 2 to the agent.*

MDPs provide a framework of choice for modelling *sequential decision problems*, in which an agent has to make decisions about which action a to take in states s of the environment. Such decisions influence the next-state s' and hence, the next decision to make.

The decisions made by an agent facing such a process are formalized by a *policy* $\pi : S \to A$, which prescribes an action $a = \pi(s)$ for each possible state s of the environment. Precisely, this defines *stationary* and *deterministic* policies, that is, policies which prescribe an action depending on the current state only, and which always prescribe the same action given a state s (as opposed to *randomized policies*, which may choose actions at random).

The typical goal for an agent is to compute an *optimal policy* π^*, that is, one which is better than any other policy under some given criterion. Typical criteria consist in maximizing the expectation of the sum of rewards obtained over some finite or infinite horizon, the average of such rewards in the limit, etc. We give more details about this in Chapter 3, when defining the computational problems associated to MDPs. Note that under some conditions on the process and on the criterion used for defining optimal policies, there always exists an optimal policy which is both stationary and deterministic (Puterman, 1994).

Example 2.18 (continued) *Consider the policy π defined by $\pi(s_1) = a$ and $\pi(s_2) = \pi(s_3) = a'$ for the MDP depicted on Figure 2.3. The expected sum of rewards for π in s_1, at horizon 3, is* $Q^3_\pi(s_1) = 0.5 \times (-1 + -1 + 0.5 \times (-1) + 0.5 \times 2) + 0.5 \times (2 + 2 + (-1)) = 0.75$.

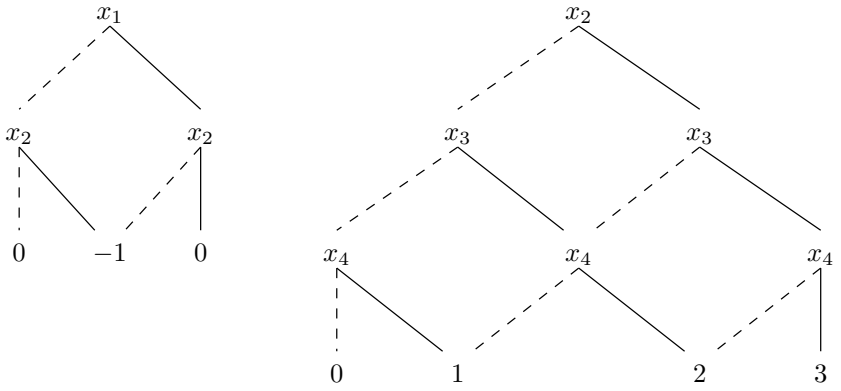


Figure 2.4: An example factored representation of a reward function

The motivation for computing policies (as opposed to computing the best decision at each timestep) is twofold. The first motivation is computational, because computing the best decision at each timestep would typically make the agent continuously reevaluate large parts of the search space. Hence it is often valuable to compute an optimal policy (for all states) once and for all, even if not all the states are encountered at execution time.

The second motivation is in the design of *reactive agents*. Typically, a policy π has a representation which allows the agent to make its decision for the current state s, that is, compute $a = \pi(s)$, in time linear in the size of s (number of variables over which s is defined). This means that once a policy has been computed, the agent is able to react to changes in its environment as fast as conceivable. This is of course to be balanced with the time required for computing the policy off-line.

2.4.2 Factored Representations

Since we are interested in propositional representations of the environment, the states s in the MDPs of interest will be assignments to some set of variables X, that is, the set of states S in an MDP will be 2^X. This *factored representation* of states calls for a factored representation of actions and rewards as well.

We will not go into many details about factored representations of rewards, but the basic idea is that rewards typically depend on a few descriptors of the environment, and will usually be additive. Hence a typical factored representation of the reward function R for an MDP over the set of variables X is as a set of functions $R_1, \ldots, R_k$, each of which depends on a few variables in X, say, R_i depends on $X_i \subseteq X$, in such a way that $R(s) = \sum_{i=1}^{k} R_i(s_{|X_i})$ holds for all states s. Then the functions R_i, $i = 1, \ldots, k$, are typically given in a factored form, for example as an algebraic decision diagram over X_i.

Example 2.19 *Assume that the reward function $s, a, s' \mapsto R(s, a, s')$ of some MDP depends on the next-state s' only, and is the sum of -1 if x_1, x_2 have different values in s', plus 1 per variable true in s' among x_2, x_3, x_4. Then R can be represented as the sum of the two functions depicted as algebraic decision diagrams on Figure 2.4 (where solid lines point to the "then" child, and dashed lines to the "else" child). The size of this representation is to be compared with the 2^n entries needed for representing R in tabular form if states are defined over n variables.*

2.4.3 Probabilistic STRIPS Operators

The factored representations of actions will be essential to our study. The first of these is by *Probabilistic STRIPS Operators* (PSOs, Kushmerick et al. 1995). PSOs are the probabilistic counterpart of STRIPS, which we introduce here in the grounded (propositional) setting.

At the heart of the definition of PSOs is that of an *effect*. An effect is simply a (consistent) term over a subset of X, usually written $e, e_1, e_i \ldots$. To *apply* an effect e to some state s means determining the next-state s', written $apply(s, e)$, and defined by

$$apply(s,e)[x] = \begin{cases} 1 & \text{if } e \models x \\ 0 & \text{if } e \models \overline{x} \\ s[x] & \text{otherwise} \end{cases}$$

Abusing notation by treating assignments as sets of literals, we may also write $apply(s, e) = (s \setminus \{\overline{\ell} \mid \ell \in e\}) \cup e$. The *empty effect* is written $\emptyset$, and satisfies $apply(s, \emptyset) = s$ for all states s.

Example 2.20 *The term $e = x_2 \wedge \overline{x}_3 \wedge \overline{x}_4$ is an effect. When applied in state $s = 10011$ it yields the next-state $apply(s, e) = 11001$. Observe that also $apply(11111, e) = 11001$, and $apply(s, x_2 \wedge \overline{x}_4 \wedge x_5) = 11001$.*

Given this, an action a is given as a PSO if it is written as

$$a = \begin{cases} c_1 & : & (e_{11}, p_{11}), \ldots, (e_{1\ell_1}, p_{1\ell_1}) \\ & \ldots & \\ c_k & : & (e_{k1}, p_{k1}), \ldots, (e_{k\ell_k}, p_{k\ell_k}) \end{cases}$$

where $c_1, \ldots, c_k$ are propositional formulas over X which partition the state space, in the sense that $M(c_1 \vee \cdots \vee c_k)$ is 2^X and $c_i \wedge c_j$ is unsatisfiable for all $i \neq j$, and where for each condition c_i and index $j = 1, \ldots, \ell_i$, e_{ij} is an effect and $p_{ij} \in [0, 1]$ is a probability, such that $\sum_j p_{ij} = 1$.

The semantics of an action so described is given by the transition function which it induces:

$$T(s'|s, a) = \sum \{p_{ij} \mid s \models c_i \text{ and } apply(s, e_{ij}) = s'\}$$

In words, the probability of moving to state s' when taking action a in state s is the sum of the probabilities of the effects, in the (unique) condition satisfied by s, which transform s to s'.

Example 2.21 *An example PSO is the following:*

$$a = \begin{cases} x_1 \wedge x_2 \wedge x_3 & : & (\overline{x}_3 \wedge x_4 \wedge x_5, 0.28), (x_4 \wedge x_5, 0.12), (\overline{x}_3 \wedge \overline{x}_4 \wedge \overline{x}_5, 0.42), (\overline{x}_4 \wedge \overline{x}_5, 0.18) \\ x_1 \wedge x_2 \wedge \overline{x}_3 & : & (\overline{x}_2, 1) \\ \overline{x}_1 \vee \overline{x}_2 & : & (\overline{x}_3, 0.7), (\emptyset, 0.3) \end{cases}$$

The transition probability from state $s = 00000$ to itself is 1, because the third condition applies, and both effects provoke this transition. On the other hand, the transition probability from 11111 to 11000 is 0.42 (third effect in first condition).

2.4.4 PPDDL

The *Probabilistic Planning Domain Description Language* (PPDDL, Younes and Littman 2004) builds on PDDL by adding a stochastic component. Though PPDDL uses first-order descriptions, we are interested here in its *grounded* version. The grounded version of a PPDDL description (for a given set of objects, that is, constants) is obtained by making a (propositional) action out of each possible combination of constants for the arguments of a (first-order) action, and similarly making a propositional variable out of each possible instantiation of a predicate.

PPDDL generalizes PSOs by allowing richer constructs. The syntax for an action (as obtained when restricting to grounded descriptions) is defined recursively by the following constructs. Abusing words, we do not distinguish between actions and their effects as described in PPDDL, and refer to both as *PPDDL effects*:

- if x is a variable, then x and (`not` x) are PPDDL effects,
- for $r \in \mathbb{R}$, (`increase` (`reward`) r) and (`decrease` (`reward`) r) are PPDDL effects,

- for c a formula over X and e a PPDDL effect, (`when` c e) is a PPDDL effect,
- for $p_1, \ldots, p_\ell \in [0,1]$ with $\sum_{i=1}^{\ell} p_i \leq 1$, and $e_1, \ldots, e_\ell$ PPDDL effects, (`prob` p_1 e_1 $\ldots$ p_ℓ e_ℓ) is a PPDDL effect,
- for $e_1, \ldots, e_\ell$ PPDDL effects, (`and` $e_1 \ldots e_\ell$) is a PPDDL effect.

Moreover, the following condition is required for conjunctive effects, which denote a *simultaneous* application of all their arguments. The construct (`and` $e_1 \ldots e_\ell$) is allowed only if $e_1, \ldots, e_\ell$ are such that, for all combinations of conjunctions $t_1, \ldots, t_\ell$ of simple effects, one per PPDDL effect e_i, which may occur in some state s with nonzero probability according to e_i, the terms $t_1, \ldots, t_\ell$ are mutually consistent, that is, no variable appears negated in one and unnegated in some other. This condition is necessary for the application of *all* effects $e_1, \ldots, e_\ell$ to be well-defined.

The syntax of PPDDL is best given by the distribution $D(e, s)$ of PSO effects (terms) which a PPDDL effect e induces on a state s. This distribution is defined recursively as follows, where (e_i, p_i) represents an effect e_i with probability p_i:

$$
\begin{aligned}
D(x, s) &= (x, 1) \\
D(\overline{x}, s) &= (\overline{x}, 1) \\
D((\texttt{increase}\ (\texttt{reward})\ r), s) &= (\emptyset, 1) \\
D((\texttt{decrease}\ (\texttt{reward})\ r), s) &= (\emptyset, 1) \\
D((\texttt{when}\ c\ e), s) &= D(e, s) \text{ if } s \models c \\
D((\texttt{when}\ c\ e), s) &= (\emptyset, 1) \text{ if } s \not\models c \\
D((\texttt{prob}\ p_1\ e_1\ \ldots\ p_\ell\ e_\ell), s) &= \bigcup_{i=1}^{\ell} \{(e_{ij}, p_{ij} p_i) \mid (e_{ij}, p_{ij}) \in D(e_i, s)\} \cup \{(\emptyset, 1 - p_1 - \cdots - p_\ell)\} \\
D((\texttt{and}\ e_1 \ldots e_\ell), s) &= \{(e_{1j} \cup e, p_{1j} p) \mid (e_{1j}, p_{1j}) \in D(e_1, s), (e, p) \in D((\texttt{and}\ e_2 \ldots e_\ell), s)\}
\end{aligned}
$$

Observe that if the probabilities do not sum up to 1 in a `prob` construct, the empty effect is implicitly assumed to occur with the remaining probability mass.

The semantics of this transformation is given by the transition function induced by an action a given as a PPDDL effect:

$$T(s'|s, a) = \sum \{p \mid (e, p) \in D(e, s) \text{ and } \mathit{apply}(s, e) = s'\}$$

This construction can be extended to include the rewards associated to transitions by the constructs (`increase` (`reward`) r) and (`decrease` (`reward`) r).

Example 2.22 (continued) *An example PPDDL action is the one given by the PPDDL effect*

$$
\begin{array}{llll}
(\texttt{and} & (\texttt{when} & x_1 \wedge x_2 & \\
 & & (\texttt{and} & (\texttt{when}\ x_3\ (\texttt{prob}\ 0.4\ (\texttt{and}\ x_4\ x_5)\ 0.6\ (\texttt{and}\ \overline{x}_4\ \overline{x}_5))) \\
 & & & (\texttt{when}\ \overline{x}_3\ \overline{x}_2))) \\
 & (\texttt{prob}\ 0.7\ \overline{x}_3)) &&
\end{array}
$$

It can be seen that this action is equivalent to the PSO in Example 2.21, in the sense that the induced transition function is the same. Also note that the topmost `and` *could not be extended with an effect e mentioning x_4, because $\overline{x}_4$ is a possibility of the first effect and so, this would violate the consistency assumption (x_4 should be under a condition inconsistent with $x_1 \wedge x_2 \wedge x_3$).*

2.4.5 Dynamic Bayesian Networks

The last representation of actions which we will use is the most popular one for factored MDPs, namely the one by *Dynamic Bayesian Networks* (DBNs), as proposed by Boutilier et al. (1999). In

this representation, the effects of an action on the value of each (next-state) variable are represented independently from each other.

As is common, we distinguish the values of variables in the current state s and in the next state s' by introducing a primed version x' of each variable x, meant to represent the value of this variable in the next state. We write X' for the set of all primed variables. Now given an action a and a variable x, the DBN representation associates a set of *parents* $Pa(x') \subseteq X \cup X'$ to the next-state variable x'. The binary relation $\rightarrow$ defined by $x_p \rightarrow x' \iff x_p \in Pa(x')$ (resp. $x'_p \rightarrow x' \iff x'_p \in Pa(x')$) is required to be (directed) acyclic.

Now for each variable x, a *conditional probability table* is associated to x', which for each term t with variables $Vars(t) = Pa(x')$ gives the probability $p_{x|t}$ that a makes x true (or leaves x true) when the starting and next states s, s' together satisfy t. Implicitly, $1 - p_{x|t}$ gives the probability that a makes x false under the same conditions. Typically, the conditional probability tables are not represented in this extensive form, but rather in some compact structure, such as an algebraic decision diagram.

The semantics of this representation is again given by the induced transition function, where ℓ' enumerates all primed literals satisfied by s', and t may contain primed and/or unprimed variables:

$$T(s'|s,a) = \prod \{p_{\ell'|t} \mid \ell' \in \{x', \overline{x}'\} \text{ and } x \in X \text{ and } s' \models \ell' \text{ and } Vars(t) = Pa(x') \text{ and } s.s' \models t\}$$

This semantics is justified by the well-known *chain rule* for random variables $X_1, \ldots, X_n$:

$$Pr(X_1 \ldots X_n) = Pr(X_1|X_2 \ldots X_n) \times Pr(X_2|X_3 \ldots X_n) \times \cdots \times Pr(X_n)$$

Finally, a relationship $x'_p \in Pa(x')$ (dependence of some variable on another *primed* variable) is called a *synchronic arc*. It is easily seen that if the DBNs representing action a have no synchronic arcs, then the effects of a on each variable x' are probabilistically independent from each other, that is,

$$T(s'|s,a) = \prod_{s \models \ell'} T(\ell'|s,a)$$

holds, where $T(\ell'|s,a)$ denotes the probability $p_{\ell'|t}$ for the term t satisfied by s. It is easily seen that not all transition functions can be represented in this manner. We will come back to this issue in Chapters 5 and 6, when discussing algorithms which operate on DBN representations of actions.

Example 2.23 (continued) *Figure 2.5 gives a representation by DBNs of the action a defined in Example 2.21. For instance, the leftmost branch of the diagram for x'_2 reads "if x_1 and x_2 are false in s, then in state s' x_2 will be false". The rightmost branch of the diagram for x'_4 reads "if x_1, x_2, x_3 are all true in s, then there is a 0.4 probability that x_4 will be true in s' (and a $1 - 0.4 = 0.6$ probability that it will be false)". We have for example*

$$\begin{aligned} & T(11000|11111, a) \\ = \; & p_{\overline{x}'_5|x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge \overline{x}'_4} \times p_{\overline{x}'_4|x_1 \wedge x_2 \wedge x_3 \wedge x_4} \times p_{\overline{x}'_3|x_1 \wedge x_2 \wedge x_3} \times p_{x'_2|x_1 \wedge x_2 \wedge x_3} \times p_{x'_1|x_1} \\ = \; & (1-0) \times (1-0.4) \times (1-0.3) \times 1 \times 1 \\ = \; & 0.42 \end{aligned}$$

Observe that a synchronic arc between x'_4 to x'_5 (in either direction) is necessary because these variables are not probabilistically independent (both are set to true, or both to false, in the first condition as written in Example 2.21). Contrastingly, though x'_4 occurs with x'_3 in some effects, they are independent from each other, as can be seen on the PPDDL description of a in Example 2.22.

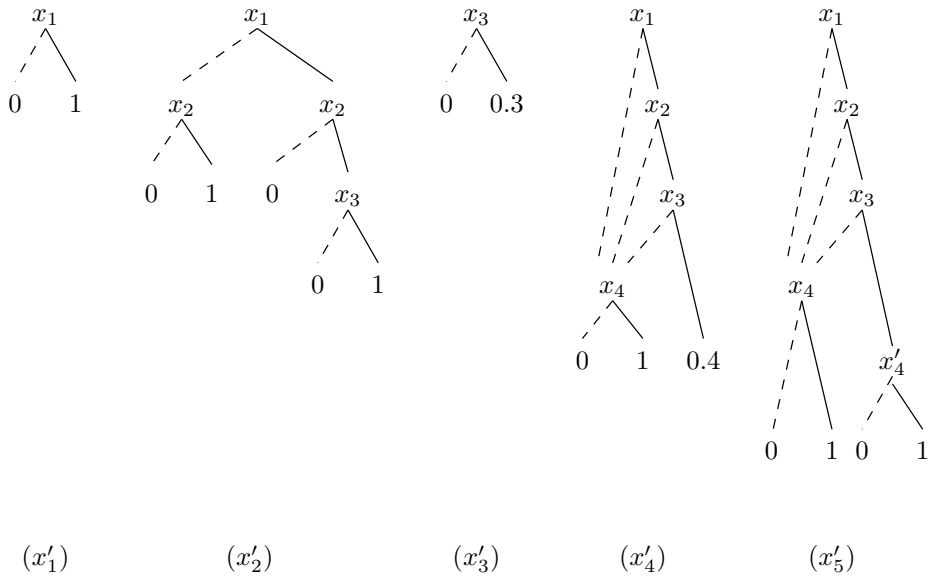


Figure 2.5: Example DBN representation for an action

Chapter 3

Preliminaries on Computational Problems

Contents

This chapter formally defines the main computational problems which we will encounter in the forthcoming chapters, namely reasoning problems (Section 3.1), problems in concept learning (Section 3.2), and problems related to decision making and reinforcement learning (Section 3.3). Each of them is formally defined through the inputs given to an algorithm and the output expected from it, as well as the elementary operations available to it, when relevant. Observe that we will not study much the reasoning problems (deduction and abduction) in the rest of the document, but their presentation is intended to make the setting clear and complete.

3.1 Reasoning

For a general introduction to the generic reasoning processes, let us borrow and adapt from Shanahan (1999) the diagram depicted on Figure 3.1. The central box on the figure stands for a *consequence relation*, which defines what formulas logically follow from other formulas. For the moment,

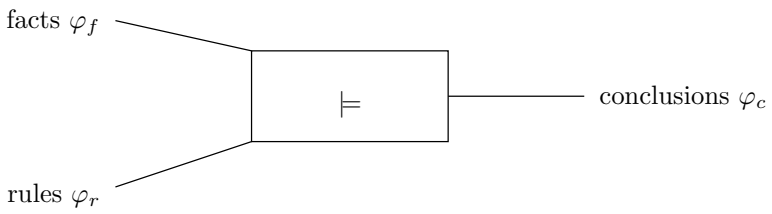


Figure 3.1: The three generic reasoning problems (adapted from Shanahan 1999)

this consequence relation will be classical logical entailment ($\varphi \models \varphi'$ holds if the models of φ satisfy φ').

According to the vision on Figure 3.1, reasoning involves three components: rules which govern the environment, known facts (e.g., what is known about the current state of the environment), and conclusions which can be drawn from these facts using the rules. In this section, all three components will be propositional formulas (or Γ-formulas for some language Γ). Typically, facts are literals (e.g., the current output of reliable sensors), but we take a more general acception. Then each basic reasoning process can be defined as the one of computing one of these three components given the two others.

3.1.1 Deduction

The basic reasoning problem is *deduction*, which corresponds to computing (or verifying) conclusions given facts and rules, that is, reading Figure 3.1 from left to right.

Definition 3.1 (deduction problem) *We call* DEDUCTION *the decision problem given by*

- *input: propositional formulas $\varphi_f, \varphi_r, \varphi_c$,*
- *question: does $\varphi_f \wedge \varphi_r \models \varphi_c$ hold?*

We write DEDUCTION(Γ) *for the problem* DEDUCTION *with φ_f, φ_r restricted to Γ-formulas.*

The typical role of deduction is to infer, or verify, that a given formula of interest holds in the current state, whereas its truth value is not directly observable. For instance, assume that a software agent has the intention to ask the user a question, a precondition of which action is that the user has an e-mail client open (wherever she is). Then it is a deductive problem to decide whether this precondition is met, given, say, that it is 08:00 AM on a weekday and that the user was at home at 07:55 (facts), and that the user takes her car for 30 minutes for going to work, that she must be at work at 08:15 AM, and that there can be no e-mail client running in a car (rules)[1].

As concerns complexity, it is well-known that deciding $\varphi \models \varphi'$ amounts to deciding whether the formula $\varphi \wedge \neg\varphi'$ is unsatisfiable. Hence DEDUCTION is coNP-complete in general. It is also known to be in P for Horn and bijunctive CNF formulas, if φ' is retricted to be a clause.

We will also sometimes refer to the problem of inference by *circumscription*, which is a deduction problem but with the consequence relation changed (the box on Figure 3.1). Precisely, φ entails φ' by circumscription if every *minimal* model of φ satisfies φ', where "minimal" refers to the componentwise order $\mu \leq \mu' \iff \forall x \in X, \mu[x] \leq \mu'[x]$ or to the order induced by a partition (P, Q, Z) of the variables. We refer the reader to McCarthy's articles (McCarthy, 1980, 1986) for explanations about this relation, but the basic idea is to make the assumption that certain variables default to some value, e.g., variables describing abnormality (of the assignment at hand) default to the value 0 ("not abnormal"). The complexity of circumscriptive deduction is also well-known (Nordh, 2004; Durand et al., 2009).

3.1.2 Abduction

The second generic reasoning process is *abduction*, which in the main lines corresponds to reasoning from effects to their (plausible) causes. It corresponds to reading Figure 3.1 from the bottom-right to the top-left corner, that is, rules and conclusions are given, and facts must be retrieved.

Definition 3.2 (abduction problem) *We call* ABDUCTION *the decision problem given by*

- *input: propositional formulas φ_r, φ_c, and set of literals L,*
- *question: is there a term $t \subseteq L$ such that the formula $\varphi_r \wedge t$ is satisfiable and entails φ_c (that is, $\varphi_r \wedge t \models \varphi_c$)?*

[1] It can also be deduced from these facts and rules that the user will be late at work!

We write ABDUCTION(Γ) *for the problem* ABDUCTION *with* φ_r *restricted to* Γ*-formulas.*

The formalization as a decision problem mainly serves the purpose of complexity analyzes. In practice, we are confronted with the problem of *computing* an explanation t. For instance, assume that the software agent of Section 3.1.1 now asks the user a question at a moment when she is supposedly at work, say, at 08:30 AM, but gets no answer despite the rule stating that the user always has her e-mail client running at work. Then it is an abductive process to infer from this *observation* (the user does not answer, which plays the role of formula φ_c) and the rules φ_r known to the agent, that the user has possibly got stuck in a traffic jam (which becomes, e.g., a fact temporarily assumed to be true).

The set of literals L given as an input to the problem is called the set of *abducibles*. For instance, in a context of medical diagnosis, observations φ_c typically correspond to symptoms, and we want to explain these by diseases (not by other symptoms, even if some rules make this possible, e.g., "fever causes shivering"). Hence in this case, the set of abducibles L would be the set of all literals which encode diseases. In a context where abduction is used for planning, the set of abducibles is typically the set of actions.

Just like deduction, the complexity of abduction in propositional logic has been well-studied, in particular by Eiter and Gottlob (1995). We also gave a complete complexity picture for restrictions of φ_r to Γ-formulas (for all finite Γ) and for various syntactic retrictions on abducibles L and observations φ_c, in a series of articles (Nordh and Zanuttini, 2005; Creignou and Zanuttini, 2006; Nordh and Zanuttini, 2008). It turns out that abduction in general, as formalized above, is a Σ_2^{P}-complete problem. Its counting complexity (deciding how many preferred explanations the problem has, under some preference criterion) is also well-studied (Hermann and Pichler, 2010).

3.2 Induction and Concept Learning

The third way to read Figure 3.1 is from the top right corner to the bottom-left one, which corresponds to reasoning from facts and derived conclusions to the rules which govern the environment. This generic process is called *induction*, which refers to the process of *generalizing* some pairs (facts,conclusions) to rules which apply to yet unseen such pairs.

Like deduction and abduction, induction is a generic process. In this document we will be mostly interested in *concept learning*, that is, the rules to be induced are *classification rules* for some hidden concept c^*, able to tell whether given objects belong to the concept. Formally, a *Boolean concept* is simply a subset of the set 2^X of all assignments. Example concepts are the concept of a *goal state* in planning problems, the concept of the user being *busy* in human-agent interaction, the concept of all states where some given action is *applicable*, etc.

Since our study takes place in the setting of computational learning, from now on we will talk of *concept learning* to refer to induction of Boolean concepts (precisely, of classification rules for them).

3.2.1 Passive Learning

The most natural framework for studying concept learning is one where the agent receives a certain number of *labelled examples*, that is, a number of assignments $\mu_1, \mu_2, \ldots$ for each of which it is told whether it belongs to the target concept c^* (if this is the case, the example is said to be *positive*, otherwise it is *negative*). Such examples may be provided by an expert, who can recognize the instances of the target concept, or by experience, e.g., a robot may try to apply an action in a certain number of states to find out in which of these it is indeed applicable.

In all cases, the point is that the number of labelled examples which can be obtained is restricted: an expert typically cannot afford the time to label all $2^{|X|}$ examples, and a robot cannot try its actions in all possible states. Hence *generalization* must take place, from labelled to yet unseen examples.

This general learning framework is called *passive* learning, because the agent simply receives examples. Many precise formalizations have been proposed for these problems, of which we give a number here which will be useful for our study.

Definition 3.3 (PAC learning, Valiant 1984) *In* Probably Approximately Correct learning*, there is a fixed but unknown probability distribution D on the set of all assignments 2^X, and a target concept $c^* \subseteq 2^X$ hidden to the agent. The agent is given $\epsilon, \delta \in]0,1]$ and access to labelled examples sampled i.i.d. according to D, and must compute a concept $\hat{c}$ such that with confidence at least $1-\delta$, $Pr_D(c^* \Delta \hat{c})$ is at most ϵ (Δ denotes symmetric difference).*

Definition 3.4 (exact identification, Angluin 1988) *In* exact identification*, there is a target concept $c^* \subseteq 2^X$ hidden to the agent. The agent has access to equivalence queries, through which it proposes concepts $\hat{c}$. If the hypothesis $\hat{c}$ is equivalent to c^*, the agent is told so and the protocol terminates on success for the agent. Otherwise, the agent is given a counterexample $\mu \in \hat{c} \Delta c^*$, and the protocol continues. The counterexamples may be adversarial (in particular, they need not obey a fixed probability distribution).*

It turns out that exact identification is more demanding than PAC-learning, in the sense that an algorithm for exact identification can be turned to one for PAC-learning. The idea is simply to simulate each equivalence query by asking sufficiently many labelled examples (Angluin, 1988). It is even more natural to see positive results with equivalence queries through this transformation, rather than imagining the agent presenting its hypothesis to the user and asking whether it is correct.

It also turns out that learning in the *on-line* model (Littlestone, 1988) is exactly as demanding as exact identification. The on-line model formalizes situations where the agent is presented a possibly adversarial sequence of assignments, and must classify all of them as they come, with a bounded number of mistakes over any (infinite) sequence. The idea is that learning in this model requires the ability to learn from each mistake.

The last model which we introduce here has been proposed recently, with algorithms for model-based reinforcement learning as a motivation.

Definition 3.5 (KWIK learning, Li et al. 2011) *In the* Knows What It Knows *setting, there is a target concept $c^* \subseteq 2^X$ hidden to the agent. The agent receives a possibly adversarial sequence of assignments, and for each of them must either classify it correctly or admit that it cannot ("don't know" answer, written "?").*

3.2.2 Learning with Queries

All the protocols presented in Section 3.2.1 can be extended to allow the agent to use a certain number of *queries*. Such extensions give rise to settings in which the agent can be *active*, that is, ask the user questions of its choice. There are many classes of concept which cannot be learnt in some passive setting, but which can in the same setting if some active queries are allowed.

Angluin (1988) proposes a list of different queries. The only type of queries used in this thesis are *membership queries*. Such a query is simply a question to the user about some assignment μ chosen by the agent. The user must answer "yes" if μ is indeed in c^*, and "no" otherwise.

3.2.3 Performance Measures

The efficiency of an algorithm for learning in a particular setting is usually measured by its *query complexity* and by its *time complexity*. These measures are defined according to some encoding scheme for concepts. Hence, by the *size* $s(c^*)$ of the target concept we refer to the minimum size of a representation of this concept in the encoding scheme at hand.

Definition 3.6 (query complexity) *Let L be an algorithm for learning a concept in some class. For PAC-learning, the* query complexity *of L is the number of examples and queries which it uses,*

as a function of the number of variables over which examples are defined ($|X|$), the size of the target concept $s(c^*)$, and the quantities $1/\epsilon, 1/\delta$.

For exact identification, the query complexity is defined to be the number of queries used as a function of $|X|$ and the size of c^*. For the KWIK setting, it is defined to be the number of queries and "?" answers used.

Naturally, it is expected from an algorithm to have a polynomial query complexity. When only passive queries are considered, one usually talks of *sample complexity* instead of query complexity.

Definition 3.7 (time complexity) *Let L be an algorithm for learning a concept in some class. For PAC-learning and exact identification, the* time complexity *of L is the number of elementary operations which it uses, as a function of $|X|$ and $s(c^*)$, where receiving an example or asking a query (and receiving the answer) count for one operation. For the KWIK setting, it is defined to be the time complexity for processing* one *example.*

An important desirable feature for a learner L, apart from polynomial query and time complexities, is *attribute-efficiency* (Littlestone, 1988; Blum et al., 1995). The idea is that a concept will typically involve few attributes, whereas there is no reason in general to project the examples (assignments) onto these variables. Rather, the agent is typically presented assignments to, say, 1,000 variables, but there are only 10 of them which are relevant to the target concept (and the learner initially does not know these). Then a learner is said to be *attribute-efficient* if its query complexity is polynomial in the size of the target concept, but only *polylogarithmic* in the number of variables over which examples are defined.

Finally, an important tool in learning theory is the *Vapnik-Chervonenkis dimension* (VC-dimension) of a class of concepts $\mathcal{C}$, which determines to some extent the sample complexity of passive learning algorithms for $\mathcal{C}$. The VC-dimension of $\mathcal{C}$ is defined to be the largest integer d such that there exists a set E of d assignments which is shattered by $\mathcal{C}$. A set E is said to be *shattered by* $\mathcal{C}$ if for any partition of E into two sets E^+, E^-, there is a concept c in $\mathcal{C}$ for which $e \models c$ (resp. $e \not\models c$) holds for all assignments $e \in E^+$ (resp. $e \in E^-$).

Example 3.8 *This notion is easily understood with $\mathcal{C}$ being the class of all intervals of real numbers. The set $E = \{-1, 2\} \subseteq \mathbb{R}$ is shattered by this class, since there is an interval of real numbers which contains -1 but not 2 ($E^+ = \{-1\}, E^- = \{2\}$), there is one which contains 2 but not -1, one which contains both ($E^+ = E, E^- = \emptyset$), and one which contains none. On the other hand, a set of at least 3 real numbers of the form $E = \{a, b, c, \ldots\}$ ($a < b < c$) cannot be shattered by the class of intervals, because there is no interval which contains a and c but not b ($E^+ \supseteq \{a, c\}, E^- \supseteq \{b\}$). Hence the VC-dimension of this class is 2.*

Intuitively, the VC-dimension of a class $\mathcal{C}$ gives a lower bound on the sample complexity of algorithms for learning $\mathcal{C}$. Indeed, if a set of d examples shattered by $\mathcal{C}$ is received in a sequence of examples, then until the last one has been received there are by definition at least two hypotheses in $\mathcal{C}$ consistent with the examples (namely, at least one consistent with the last one being negative, and one consistent with the last one being positive). In fact, much more can be said, including upper bounds on the sample complexity and more generally, on the *query* complexity (Auer and Long, 1999).

3.3 Decision Making

The last type of computational problems which we specifically study in this thesis are *decision-making* problems. We start with decision-making problems according to *known* models.

Definition 3.9 (policy computation problem) *We write* POLICY-COMPUTATION *for the problem defined by*

- *input: an MDP $M = (S, A, T, R)$,*

- *output: an optimal policy π^* for M.*

Clearly enough, for this problem to be well-defined, some representation must be assumed for the given MDP. We will specifically address the case when the MDP is given in factored form, with actions represented in PPDDL, in Chapter 5.

Similarly, the notion of an *optimal* policy must be precisely defined. We will use the cumulative reward criterion, defined as follows.

Definition 3.10 (cumulative reward criterion) *Let $M = (S, A, T, R)$ be an MDP, let $\gamma \in]0, 1]$ be a discount factor, and let h be a positive integer, or ∞ (in which case $\gamma < 1$ must hold). The value of a policy π* under the cumulative reward criterion *is defined recursively, for all states $s \in S$, as*

$$V_\pi^h(s) \;=\; \sum_{s' \in S} T(s'|s, \pi(s)) \;\times\; \big(R(s, \pi(s), s') + \gamma V_\pi^{h-1}(s')\big)$$

with $V_\pi^0(s) = 0$ for all states $s \in S$.

Observe that we abuse notation by writing $h - 1$ for $h = \infty$, but the discount factor γ ($\gamma < 1$ in this case) guarantees that the sum converges. Then an optimal policy is one with maximal values among all policies in each state. Such a policy is guaranteed to exists both in the finite-horizon and in the infinite-horizon cases. For more details we refer the reader to the book by Puterman (1994).

When the model is not known *a priori* but decisions must still be made, the agent faces a *reinforcement learning* problem.

Definition 3.11 (reinforcement learning) *In* reinforcement learning*, there is an MDP $M = (S, A, T, R)$. The agent knows S and A from the beginning, but not T nor R. The agent starts in a given state s_0. Then, at each timestep (to infinity) it is in some state $s \in S$, chooses an action $a \in A$, is transported to some state $s' \in S$ sampled according to the distribution $T(\cdot|s, a)$, and receives and observes the reward $R(s, a, s')$.*

The goal of the agent is to be able to make decisions so that a small number of these are suboptimal (for M). There are several formal definitions of this, with which we will not be directly concerned in this thesis. For more details we refer to Sutton and Barto (1998) and Kakade (2003).

Chapter 4

Representations and Problems in Action

Contents

This chapter places in a concrete context all subsequent studies together with relevant results from the literature. We show how we intend symbolic representations of the environment to be used by a concrete agent, together with logical fragments and compact representations of preferences and decision processes, as presented in Chapter 2, and efficient algorithms for the "deliberative" problems presented in Chapter 3.

The purpose of this chapter is not to design an architecture for an agent, but rather to show how representations and algorithms can interact together and enable concrete agents to fulfill their missions, from the application point of view. To make things concrete, we build on the example of software assistants presented in the introduction. We first develop this scenario with some more details (Section 4.1), and then discuss how the agents can handle perceptions and make decisions.

We have purposedly chosen a nontrivial scenario so as to point out not only contributions, but also limits of our settings and studies. Moreover, so as to focus on the "intellectual abilities" of our agent, we assume that some complex actions and perceptions are available, in particular for Natural Language Processing.

4.1 The Scenario

As presented in the introduction, our example agent is installed on a user's electronic devices (her laptop, her smartphone, etc.), and its overall mission is to help her scheduling her various obligations, including daily tasks, deadlines, organization of meetings with other participants, etc. Ideally, after some time the agent will have a sufficiently precise knowledge of its user for making almost all decisions on her behalf. An example such decision would be to answer requests from other persons about the user's available and preferred dates for a given meeting to be held. A

maybe more cautious decision would be to *suggest* the user to answer so and so, and explain her why.

Example 4.1 *Assume the agent has learnt (believes) that its user is never available on Wednesdays for her work. Also assume that it knows of a very time-consuming task which the user has to achieve, with a firm deadline at the end of the current week. Then if the agent catches an incoming e-mail requesting the user to be present at a full-day meeting on the next Friday, it may suggest the user to work at her time-consuming task as soon as possible (so as to avoid having to both attend the meeting and finish the task on Friday). If it later notices that the user is indeed trying to finish this task on Wednesday morning, then it must revise its belief that she is never available for work on Wednesdays.*

In general, let us assume that the agent is able to perceive the following events and information in its environment:

- a request coming in for participating in some event at a given date, e.g., "you are invited to the kick-off meeting of Project X, to be held on Wednesday, 04:00 PM",
- the user answering such a request, e.g., the agent catching the answer "I'll be there but a little later than 04:00",
- the user logging in/out from some device, e.g., turning her mobile phone off,
- the user emitting a request to other people for participating in an event, e.g., "please participate in the poll for choosing one hour next week where we can have a drink together",
- the user not being available at such dates, e.g., writing "Monday 4th–Sunday 24th: holidays" in her agenda,
- etc.

On the other hand, assume that the agent has the following actions at its disposal (where examples are chosen to illustrate what the agent would *ideally* be able to decide to do):

- ask a question to the user, e.g., "may I accept this invitation to a meeting on Thursday on your behalf?", or "on which day would you prefer finishing your IJCAI article?",
- answer some incoming e-mail, e.g., "my user will not be able to participate in your meeting, since she will be at another meeting for some work that she finds more important",
- initiate some e-mails, e.g., "could you soon propose a date for your monthly meeting, since my user will be in holidays from the end of next week on?",
- etc.

Recall from the introduction that we aim at agents which are not initially tailored to a specific user. Said otherwise, we do not want the user to undergo a long kick-off phase where she constantly must inform the agent about her preferences, her habits, etc., without anything in return. Hence our agent will not have an overall, hard-wired mission. It will have to learn what decisions, or what situations, are "good" or "bad" from the user's viewpoint (and hence, from *its* viewpoint).

Obviously enough, our agent will need *variables* at its disposal. Some of these will be directly mapped to its perceptions of the current state of the environment (when available), e.g., variables denoting the current hour, day, month, etc., variables denoting whether the user is currently logged in such and such device, whether the question it has just asked to the user is currently displayed on the screen, etc. Other variables will allow the agent to represent specific objects, say, e-mails (by their sender, receiver, content-type — request for a meeting, answer, obligation...) or dates (through attributes such as day, month, etc.), but also, say, the user's availability. This last descriptor will typically be nonobservable for most dates.

Finally, our agent will also need additional variables able to represent more abstract concepts. A good example is a variable, say, "busy", telling whether the user is busy or not in a given situation.

For the context of this thesis, let us assume that this set of "meaningful" variables is given once and for all to the agent (at the moment of its construction — or compilation). We can however assume that the agent is able to create new variables, as desired, as possibly required by internal computations.

4.2 Handling Perceptions

We start with a description of how our agent can handle its perceptions, that is, mostly how it can react to external events.

4.2.1 Incoming Requests

First consider the case when a request comes in, for the user to participate in an event at a given date, e.g., "you are invited to the kick-off meeting of Project X, to be held on Wednesday, 04:00 PM". This clearly should define a temporary "goal" for our agent, namely, answering this request on behalf of the user. The first, obvious treatment to apply to such a request is to work out whether it is honourable at all. This is a *deduction* problem, precisely, this amounts to decide whether what the agent believes to be true about the user's agenda is consistent with the proposition "the user will be at the kick-off meeting of Project X on Wednesday, 04:00 PM". Note that this is more than a simple check with the agenda: this deductive process should take into account transportation from one place to another if the user has another obligation until, say, 3:30 PM on Wednesday, it should take into account beliefs such as "my user does not want to meet such person more than once a week", etc.

If it turns out that it is *impossible* for the user to attend the meeting, as far as the agent knows, the agent may simply ask confirmation to the user and then answer "no" (we will discuss later the case when dates can be negociated). In case it is *possible*, the agent may again ask confirmation to the user, answer "yes" and add the appointment to the user's agenda (and to its own knowledge base).

Now the most interesting case is when it *could* be possible for the user to attend the meeting, but this would require to cancel or postpone another obligation. Finding such obligations is an *abductive* process. Precisely, given general rules known to the agent (about the user's preferences and hard constraints, about transportation, etc.), it is an abductive process to find out what obligations together with these rules entail the impossibility to attend the meeting. Abducibles are precisely the current obligations, and the explanations found are cancellations to be proposed to the user, if she wants to honour the request. Abduction has been the focus of our work in a series of articles (Zanuttini, 2002a; Nordh and Zanuttini, 2005; Creignou and Zanuttini, 2006; Nordh and Zanuttini, 2008).

When presented such a choice of obligations to cancel so as to be able to attend the meeting, the user may want to test different hypotheses, such as "what if I work half an hour less on my IJCAI article?". An interesting formalization of such interactions is given by *configuration problems* (Amilhastre et al., 2002), which themselves involve deductive and abductive processes.

These processes are those directly related to answering the request. An autonomous agent should however be able to take any occasion to *learn* about its environment. The mere fact of receiving the request already provides an occasion to learn things. As far as *concept learning* is concerned, it provides an example of the concept "possible request" (to be distinguished from the syntactic description of what a request is). This concept should now be extended, if necessary, so as to encompass the possibility of requests for meetings on Wednesdays. In this manner, for such user the agent could in the longterm learn that this user is requested for meetings only during weekdays and during office hours, while for such other user it could learn that there are some "possible requests" for meetings in the evening. Having a precise definition of this concept is

clearly important for the agent. For instance, if its goal at hand is to plan some work for the next month so that the plan is (almost) guaranteed not to change, knowing what requests are likely to come in in the meantime is crucial. Concept learning is the focus of our Chapter 9.

4.2.2 Learning from the User's Answers

We do not imagine our agent as replacing the user in all interactions, but rather as helping her. Hence there will typically be many answers to incoming requests which the user will answer directly. Monitoring such answers are the occasion for the agent to *learn* from *passive observation* of the user, in particular, without any effort from her.

Assume that the user answers the request for a meeting on Wednesday, 04:00 PM by an email such as "I'll be there but a little later than 04:00", and the agent catches this answer. First assume that according to the agent's knowledge, the user had another obvious possibility, namely, being on time at the meeting but at the price of doing some work *after* the meeting instead of *before*, as currently planned. Then for the agent this is an example of the user's *preferences* on two options, namely, being on time at the meeting but then working a little late in the afternoon, or being late at the meeting but not working late in the afternoon. We study learnability issues for such preferences in Chapter 7.

Now assume that the user has no reason to be late on Wednesday, 04:00 PM, according to the agent's current beliefs. Precisely, assume that arriving late on Wednesday is not the optimal *rational decision* according to these beliefs. A plausible explanation for this is that the agent misses some information. Then a way to *learn* about the current state of the environment is again to perform some *abduction* so as to find candidate explanations of why the user chose the option to be late. These candidate explanations can then be confirmed by questions asked to the user (when she has time to answer), or simply kept as constraints, at least one of which must be true in the current state of the environment.

4.2.3 More Occasions to Learn from Observations

So far, we have presented how our agent can learn about the user's preferences, and about various concepts, by taking incoming requests and answers by the user as examples.

There are many other occasions for the agent to learn from observations during its lifetime. For instance, a clearly useful concept to be learnt is one describing when the user is "reachable", which we intend as "likely to answer a question by the agent". Typical observations (examples) from which this concept can be learnt are her logins and logouts of the various devices on which the agent runs. From these the user should be able to infer rules such as, say, "my user is never logged in on Sunday mornings" (which entails that she is unreachable on Sunday mornings). Other examples of this concept are also all situations in which the user answers a question by the agent (positive example), or in which the agent asks a question but the user does not answer (negative example). In the same vein, the agent has many occasions to learn the concept of "typical day off", "typical arrival hour at work", etc.

We use the term "typical" above to emphasize the need for being able to learn concepts which are not pure Boolean concepts. The agent will typically face problems having to do with *noise*. For instance, in our context, the user could indeed *typically* work on Mondays, but some negative examples could come with noise, e.g., the user could be absent from work on some Monday because she is ill, because bus drivers are on strike, etc. Another problem is that the concept itself may be fuzzy. For instance, it may be the case that "typically" means "most often". One framework for encompassing such fuzziness is the framework of *probabilitic concept learning* (Kearns and Schapire, 1994). Nevertheless, we will not address these issues here, rather focusing on pure Boolean concept learning in Chapter 9.

When it comes to *preferences*, again the agent may learn without requiring any effort from the user, by monitoring her choices among alternatives which it proposed. For instance, assume that the agent suggests the user to plan some work on the IJCAI article on Tuesday and have a day off on Wednesday, or, as a second choice (according to the agent's beliefs), to have first a day off

and then work on the article. Then if the user chooses the second alternative, the agent has the occasion to learn more about her preferences and to revise its beliefs. We consider this question with some details in Chapter 7.

4.3 Proactive Behaviour

In Section 4.2 we have considered our agent's behaviour mostly as reactions to some events happening. We now consider enrichments of such reactions by *proactive* behaviours.

4.3.1 When to Think, When to Act, and When to Chat

In an application such as ours, an agent will typically have a lot of "spare time", especially when the user is not logged in and almost no request comes in (think of nightime). Hence this time can be used for performing time-consuming computations. This remark fits well in particular in the framework of *knowledge compilation* (Liberatore, 1998; Darwiche and Marquis, 2002), but also in the framework of decision-making, when a contingent policy is computed off-line (at nighttime, in our case) and then used as a controller for reactive behaviour.

We will not directly study compilation problems in this thesis, but observe that tractable fragments of propositional logic are typical target languages for compilation. This has been studied in depth by Selman and Kautz (1996) and del Val (1995) in the context of *approximate* compilation, that is, the case when the target fragment is not fully expressive. Tractable fragments have also been used recently for constructing *fully expressive* target languages (Fargier and Marquis, 2008). Using such compilation schemes together with results about the complexity of problems for each tractable fragment suggests that the agent may use its spare time for computing approximations or disjunctive covers of its knowledge bases and concept descriptions, and then use these approximations and covers for efficient on-line reasoning.

There is however at least one task for which the agent needs some "spare time", but also needs the *user* to have some time. This is the generic task of *active learning*, for which the agent needs to ask questions to the user so as to refine its knowledge (about concepts or about her preferences). Clearly, it is not reasonable to ask questions to the user at any moment. At least, when the current state is classified as one where the user is "unreachable", the agent should not ask questions. Hence the agent must be able to somehow store its questions, and query the user only when she is prone to answer. As we will see in Chapter 7, being able to query the user may indeed be necessary for being able to learn her preferences.

4.3.2 Handling Requests and Negociating Fast

In Section 4.2 we described a behaviour for our agent, when a request comes in for participating to a meeting at some given date. In a more general setting, the request may be one for attending a meeting, and suggesting dates for it to be held, or at least revealing the user's available dates for it. In particular, in such a meeting there is typically a negociation phase, where the requester and the agent will exchange information in order to reach an agreement on a date.

We will not study negociation in this thesis, but what is of interest here is the fact that such exchanges should be reasonably fast. One typically appreciates when all persons invited to meet answer the request rapidly, and stay tuned long enough for enabling convergence to a date at which everyone is able to attend. If one ever needs one or two hours before continuing the negociation, then the discussion usually runs short, and some other negociation phase has to be started from scratch later on.

An agent using knowledge bases and preference representations at the scale which we imagine here will certainly not be able to answer all requests rapidly from scratch. If complex inferences are needed so as to decide on a date or to suggest one, it may be the case that performing such inferences from scratch takes hours, which is not acceptable in negociation phases. Hence what is needed is *precomputed* policies, which enable the agent to react fast when needed.

Computing such policies in stochastic environments is the focus of Chapter 5. There we consider actions with probabilistic outcomes described over attributes. In our context, stochasticity enables the agent to model the uncertainty of its actions. In particular, stochastic outcomes in a negociation phase can represent the probabilities that one of the participants in the discussion stops participating, that the agent asks the user a question but gets no answer, or even the probability that a suggested date gets accepted or not. Having precomputed policies allows the agent to react rapidly, at negociation time, to each such event.

Another interesting possibility for an agent is the ability to be *self-aware* of its computational abilities. Precisely, it is interesting for an agent to be able to decide to use one knowledge base or the other, or one approximation of some concept or the other, for reasoning and deciding, depending on the quality (or completeness) required for the inferences drawn and the decisions made, and the time available for making them. Tractable fragments and Post's lattice offer interesting possibilities for such self-awareness, and we investigate this point in details in Chapter 8.

4.3.3 Learning and Acting at the Same Time

In the previous section we briefly discussed the need for being able to represent *probabilistic* outcomes for actions, such as the probability to see a proposed date accepted or not by such participant in the discussion. Clearly enough, such probabilities cannot be given to the agent once and for all.

Hence the agent needs to *learn* such probabilities, and more generally the model of its actions. For instance, it must be able to learn under what conditions it can ask a question to the user and get an answer immediately (the concept of her being "reachable"), but also with what probability, when these conditions are not met, it will nevertheless get an answer. Here, probabilities may counterbalance the absence of enough descriptors for determining for sure whether the user will answer or not, or simply be inherent to the action, because the typical user is not "deterministic" (however precisely we know the current situation).

Learning models of probabilitic actions is the subject of Chapter 6. There we view this problem essentially as a concept learning problem, but the agent typically faces a more difficult problem. Namely, the agent can learn probabilities as above only by *indeed interacting* with participants in a negociation phase or with the user. This is not a system which can be simulated again and again, hence the occasions for the agent to learn are precisely those where it must act at best. This is precisely the setting of *reinforcement learning*, which we also discuss in Chapter 6. When moreover preferences must be taken into account, we face a still more difficult problem, which we discuss in Chapter 7.

4.4 Summary

To summarize this example, our agent faces *reasoning* problems, in particular when it must evaluate the current situation. Deciding on a definitive answer to some request may be the conclusion of a *deductive* process, while trying to resolve the impossibility to answer positively can be formalized as an *abductive* process.

Each event can be seen as a positive or negative example of one or several concepts, including the preference relation of the user. Other occasions to learn concepts are by series of questions asked to the user, while the occasions to learn models of its actions are typically the real interactions themselves, in which the agent must satisfy its goals while it is learning.

We leave to other research projects the question of determining the relevant descriptors to be embedded in such an agent. Assuming that they exist, the agent can use them to represent objects (e.g., e-mails) and contexts (e.g., the situation at a precise moment). Building on them it can learn the descriptions of complex concepts and action models, and represent the user's preferences. Using several approximations of these concepts and using theoretical tools related to tractable fragments, it can be aware (to some extent) of its own expressivity/complexity tradeoff, and decide on the quality of its decisions depending on the time available for taking them.

This discussion is intended to give a proof of concept that agents *may* be designed which use the techniques and tools studied in this thesis. Indeed designing them would constitute a complete research project. Moreover, as outlined at many places, there are of course many computational problems and representation languages which are essential for such agents, and which we do not study here. Other essential aspects which we do not study in this thesis, apart from the design of the "physical" abilities of such agents (in our case, monitoring several devices, catching and sending e-mails, etc.), are natural language processing and synthesis, which are essential for real-world human-agent interaction, architectures (from the high-level representation of goals, beliefs, etc. to software engineering questions), representation choices for the environment (choice of descriptors), and "social" abilities enabling the agents to negociate, argue, reach collective decisions, etc.

Chapter 5

Decision Making in Factored MDPs

Contents

This chapter presents original results and perspectives for decision making in the framework of Markov Decision Processes (MDPs), using factored, propositional representations. The main originality of the work presented lies in its handling of very compact representations of actions (probabilistic STRIPS operators and even grounded PPDDL), while almost all existing work has focused on representations by Dynamic Bayesian Networks. We also point at promising directions for the problem of revising policies when the model of actions changes, which yields perspectives for control and (model-based) reinforcement learning.

The work and ideas presented here have been developed with Boris Lesner in the context of his PhD. The work presented in Sections 5.3–5.5 is also reported, with some complementary details, in a conference article (Lesner and Zanuttini, 2011). The preliminaries relevant to this chapter are those given in Sections 2.4 (Page 15) and 3.3 (Page 27).

5.1 Structured Representations of MDPs

Recall that a *Markov Decision Process* (MDP) is given by a 4-tuple (S, A, T, R), where S is a set of states, A is a set of actions, $T : S \times A \times S \to [0, 1]$ is a transition function (we write $T(s'|s, a) \in [0, 1]$

for the probability of ending in state s' when taking action a in state s), and $R : S \times A \times S \to \mathbb{R}$ is a reward function ($R(s, a, s')$ is the reward obtained for taking action a in state s and ending in state s').

As presented in details in Section 2.4, when states are described over propositional variables, that is, each state s is a complete assignment to a set of propositional variables X, three main representations have been proposed:

- in the representation by Dynamic Bayesian Networks (DBNs), for each action a the distribution on next-states is represented by a Bayesian Network for each variable x, which gives the probability that a sets x to true in the next-state depending on the values of some variables (pre- and post-action),
- in the representation by Probabilistic STRIPS Operators (PSOs), for each action a a list of *conditions* $c_1, \ldots, c_k$ is given, and for each condition c_i a probability distribution D_i on *effects* is given, where each effect is a consistent term,
- in the representation in the grounded Probabilistic Planning Domain Description Language (PPDDL), which generalizes PSOs, the representation of an action is defined recursively to be a consistent term (as for PSOs), a conditional effect (`when` φ e), where e may itself be a complex representation, a conjunctive effect (`and` e_1 e_2 $\cdots$), where e_i's must be consistent together, or a probabilistic effect (`prob` p_1 e_1 p_2 e_2 $\ldots$).

Before discussing work related to the study presented in this chapter, we first outline the main results about the relative succincteness of, and transformations between, all three representations.

PPDDL and PSOs A straightforward observation is that PPDDL generalizes the representation of actions by PSOs, and thus offers at least as compact a description for any action. To see this, simply observe that an effect distribution $\{(e_{ij}, p_{ij})\}$ for a condition c_i in a PSO representation translates directly to the PPDDL construct (`when` c_i (`prob` e_{i1} p_{i1} e_{i2} p_{i2} $\ldots$)), and that all such pairs condition/effect distribution can be linked together by the PPDDL operator `and`. Moreover, PPDDL may be exponentially smaller, since, for instance, the construct

$$(\texttt{and}\ (\texttt{when}\ x_1\ (\texttt{prob}\ p_1\ x))\ (\texttt{when}\ x_2\ (\texttt{prob}\ p_2\ x))\ \ldots\ (\texttt{when}\ x_n\ (\texttt{prob}\ p_n\ x))) \tag{5.1}$$

has 2^n conditions, each of which has one effect (x) with probability $1 - \prod_{i \in I}(1 - p_i)$ for one index set $I \subseteq \{1, \ldots, n\}$.

PPDDL and DBNs Things are a little more complex when comparing these representations with DBNs. Indeed, it is common to use DBN representations with the conditional probability tables (CPTs) represented by some compact structure such as Algebraic Decision Diagrams (ADDs). Contrastingly, as a standard language, PPDDL does not enable such representations for conditions of `when` clauses. It follows that DBNs are exponentially more compact than PPDDL or PSOs for an action as simple as the one which sets x to true with probability 1 if at least k variables among $x_1, \ldots, x_n$ are true, and to false otherwise. Indeed, the Binary Decision Diagram for "at least k among n" has kn nodes, whereas representing the same function by a disjunction of terms (as allowed by PPDDL) requires $\binom{n}{k}$ terms. Contrastingly, PPDDL and even PSOs can offer exponentially smaller representations than DBNs, especially when the formulas at hand have small DNFs, as we now show in a more general setting.

Assume that we allow formulas to be represented by Algebraic Decision Diagrams in PPDDL descriptions. This can in fact be assured by a simple preprocessing of plain PPDDL descriptions. Then it is easy to see that PPDDL always offers representations at least as compact as DBNs *without synchronic arcs*. This is because any CPT/ADD with ℓ leaves in such a DBN can be represented by ℓ PPDDL `when` clauses, namely, (`when` c_i (`prob` p_i x $(1 - p_i)$ $\overline{x}$)), where p_i, c_i enumerates the leaves of the ADD and the paths from the root to them, respectively. Moreover,

PPDDL can be exponentially more compact. For instance, the construct (5.1) above has an exponentially large equivalent CPT[1].

For DBNs *with synchronic arcs* the same result holds by a slightly different argument, since primed variables cannot occur in conditions of PPDDL `when` clauses. The argument now is that if a branch of the form $\ell_1 \dots \ell_k \ell'_{i_1} \dots \ell'_{i_{k'}}$, with value p_x, occurs in the ADD representing the CPT for x, then it can be represented in PPDDL by the construct

$$(\texttt{when}\ \ell_1 \wedge \dots \wedge \ell_k\ (\texttt{prob}\ p'\ (\texttt{and}\ \ell_{i_1}\ \cdots\ \ell_{i_{k'}}\ (\texttt{prob}\ p_x\ x\ (1-p_x)\ \overline{x}))))$$

where p' is computed recursively (acyclicity of the dependency graph ensures that this process terminates).

However, a linear-size translation from PPDDL to DBNs has been proposed, which uses auxiliary variables. The idea is to represent each effect e over several variables in a PPDDL description, by a fresh variable which acts as a "meta-variable" for the effect. The probability and parents (variables in the condition of the effect) are given to the meta-variable, which itself has a synchronic, deterministic arc to the variables in e. It can be seen that with some work for handling nested PPDDL descriptions, adding such fresh variables allows for an efficient translation.

PSOs and DBNs The picture for PSOs and DBNs is less clear. In fact, it can be shown that even if ADDs are allowed in PSO descriptions, there are actions which can be expressed succinctly by DBNs but not by PSOs, and vice-versa. A way to make PSOs always as compact as DBNs is to enable *aspects* in PSO representations (see the discussion by Boutilier et al. 1999, Section 4.2.4). We however do not elaborate on this point here, because we are concerned with PPDDL descriptions, and most related work is concerned with DBN representations.

Summary Over the same set of variables PPDDL is as succinct as PSOs and DBNs, and possibly exponentially smaller, provided the conditions in its `when` clauses can be represented by ADDs. However, if auxiliary variables are allowed in DBNs, then PPDDL and DBNs are equally succinct. Note however that these additional variables may be numerous (one per branch in the PPDDL tree). On the other hand, PSOs and DBNs are incomparable from the point of view of succinctness, except if aspects are allowed for PSOs. For more on these succinctness issues we refer the reader to Littman (1997), Boutilier et al. (1999), Rintanen (2003), Younes and Littman (2004).

5.2 Related work

The main result reported in this chapter is an algorithm for computing optimal policies of Markov Decision Processes (under the discounted total-reward or the finite-horizon cumulative reward criterion), starting from a description of actions in grounded PPDDL.

When optimal policies are sought for, efficient algorithms have been known for a long time if the state space and transition functions are given explicitly. The famous *Value Iteration* algorithm, on which we will build in this chapter, works backwards, by computing the expected value of the best action in each state for one step remaining (this step is called an *action backup*), then using these values for computing the expected values with two steps remaining, and so on. This algorithm is depicted on Figure 1 for the finite-horizon, γ-discounted, cumulative reward criterion.

Example 5.1 *Consider again the MDP of Example 2.17, reprinted here on Figure 5.1 (recall that transitions are labelled with triples giving the action a, probability of transition $T(s'|s,a)$, and reward $R(s,a,s')$. Value iteration works as follows at horizon $h = 3$ and with discount factor $\gamma = 1$ (no discount).*

[1] A noisy-or representation could be used, but this would require the algorithms for computing policies to explicitly handle this compact representation.

Algorithm 1: Value Iteration (at horizon h)

foreach *state* s **do** $V(s) \leftarrow 0$;
for $i = 1, 2, \ldots, h$ **do**
  foreach *state* s **do**
    $V_{current}(s) \leftarrow -\infty$;
    foreach *action* a **do**
      $Q^a_{current}(s) \leftarrow \sum_{s' \in s} T(s'|s, a) \times \ (R(s, a, s') + \gamma V(s'))$;
      $V_{current}(s) \leftarrow \max(Q^a_{current}(s), V_{current}(s))$;
    end
  end
  foreach *state* s **do** $V(s) \leftarrow V_{current}(s)$;
end
foreach *state* s **do**
  $\pi(s) \leftarrow argmax_{a \in A}\, Q^a_{current}(s)$;
end
return π;

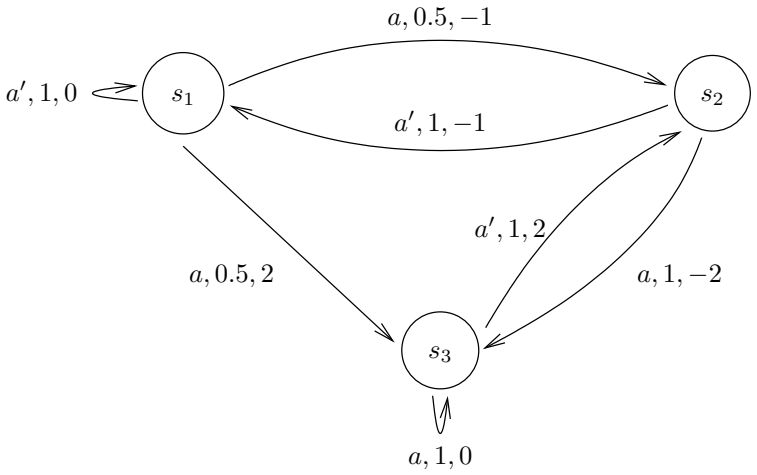


Figure 5.1: An example Markov Decision Process

With one action remaining, the action values are the following:

$$\begin{aligned} Q^a(s_1) &= 0.5 \times (-1) + 0.5 \times 2 = 0.5 \\ Q^{a'}(s_1) &= 1 \times 0 = 0 \\ Q^a(s_2) &= 1 \times (-2) = -2 \\ Q^{a'}(s_2) &= 1 \times (-1) = -1 \\ Q^a(s_3) &= 1 \times 0 = 0 \\ Q^{a'}(s_3) &= 1 \times 2 = 2 \end{aligned}$$

From this it follows that the best action is a in state s_1 and a' in states s_2, s_3, from what we get the value function:

$$V(s_1) = 0.5, \quad V(s_2) = -1, \quad V(s_3) = 2$$

Now with two actions remaining, the process is similar but with next-state values defined by the current value function V. We get:

$$\begin{aligned} Q^a(s_1) &= 0.5 \times (-1 + V(s_2)) + 0.5 \times (2 + V(s_3)) \\ &= 0.5 \times (-1 + (-1)) + 0.5 \times (2 + 2) = 1 \\ Q^{a'}(s_1) &= 1 \times (0 + V(s_1)) = 1 \times (0 + 0.5) = 0.5 \\ Q^a(s_2) &= 1 \times (-2 + 2) = 0 \\ Q^{a'}(s_2) &= 1 \times (-1 + 0.5) = -0.5 \\ Q^a(s_3) &= 1 \times (0 + 2) = 2 \\ Q^{a'}(s_3) &= 1 \times (2 + (-1)) = 1 \end{aligned}$$

and

$$V(s_1) = 1, \quad V(s_2) = 0, \quad V(s_3) = 2$$

Observe that contrary to the first stage, a is now the best action in all states.

Finally, with three actions remaining we get:

$$\begin{aligned} Q^a(s_1) &= 0.5 \times (-1 + 0) + 0.5 \times (2 + 2) = 1.5 \\ Q^{a'}(s_1) &= 1 \times (0 + 1) = 1 \\ Q^a(s_2) &= 1 \times (-2 + 2) = 0 \\ Q^{a'}(s_2) &= 1 \times (-1 + 1) = 0 \\ Q^a(s_3) &= 1 \times (0 + 2) = 2 \\ Q^{a'}(s_3) &= 1 \times (2 + 0) = 2 \end{aligned}$$

and

$$V(s_1) = 1.5, \quad V(s_2) = 0, \quad V(s_3) = 2$$

Then value iteration computes a (3-steps-to-go) policy by choosing the action which yields these values in each state, namely, a in state s_1 and either a or a' (arbitrarily) in states s_2, s_3.

Other algorithms such as *Policy Iteration*, *Modified Policy Iteration*, or algorithms based on *Linear Programming* have also been proposed. We refer to their in-depth description and analysis by Puterman (1994, Chapter 6), and to the presentation by Garcia and Rachelson (2010).

A common drawback of these approaches is that their time and space complexity depends on the total number of states, which is 2^n if there are n variables. A notable exception is *Sparse Sampling* (Kearns and Singh, 2002), whose time complexity is independent from the number of states in the MDP, but exponential in the horizon time (i.e., the number of steps to ϵ-convergence). But if the problem is described by structured representations of size, say, polynomial in n, then

we clearly expect a "good" algorithm to have a complexity polynomial in n (and in the horizon time) instead of 2^n.

There have been a number of works which propose algorithms which exhibit a good behavior, at least in practice. It is indeed difficult to give a polynomial-time guarantee in the worst-case, because in general a representation of the problem of size polynomial in n does not guarantee that the optimal policy will also have a polynomial-size representation. Even *output-polynomial* time guarantees are unlikely to be given, since in general the exact algorithms compute successive approximations of the optimal value function, and nothing guarantees that such intermediate value functions have size polynomial in that of the input and output. We refer the reader to the discussion about model minimization for MDPs by Givan et al. (2003) about this point. Also note that Kearns et al. (2002) show that an exponential dependency on the horizon time is unavoidable, under some assumptions (for the discounted infinite-horizon optimality criterion).

Nevertheless, SPUDD (Hoey et al., 1999), which operates on DBN representations (possibly with synchronic arcs) and performs computations on ADDs, shows a very good behavior in practice. It has recently been revised to use *Affine* ADDs (Sanner and McAllester, 2005), which seems to further improve its behavior. Nevertheless, when the description at hand is "too far" from being a DBN representation, many auxiliary variables must be introduced, which can make the intermediate ADDs grow too much. Hoey et al. (1999) propose techniques for precomputing some products of DBNs, so as to have them directly applicable at each iteration, but in some cases this "transition matrix" may grow too large to be useful. This would be the case, for instance, with the construct (5.1) above.

Now as far as *approximate* policies (not guaranteed to be optimal) are sought for, various techniques for computations on DBN representations have been proposed, e.g., by St-Aubin et al. (2000), by Feng and Hansen (2002). and by Guestrin et al. (2003).

Finally, there has been a recent excitement about computing policies directly for *relational MDPs* (recall that PPDDL builds on first-order logic). However, the techniques proposed largely try to extend known methods for the grounded setting, and as far as we know the practical benefit is (still) marginal. Anyway, the theory is well-advanced, and in particular there are generalizations of BDDs/ADDs to the first order. Recently published work on this topic is by Sanner and Boutilier (2009), Lang and Toussaint (2010), and Wang et al. (2008) (for generalizations of BDDs/ADDs).

5.3 Frameless Action Values

Recall that a basic ingredient of the definition of optimal value functions and policies for MDPs is the notion of an *action value function*, also known as a *Q-function*. The Q-function of an action a with respect to some value function V, which gives the value expected to be obtained at the next-step, is the function defined by

$$Q_V^a(s) = \sum_{s' \in S} T(s'|s,a)(R(s,a,s') + \gamma V(s'))$$

In words, $Q_V^a(s)$ is the expected value of performing action a in state s and receiving the value given by V in the resulting state.

An assumption common to all factored representations for MDPs presented in Section 5.1 is the following. When the agent takes action a in state s, and effect e occurs, the value of all variables not explicitly affected by e persist from s to the resulting state $s' = \mathit{apply}(s,e)$. Accordingly, given a one-step expected value function V, action value functions Q_V^a are formally defined for all states s by

$$Q_V^a(s) = \sum_{e \in D(a,s)} p_e \times (R(s,a,\mathit{apply}(s,e)) + \gamma V(\mathit{apply}(s,e)))$$

where $D(a,s)$ denotes the distribution of effects induced by a on s (cf. Section 2.4.4, Page 18), and p_e refers to the probability of effect e in this distribution. In particular, Q_V^a is defined to be the expected value of performing action a, given that the effects of a apply, *and the value of other variables persist.*

Example 5.2 *Let V be the value function defined by $V(x_1, x_2, x_3) = x_2 + x_3$, and let a be an action with two effects in all states, namely, x_2 with probability 0.4 and $\overline{x}_2$ with probability 0.6. Assume moreover $R(s, a, s') = -1$ for all states s, s'. Then we have*

$$Q^a_V(001) = 0.4 \times (-1 + \gamma V(011)) + 0.6 \times (-1 + \gamma V(001))$$

We now propose the notion of a *frameless action value* (F-value for short), which removes this latter assumption by assuming nothing about the effect of a on the values of variables not explicitly affected. In other words, the F-value for a gives the expected value for performing a, given that the explicit effects occur but *leaving the possibility for other variables to evolve in any manner.*

So as to distinguish the value of variables in the current state (as used in the first argument of an F-value, and in conditions in `when` constructs) from those of variables at the next state (as used in the next-step value function V and in effects), we duplicate each descriptor x to a variable x', representing the value of x at the next step. This notational convention is common in many areas of computer science. Then by using the notation V' instead of V we stress that the value function V concerns next-state variables. Formally, $V'(s, s') = V(s')$.

Hence the F-value $F^a_V(s)$ of a particular state is not a scalar value (as is a Q-value), but is itself a function, which takes as argument an assignment to a set of descriptors X' describing how "other variables" evolve; in equations,

$$\forall s \in S, F^a_V(s) : 2^{X'} \to \mathbb{R}$$

Example 5.3 (continued) *Using the action a and value function V of Example 5.2, we have that $F^a_V(001)$ is a function from $2^{\{x'_1, x'_3\}}$ to $\mathbb{R}$. This function satisfies $F^a_V(001)(01) = 0.4 \times (-1 + \gamma V(011)) + 0.6 \times (-1 + \gamma V(001)) = Q^a_V(001)$, but $F^a_V(001)(00) = 0.4 \times (-1 + \gamma V(010)) + 0.6 \times (-1 + \gamma V(000)) \neq Q^a_V(001)$.*

Of course, what the "other variables" are depends on the particular effect which occurred, as chosen by random sampling among the stochastic effects of a in s. We take this into account by letting the argument of $F^a_V(s)$ be an assignment μ' to the set X' of *all* primed variables, but keeping in mind that in the state s' reached from s when a particular effect e is applied, only the variables not mentioned in e count.

Definition 5.4 (F-value, Lesner and Zanuttini 2011) *Let $V : S \to \mathbb{R}$ be a value function and a be an action. The* F-value of a with respect to V *is the function $F^a_V : S \times 2^{X'} \to \mathbb{R}$ such that for all states s and assignment to variables μ':*

$$F^a_V(s, \mu') = \sum_{e \in D(a,s)} p_e \times (R(s, a, s'_{e,\mu'}) + V(s'_{e,\mu'}))$$

where, given an effect e, $s'_{e,\mu'}$ is the state obtained from s by applying e, then setting all variables not mentioned in e to their value in μ': $s'_{e,\mu'} = apply(s, e)_{|Vars(e)} \cdot \mu'_{|X \setminus Vars(e)}$.

Observe that we have not taken the discount factor γ in the definition of an F-value. This is for reasons that will become clear in Section 5.4. Anyway, this factor can be taken into account by simply considering the F-value with respect to γV instead of V.

Example 5.5 (continued) *Let again $V(x_1, x_2, x_3) = x_2 + x_3$, but now let a' be an action with two effects, x_3 with probability 0.3 and $x_2 x_3$ with probability 0.7. Assume again the cost of a' to be constantly -1. Then the F-value of a' with respect to V satisfies $F^{a'}_V(000, 100) = 0.3 \times (-1 + \gamma V(101)) + 0.7 \times (-1 + \gamma V(111))$. The next-state value for x_1 is taken into account in both outcomes, but the one for x_2 is taken into account only in the first outcome (since the second one already sets x_2 to some value).*

Observe that, contrary to Q-values, the state s in which a is taken now influences the expected value only in the sense that it conditions the distribution of effects $D(a, s)$. Since no persistence of values is assumed, the values of variables in s has no influence on their values in any next-state s'.

Clearly enough, F-values embed richer information than Q-values. In particular, from an F-value the corresponding Q-value can be retrieved efficiently.

Proposition 5.6 *Let a be an action and V be a value function. Then given the F-value of a with respect to V, the Q-value of a with respect to V can be retrieved in all states s by*

$$Q_V^a(s) = F_V^a(s, s)$$

This is easily seen to be true, as the construction corresponds to prescribing the variables not affected by the action to "evolve" by keeping the value which they had in s.

Example 5.7 *Let V be the value function defined by $V(x_1, x_2, x_3) = 4x_1+2x_2+x_1$ (hence assigning a different value to each state), and let a be an action with effects x_2x_3 (probability 0.1) and $\emptyset$ (probability 0.9), and with constant reward -1. Then for all x'_1, x'_2, x'_3 we have $F_V^a(000, x'_1x'_2x'_3) = 0.1 \times (-1 + \gamma V(x'_1 11)) + 0.9 \times (-1 + \gamma V(x'_1x'_2x'_3))$. The Q-value of a in state 000 can be retrieved from this F-value as $Q_V^a(000) = F_V^a(000, 000) = 0.1 \times (-1 + 3\gamma) - 0.9$.*

The counterpart of embedding more information is of course that the representation of an F-value is always at least as large as the representation of the corresponding Q-value. This is because if any factorization can be applied to an F-value, then it can *fortiori* be applied to the corresponding Q-value. In pathological cases where few variables are explicitly modified by the effects, but all combinations of variables have different values according to V, the representation for the F-value may require up to $\Theta(2^{2n})$ space, while the one for the Q-value will always require $O(2^n)$ space.

5.4 Computing Policies for PPDDL Representations

The most direct application of F-values is an original algorithm for performing action backups, which is able to operate directly on (grounded) PPDDL descriptions of actions, with no syntactical restriction. Plugging such backups into a classical Value Iteration scheme yields an algorithm for computing optimal policies. Our algorithm works by applying rules specific to each PPDDL construct, recursively on the PPDDL representation of the action a at hand. Accordingly, we call it *Rule-Based Action Backup* (RBAB).

5.4.1 Main Ideas

To catch the basic idea of RBAB, consider a (next-step) value function V, two deterministic effects e, e', and the PPDDL construct (`and` e e'). This construct states that e and e' both apply to the state s at hand, and by the consistency assumption in PPDDL, they must be consistent together. Then RBAB exploits the fact that applying (`and` e e') to s with expected value V can be decomposed into

1. applying e to s with next-state values given by V *without assuming that the values of unaffected variables persist*, resulting in an F-value function $\lambda\mu'.F_V^e(s, \mu')$,
2. applying e' to s *with next-state (F-)values given by* $\lambda\mu'.F_V^e(s, \mu')$.

Example 5.8 *Let V be defined by $V(x_1, x_2, x_3) = x_1 + x_2 + x_3$, and let a be an action described by the PPDDL effect*

$$e = (\texttt{and}\ x_2\ (\texttt{prob}\ 0.5\ \overline{x}_3\ 0.5\ x_3))$$

Assume moreover that the reward function R is constantly 0 and let the discount factor be $\gamma = 1$. Then we have $F_V^{x_2}(x_1x_2x_3, x'_1x'_2x'_3) = x'_1 + 1 + x'_3$. Then F_V^e can be computed as $F_F^{e'}$, where e' is the PPDDL effect (`prob` 0.5 $\overline{x}_3$ 0.5 x_3) *and $F = F_V^{x_2}$, and hence $F_V^e(x_1x_2x_3, x'_1x'_2x'_3) = 0.5 \times (x'_1 + 1 + 0) + 0.5 \times (x'_1 + 1 + 1)$.*

Exploiting this principle, RBAB is able to process the PPDDL description of an action incrementally with respect to `and` constructs. This is allowed by intermediate representations of value functions by F-values, and, as we will see in Section 5.5, only by such representations. Moreover, using more common ideas, RBAB is able to process other constructs than `and` in a recursive fashion.

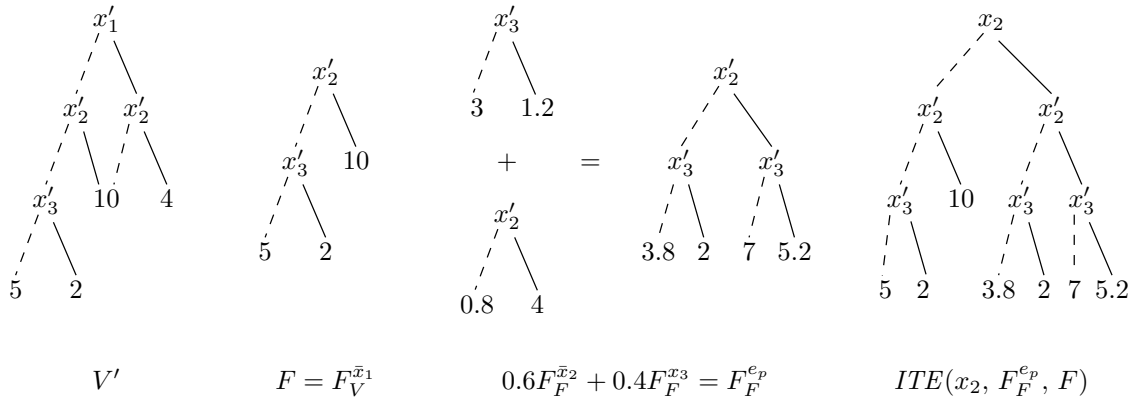


Figure 5.2: An example of RBAB running

5.4.2 The Algorithm

At a coarse grain, RBAB can be described as an algorithm which computes the F-value F_V^a for action a with respect to an expected next-step (traditional) value function V as follows:

1. initialize the result to the F-value F defined by $F(\cdot, \mu) = V(\mu)$,
2. for each construct e in the PPDDL description of a, apply the specific rule to compute the F-value F_F^e of this construct with respect to the current result,
3. return F.

The rules specific to each PPDDL construct are the following, where the operators apply to functions, e.g., $f + g$ is defined by $(f + g)(s) = f(s) + g(s)$, and $ITE(\varphi, f, g)$ ("if-then-else") is the function defined by $ITE(\varphi, f, g)(s) = f(s)$ if s satisfies φ, and $ITE(\varphi, f, g)(s) = g(s)$ otherwise:

- $RBAB(x, V') = \exists x'(V' \wedge x')$,
- $RBAB((\texttt{not}\ x), V') = \exists x'(V' \wedge \overline{x}')$,
- $RBAB((\texttt{increase}\ (\texttt{reward})\ r), V') = V' + r$,
- $RBAB((\texttt{decrease}\ (\texttt{reward})\ r), V') = V' - r$,
- $RBAB((\texttt{when}\ c\ e), V') = ITE(c, \quad RBAB(e, V'), \quad V')$,
- $RBAB((\texttt{prob}\ p_1\ e_1 \ldots p_\ell\ e_\ell), V') = \sum_{i=1}^{\ell} p_i \times RBAB(e_i, V')$,
- $RBAB((\texttt{and}), V') = V'$,
- $RBAB((\texttt{and}\ e_1\ e_2 \ldots e_\ell), V') = RBAB((\texttt{and}\ e_2\ \ldots\ e_\ell), RBAB(e_1, V'))$.

Example 5.9 *Figure 5.2 details the computation of RBAB for the next-state value V depicted on the left and the PPDDL effect*

$$e = (\texttt{and}\ (\texttt{not}\ x_1)\ (\texttt{when}\ x_2\ (\texttt{prob}\ 0.6\ (\texttt{not}\ x_2)\ 0.4\ x_3)))$$

Functions are represented as algebraic decision diagrams, which is indeed the data structure used by our implementation of RBAB (we take some liberty with the graphical representation, like duplicating leaves, to improve readbility).

RBAB first handles $(\texttt{not}\ x_1)$ *(as the first argument of* `and`*), hence computing the F-value of* $(\texttt{not}\ x_1)$ *with respect to V, denoted by F. Then RBAB handles the second argument of*

`and`, *namely* e_w = (`when` x_2 ...) with respect to F. *This first involves a recursive call on* e_p = (`prob` 0.6 (`not` x_2) 0.4 x_3). *In the end, the F-value of* e *is the F-value of the second argument of* `and` *with respect to the F-value of its first argument, namely,* $F_F^{e_w}$, *as depicted on the right of Figure 5.2.*

Proposition 5.10 (Lesner and Zanuttini 2011, Theorem 1) *Let* a *be an action represented as a PPDDL effect* e*, and let* V *be a value function. Then the backup rule* $RBAB(e, V')$ *computes* F_V^e.

Hence RBAB correctly computes the F-value of an action a with expected next-state value V. However, so as to plug these "backups" into a classical value iteration scheme, we need to have Q-values, not F-values, so as to choose the best action in each state and determine the resulting value function. Using Proposition 5.6, we are anyway able to modify RBAB to return the Q-value of a with expected value V, by letting its last step return $Q = \lambda s.F(s, s)$ instead of F.

The resulting, complete algorithm is given on Figure 2 for the γ-discounted, h-horizon cumulative criterion (the algorithm returns an optimal h-steps-to-go policy). A more general version, including action preconditions and the infinite-horizon criterion, is given in the article (Lesner and Zanuttini, 2011).

Algorithm 2: Rule-Based Action Backup (RBAB)

$V' \leftarrow \lambda s.0$;
for $i = 1, 2, \ldots, h$ **do**
  $V_{current} \leftarrow -\infty$;
  foreach *action* a **do**
    $e \leftarrow$ PPDDL description of a;
    $F_V^a \leftarrow RBAB(e, V')$;
    $Q_V^a \leftarrow \lambda s.F_V^a(s, s)$;
    $V_{current} \leftarrow \max(Q_V^a, V_{current})$;
  end
  $V' \leftarrow \gamma V'_{current}$;
end
$\pi \leftarrow \lambda s.\text{argmax}_{a \in A} Q_V^a(s)$;
return π;

5.4.3 Implementation and Experiments

The article (Lesner and Zanuttini, 2011) presents with some details the data structures used and some optimizations. The basic idea is to represent functions of variables (in unprimed and primed versions) as *Algebraic Decision Diagrams* (ADDs), a generalization of Binary Decision Diagrams to real-valued functions of Boolean descriptors (Bahar et al., 1997), as we did on Figure 5.2. ADDs come with a variety of operations implemented with very efficient algorithms, as well as with very efficient implementations, and they came out as a representation of choice for Q-values and value functions (Hoey et al., 1999).

The article also reports on some experiments on benchmarks from the International Planning Competitions (probabilistic tracks of the 5th and 6th IPC). The main conclusions which can be drawn from these experiments is that value iteration using RBAB is much faster than variants of SPUDD on some benchmarks, but that the converse also holds for other benchmarks.

Hence this algorithm must be seen as complementary to DBN approaches. Finding the exact indicators of when it is much faster is still an open question, however from the benchmarks used and some preliminary experiments we can say that RBAB is better than SPUDD when

- the PPDDL description of actions is a tree of greater depth, or
- there are many conditions which combine together (i.e., there are many aspects for actions),

- the PPDDL description is smaller (for the same problem).

These conclusions are quite intuitive, since they all merely say that RBAB is able to exploit the PPDDL factorization, and that it is better than SPUDD when the PPDDL representation differs greatly from a DBN representation.

Open Question 1 *Find precise (and efficiently computable) indicators of when RBAB is likely to be very efficient on a given instance.*

5.5 Revising Policies using F-Values

We have shown in Section 5.4 how the notion of frameless action-values may be used for designing a value iteration-like algorithm which proves to be complementary to other approaches. We now propose another application of this notion, to the problem of revising policies.

By *revising a policy*, we mean the general process of computing a policy for an MDP M', given a policy π for an MDP M (seen as a "previous version" of M') and information about how M' differs from M. Here, what we want is to compute an *optimal policy* for M', starting from an optimal policy π for M, under some criterion. Clearly, what we want is computation methods which avoid computing such a policy from scratch, but rather take advantage of the previously computed π. In the following, we will refer to M' as the *revised MDP*, and to the policy which we want to compute as the *revised policy*.

There are two main kinds of revisions from M to M' which are worth considering. The first, obvious one, is when the reward function R is revised, without the action model (i.e., the transition function T) changing. This kind of revisions happens wen an agent has a fixed model of actions, but when its missions, as formalized by the reward function R, change from time to time. It also happens in reinforcement learning problems, when the model of actions T is known (or has been already learnt) but the reward function R has to be discovered through experience.

The second kind of revisions, which we specifically address here, is when the transition function T is revised, with or without the reward function R changing. This typically happens when the agent updates its action model from time to time, especially when it is learning it while acting, as in reinforcement learning scenarios.

5.5.1 One-Step Revisions

We show in the following that an agent which stores the F-values of its actions in an MDP M (instead of simply a policy π or Q-values Q^a_V) is able to perform some forms of policy revision very efficiently. We stay in the framework of Section 5.4 and consider revisions of the PPDDL description of M.

The kind of revisions which we consider are *one-step* revisions. Precisely, given a policy π for an MDP M, the *one-step revision problem* takes as input a new version a' of some action a in M, and asks for the expected value of a' when π is used from the next-state on, that is, exactly $Q^{a'}_{(\gamma V_\pi)}$ where V_π is the expected value function of policy π (from the next-state on). We will see at the end of this section why this problem occurs in control and in certain approaches to reinforcement learning.

Example 5.11 (continued) *Given the PPDDL action of Example 5.9*

$$e = (\texttt{and}\ (\texttt{not}\ x_1)\ (\texttt{when}\ x_2\ (\texttt{prob}\ 0.6\ (\texttt{not}\ x_2)\ 0.4\ x_3)))$$

and its F-value with respect to the value function V, as depicted on Figure 5.2, revising the relative probabilities of x_2, x_3 to $0.8/0.2$ instead of $0.6/0.4$ means computing the F-value of the action

$$e' = (\texttt{and}\ (\texttt{not}\ x_1)\ (\texttt{when}\ x_2\ (\texttt{prob}\ 0.8\ (\texttt{not}\ x_2)\ 0.2\ x_3)))$$

with respect to the same next-state value function V.

5.5.2 Adding an Effect

The first, simplest case is when the description of some action is extended with a new, compatible effect. An example of this is when an agent has a model of this action a, but suddenly "notices" that under some condition c, a has an effect on some variable x which the agent believed to be unaffected. For instance, when a car driver suddenly notices that its rear light switches off when the radio is on (say, because of insufficient battery). In PPDDL terms, this corresponds to adding an effect (`when` rear-light (`not` rear-light)) to the action of turning the radio on.

Proposition 5.12 (adding an effect) *Let a be an action described by a PPDDL effect e, and V be a value function. Let e' be a PPDDL effect such that the PPDDL description $a' = (\texttt{and}\ e\ e')$ is consistent. Then given the F-value $F = F^a_V$, the F-value $F' = F^{a'}_V$ can be computed by*

$$F' = F^{e'}_F$$

Example 5.13 (continued) *Assume the action of Example 5.9 is revised by adding the effect* $(\texttt{when}\ \overline{x}_2\ (\texttt{and}\ x_2\ x_3))$*, that is, to*

$$e' = (\texttt{and}\ (\texttt{not}\ x_1)\ (\texttt{when}\ x_2\ (\texttt{prob}\ 0.6\ (\texttt{not}\ x_2)\ 0.4\ x_3))\ (\texttt{when}\ \overline{x}_2\ (\texttt{and}\ x_2\ x_3)))$$

Then revising the F-value can be done by computing the F-value of $(\texttt{when}\ \overline{x}_2\ (\texttt{and}\ x_2\ x_3))$ *with respect to the F-value computed in Example 5.9 (rightmost ADD on Figure 5.2), yielding the revision of the left child of the root to a single leaf with value* 10*.*

As a special case, a one-step revision of the cost or reward of an action is easy to do when this cost/reward does not depend on the next-state s'. This is because revising the reward r of a to $r' \neq r$ in all states satisfying some formula c amounts to add the PPDDL effect $(\texttt{when}\ c\ (\texttt{increase}\ (\texttt{reward})\ (r' - r)))$ (if $r' > r$) or $(\texttt{when}\ c\ (\texttt{decrease}\ (\texttt{reward})\ (r - r')))$ (otherwise) to the description of a.

This result however is not surprising at all. Indeed, because we consider rewards not depending on the next-state, exactly the same construction can be used to revise Q-values.

5.5.3 Revising Probabilities

We now consider the case when the probabilities of some effects are revised, namely, when an effect of the form $(\texttt{prob}\ p\ e\ldots)$ is revised into $(\texttt{prob}\ p'\ e\ldots)$. We restrict ourselves to the case where this clause occurs at the highest level in the description of the action, namely, when a is of the form

$$a = (\texttt{and}\ e_1\ \ldots e_\ell\ (\texttt{when}\ c\ (\texttt{prob}\ p\ e\ldots)))$$

(wlog, since `and` is commutative). We also assume that e is a conjunction of simple effects (literals), as in PSO representations. However, we place no restriction on other e_i's.

For clarity, we start with the case when there is only one probabilistic effect, namely, when we want to revise

$$a = (\texttt{and}\ e_1 \ldots e_\ell\ (\texttt{when}\ c\ (\texttt{prob}\ p\ e)))$$

to

$$a' = (\texttt{and}\ e_1 \ldots e_\ell\ (\texttt{when}\ c\ (\texttt{prob}\ p'\ e)))$$

The next proposition shows that again, revision can be performed with the sole knowledge of F^a_V. Observe that this is a special case of Proposition 5.16 (with $e_2 = \emptyset$ there), but the intuition in this particular case is that revising p to p' amounts to add the effect $(\texttt{when}\ \phi\ (\texttt{prob}\ \delta\ e))$ to a, because this means for e *not* to occur with probability $(1-p)(1-\delta) = (1-p')$, as desired (Proposition 5.16 shows correctness for negative δ).

Proposition 5.14 *Let*

$$a = (\texttt{and}\ e_1 \ldots e_\ell\ (\texttt{when}\ c\ (\texttt{prob}\ p\ e)))$$
$$a' = (\texttt{and}\ e_1 \ldots e_\ell\ (\texttt{when}\ c\ (\texttt{prob}\ p'\ e)))$$

where e is a conjunction of simple effects and $p \neq 1$, $p' \neq p$ hold. Let V be a value function. Then $F_V^{a'}$ can be computed from $F = F_V^a$ as

$$ITE(c, \quad \delta \times F^e_{F \wedge c} + (1 - \delta) \times (F \wedge c), \quad F)$$

with $\delta = 1 - \frac{1-p'}{1-p}$.

Example 5.15 *Let $a = (\texttt{prob}\ 0.5\ x_1)$ (that is, $(\texttt{and}\ (\texttt{when}\ c\ (\texttt{prob}\ 0.5\ x_1)))$ with tautological c), and let $V(x_1, x_2) = x_1 + x_2$. Then $F = F_V^a$ is given by*

$$\begin{aligned} F(x_1x_2, x_1'x_2') &= 0.5 \times V'(1, x_2') + 0.5 \times V'(x_1', x_2') \\ &= 0.5 \times (1 + x_2') + 0.5 \times (x_1' + x_2') \\ &= 0.5 + 0.5x_1' + x_2' \end{aligned}$$

Now let $a' = (\texttt{prob}\ 0.9\ x_1)$, i.e., let a be revised by augmenting the probability of effect x_1 from 0.5 to 0.9 ($\delta = 1 - 0.1/0.5 = 0.8$, that is, $(1 - 0.9) = (1 - 0.5) \times (1 - 0.8)$). Then $F_V^{a'}$ is given by $0.8 \times F_F^{x_1} + 0.2 \times F$, that is

$$\begin{aligned} F_V^{a'}(x_1x_2, x_1'x_2') &= 0.8 \times (1 + x_2') + 0.2 \times (0.5 + 0.5x_1' + x_2') \\ &= 0.9 + 0.1x_1' + x_2' \end{aligned}$$

which can be easily checked to be correct.

We now turn to the more complex case where a clause of the form $(\texttt{prob}\ p\ e_1\ (1-p)\ e_2)$ is revised. Again, we assume that it occurs at the highest level of a and that e_1, e_2 are conjunctions of simple effects. Moreover, we assume that e_1, e_2 are mutually consistent (contain no opposite literals).

So assume p is revised to p'. We cannot simply "add" new clauses, say $(\texttt{prob}\ \delta_{e_1}\ e_1)$ and $(\texttt{prob}\ \delta_{e_2}\ e_2)$ as in the case of a single outcome, since this would incorrectly introduce possible outcomes with e_1 and e_2 both occurring. Once again however, it turns out that the F-value of a can be revised without recomputing it from scratch.

Proposition 5.16 (Lesner and Zanuttini 2011, Proposition 2) *Let*

$$a = (\texttt{and}\ e\ (\texttt{when}\ c\ (\texttt{prob}\ p\ e_1\ (1-p)\ e_2)))$$
$$a' = (\texttt{and}\ e\ (\texttt{when}\ c\ (\texttt{prob}\ p'\ e_1\ (1-p')\ e_2)))$$

where e_1, e_2 are mutually consistent conjunctions of simple effects and $p, 1-p \neq 0$ and $p' \neq p$ hold. Let V be a value function. Then $F_V^{a'}$ can be computed from $F = F_V^a$ as

$$ITE(c, \quad \frac{p'}{p} \times \exists Vars(e_1')F_1 \ + \ \frac{1-p'}{1-p} \times \exists Vars(e_2')F_2 \ - \ \alpha \times \exists Vars(e_1' \wedge e_2')F_{12}, \quad F)$$

with $\alpha = \frac{p'(1-p)}{p} + \frac{p(1-p')}{1-p}$, e_1', e_2' the primed versions of e_1, e_2, $F_1 = F \wedge c \wedge e_1'$, and similarly for F_2, F_{12}.

Example 5.17 (continued) *Assume as in Example 5.11 that the action of Example 5.9 is revised to*

$$e' = (\texttt{and}\ (\texttt{not}\ x_1)\ (\texttt{when}\ x_2\ (\texttt{prob}\ 0.8\ (\texttt{not}\ x_2)\ 0.2\ x_3)))$$

that is, the probabilities of $(\texttt{not}\ x_2), x_3$ in the when *clause are revised to 0.8/0.2. Then revising its F-value can be done starting from the previously computed F-value (Figure 5.2), as depicted on Figure 5.3 (there, only the right branch of the root node, corresponding to $c = x_2$, is depicted).*

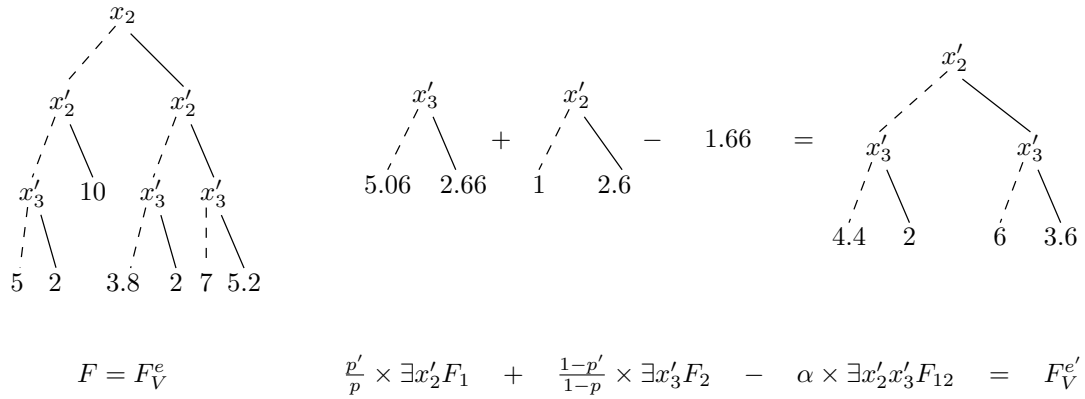


Figure 5.3: Revising probabilities of effects (continued from Figure 5.2)

When there are more than two effects or effects are not consistent together, it can be shown that probabilities can still be revised from F-values alone, albeit with a potential combinatorial explosion (depending on how effects interact). We do not elaborate on these extensions here.

It can also be easily seen that this contruction extends to any *deterministic* effect e_1 (possibly with nesting), albeit with corrective terms proportional to the associated reward (which is 0 for conjunctions of simple effects).

Open Question 2 *Are there efficient methods for revising probabilities when more than two effects and/or mutually inconsistent effects are considered?*

Finally, we wish to emphasize that such revision of probabilities cannot be done using Q-values (the information embedded in F-values is necessary).

Example 5.18 *Consider the action* $a = (\texttt{when}\ x\ (\texttt{not}\ x))$ *and the value function* V *defined by* $V(1) = 0, V(0) = 1$. *It is easily seen that the Q-value of* a *with respect to* V *satisfies* $Q_V^a(1) = Q_V^a(0) = 1$. *Hence when revising* a *to* $a' = (\texttt{and}\ (\texttt{when}\ x\ (\texttt{not}\ x))\ (\texttt{when}\ \overline{x}\ x))$, *for instance, the Q-value does not distinguish between the "reality"* $V(1) = 0$ *and, say,* $V(1) = 10$, *whereas revising* a *to* a' *should yield* $Q_V^{a'}(0) = 0$ *in the former case, but* $Q_V^{a'}(0) = 10$ *in the latter.*

5.5.4 The Impossibility to Remove Deterministic Effects

We end this section by observing that, even using F-values, it is impossible to revise an action by removing a deterministic effect from its description, that is, to revise $a = (\texttt{and}\ e_1\ e_2)$ to $a' = e_1$, if e_2 is deterministic. In essence, this is because when e_2 is deterministic, the expected value of situations where e_2 does not occur is per definition not relevant to the F-value of a. Of course, such revision is *a fortiori* impossible using Q-values.

Technically, this shows that the requirement $p, 1-p \neq 0$ in Proposition 5.16 is not a coincidence due to these quantities occurring as denominators, but is unavoidable.

Example 5.19 (continued) *Consider again the example on Figure 5.2. Because of the deterministic effect* $(\texttt{not}\ x_1)$, *the value of* V *on* x_1 *(right child of the root on the leftmost diagram) is lost, whereas, obviously, it is relevant to the F-value of the action*

$$e' = (\texttt{when}\ x_2\ (\texttt{prob}\ 0.6\ (\texttt{not}\ x_2)\ 0.4\ x_3))$$

that is, the action obtained when revising e *by removing its first (deterministic) effect.*

5.6 Perspective: Improvements of RBAB and First-Order

We have performed proof-of-concept experiments showing that our algorithm RBAB, for computing optimal policies of MDPs represented in grounded PPDDL, clearly outperforms its main concurrent SPUDD (Hoey et al., 1999), in two of its versions, on several benchmarks (Lesner and Zanuttini, 2011); the converse holds on other benchmarks. This study still has to be confirmed by a wider range of experiments, and by experiments designed to make precise the characteristics of problems which make RBAB more efficient (or less efficient) than other approaches.

A perspective is to try to make RBAB more efficient in general, especially by incorporating ideas which have proven useful for other approaches. Maybe the first of these is to make RBAB use *Affine Algebraic Decision Diagrams* (AADDs, Sanner and McAllester 2005), which is a data structure particularly well-suited for representing functions such as value, action-value, and frameless action-value functions (especially, because it proposes compact representations of mutiplication and addition of scalars, operations which are used when combining probabilistic effects and incorporating rewards, respectively). Another idea is to precompile certain parts of the transition matrix, just as was proposed for SPUDD (Hoey et al., 1999), so as to avoid performing certain computations again and again.

It would also be interesting to incorporate the ideas underlying RBAB to other algorithms than Value Iteration. In the exact case, it would be interesting to incorporate these ideas in Policy Iteration and in Prioritized Sweeping. A further direction is to explore ways of recognizing in PPDDL descriptions what the "most important" effects are, and compute approximate policies focusing only on these effects. The incremental flavor of our rules would possibly lead to an *anytime* approach, in which the effects of actions would be incorporated one by one, starting with the most important ones, as long as time permits.

Finally, an obvious perspective of this work is to try to generalize the algorithm so that it can handle (a subset of) PPDDL directly at the first-order level. Despite the inherent difficulty of such first-order descriptions (e.g., universally quantified goals), several approaches have been proposed, in different frameworks (Kersting et al., 2004; Sanner and Boutilier, 2009; Lang and Toussaint, 2010). Moreover, some theoretical work already done opens interesting perspectives for lifting RBAB to first-order descriptions. In particular, as concerns representations of value functions (or F-values), generalizations of algebraic decision diagrams to the first order have been proposed, and their usefulness to decision-making demonstrated (Wang et al., 2008).

5.7 Perspective: Further Applications

Apart from the mere computation of optimal policies, the incremental flavor of our backup rules open perspectives on various problems. We discuss two of them.

A first problem arises in multiagent (cooperative) systems, where decisions have to be computed respective to optimal *joint actions*. Using our approach, and under some conditions (consistency of actions), it would be possible to partially distribute the computation of Q-values for joint actions, by letting the first agent compute the F-values $F_1^a = F_V^a$ for each of its actions a, respective to the next-step value function V, then let the second agent compute the F-values $F_2^a = F_{F_1^a}^a$ for each of its actions a respective to the F_1^a's, etc. By construction, this process would allow the computation of the values of joint actions. Such a process would not only be distributed, it would also respect privacy in some sense, since each agent would be able to participate in the computation without revealing its own action descriptions (at least, explicitly).

Another perspective concerns the automatic detection of conflicts in the specification of an action, or between actions of different agents. Let the backup rules on deterministic simple effects be modified to $RBAB(x, V') = V' \wedge x'$ and $RBAB((\texttt{not } x), V') = V' \wedge \overline{x}'$, that is, by making deterministic effects explicit in F-values (removing existential quantification). Then it can be seen that two conflicting action specifications, that is, for which two mutually inconsistent effects may occur together, would show up through inconsistent F-values, precisely through branches restricted to both conditions x' and $\overline{x}'$. Automatically detecting such conflicts in joint actions may be useful

in particular for computer-aided design of multiagent systems.

5.8 Perspective: Using F-values For Control

We now detail a perspective of F-values, for control in the presence of exogenous but short-term predictable events.

For instance, assume that you know the probability for rain to fall (variable x_{rain}) on each day, but you also have the opportunity, at execution time, to know whether or not it will rain in the next state (by watching the weather forecast, say). Then embarking F-values of actions instead of policies allows you to compute the optimal decision, at execution time, given the additional information that it will rain (or not rain) at the next step. Indeed, incorporating this information amounts to select only the branches of F-values which are consistent with this information on the (next-state) variable x'_{rain}, and only then compute Q-values from F-values. Using revision techniques such as those presented in Section 5.5, it is even possible to take into account *uncertain* weather forecast, by revising the default probability of rain to the one given by the forecast, at execution time, for the next-state.

Observe however that embarking F-values (for all actions) instead of simply policies comes at the cost of extra space requirements. We also wish to point out that if the forecast is assumed to be "deterministic", other techniques may make usual Q-values or even simple policies sufficient for control, namely, techniques consisting in adding propositional atoms $K(rain')$ and $K(\overline{rain}')$ to each state, with the obvious epistemic interpretation (but handled as simple propositions). Then usual policies will be conditioned on the forecast. Nevertheless, if there are several such short-term predictable exogenous events, this approach may yield a combinatorial explosion for the state space. Moreover, contrary to our approach, such techniques are unlikely to be able to take uncertain forecasts into account. Hence we think that the approach using F-values is more appropriate (and more elegant).

5.9 Perspective: Using Revision for Reinforcement Learning

We end this chapter with another perspective of F-values, in the setting of reinforcement learning (RL). Our proposal builds on the algorithms RTDP-Rmax and RTDP-MBIE (Strehl et al., 2006), which are variants of the well-known algorithms R-max (Brafman and Tennenholtz, 2002) and MBIE (Strehl and Littman, 2008) (MBIE stands for "Model-Based Interval Estimation").

In a nutshell, R-max and MBIE are provably efficient algorithms for RL, which maintain an estimate of the transition model of the underlying MDP (they constitute *model-based* approaches to RL). From this model, at each timestep t they derive another MDP M_t, the optimal policies of which are policies allowing for efficient exploitation and exploration. A major drawback of these approaches is that the MDP M_t is revised very often, in fact, each time new information is collected about the underlying MDP. Hence, computing its optimal policy, which is a costly operation, has to be performed very often.

What RTDP-Rmax and RTDP-MBIE use instead is a simple *revision* of the MDP M_t at each timestep, which turns out to be very similar to the *one-step revision* which we studied in Section 5.5. Precisely, let a be an action, and write R_t, T_t for the reward and transition functions in M_t, V_t for its value function, and R_{t+1}, T_{t+1} for the reward and transition functions as modified by incorporating information about a as collected at timestep $t+1$. Then instead of computing an optimal policy for the new MDP M_{t+1} defined by R_{t+1}, T_{t+1}, RTDP-Rmax simply revises the Q-value of action a by computing for all states s

$$Q_{t+1}(s,a) = R_{t+1}(s,a) + \gamma \sum_{s' \in S} T_{t+1}(s'|s,a) V_t(s')$$

RTDP-MBIE is simply a similarly modified version of MBIE.

RTDP-Rmax and RTDP-MBIE are proposed for flat state spaces, that is, Q-values and value functions are stored in a lookup table. However, as the authors note, factored representations of these functions would not impact the convergence proofs. Hence we propose a study of factored versions of these algorithms, focusing on the revision step above, which is the main step of the algorithm.

In the revision step defined above, $V_t(s')$ is per definition the maximum of $Q_t(s,a)$ over all actions a. What we propose first is to always keep the Q-value estimates $Q_t(s,a)$ up-to-date with V_t. For this it is enough to perform a new (factored) backup each time an action has its Q-value estimate updated. Precisely, when $Q_t(s,a)$ is updated to $Q_{t+1}(s,a)$, we can compute the resulting modification in the value function estimate as $\delta V(s) = V_{t+1}(s) - V_t(s) = \max(0, Q_{t+1}(s,a))$. This simply expresses the fact that the value function changes exactly in those states where a now appears to be better than the previously best action. Then the Q-value estimates of all actions can be updated by $Q_t(a) \leftarrow Q_t(a) + Q^a_{\delta V}$ (by linearity of the backup operator with respect to the next-step value function). Obviously, such an update would be performed as only one, factored backup.

What is important in this construction is that typically, the revision of an action in one step will have little impact on the resulting value function estimate V, that is, δV is typically 0 except for a small number of states. Hence $Q^a_{\delta V}$ is typically easily computed.

With the Q-value estimates now always up-to-date with the current value function estimate V_t, we can rewrite the revision step of RTDP-Rmax as follows:

$$\begin{array}{lllllll} \text{compute} & Q_{t+1}(s,a) & = & R_{t+1}(s,a) & + & \gamma \sum_{s' \in S} & T_{t+1}(s'|s,a) \quad V_t(s') \\ \text{given} & Q_t(s,a) & = & R_t(s,a) & + & \gamma \sum_{s' \in S} & T_t(s'|s,a) \quad V_t(s') \end{array}$$

This is exactly a one-step revision (of T_t and R_t), as studied in Section 5.5.

Obviously, the computation of V_t in the revision step is not avoided. Where Strehl et al. (2006) recompute V_t when revising a Q-value function estimate *which depends on* V_t, we propose to keep the Q-value estimates up-to-date with V_t in an incremental way. Though this may cause some extra computations (which, as we argued, should be very efficient), as a side effect this may have the benefit of making up-to-date estimates always available, hence exploration and exploitation more effective.

To conclude, we believe that our approach to revision has a potential impact in incremental reinforcement learning algorithms such as RTPD-Rmax (the reasoning would be similar for RTDP-MBIE). This still has to be confirmed, especially by experiments for measuring computation time, but using the ideas developed here, we think that lightweight (in terms of computation), and nevertheless provably efficient, approaches to reinforcement learning with factored models of actions can be designed.

Chapter 6

Learning Compact Models of Actions

Contents

This chapter explores issues pertaining to learning compact action models, namely, representations of actions as Probabilistic STRIPS Operators (PSOs). In particular, we do not assume that the effects of actions on various variables are probabilistically independent from each other. This departs this work from the mainstream, which assumes such independency. Noteworthy exceptions are in the field of *relational* (reinforcement) learning (Džeroski et al., 2001), where authors typically try to design algorithms able to learn relational rules for describing the actions, and such rules indeed take correlated effects into account (Safaei and Ghassem-Sani, 2007; Pasula et al., 2007; Rodrigues et al., 2010). Nevertheless, such works face other, often much harder problems due to the relational setting itself, and they give *heuristic* approaches.

What we propose here is a theoretical study of action learning with correlated effects in the propositional setting, thus avoiding the difficulties of first-order semantics. Nevertheless, as we will see, even in this simpler setting many difficulties arise, raising important open problems.

The typical setting in which action models have to be learnt is that of reinforcement learning. Though we will always have this setting in mind, the study presented here focuses on learning an

action by observing the transitions which it provokes in different states, without assuming a particular exploration strategy for choosing the most informative states. We discuss our perspectives in adapting this work to the reinforcement learning setting at the end of this chapter.

The results and ideas presented here are under ongoing development in collaboration with Boris Lesner, in the context of his PhD. Because this work has not been published so far, we give a detailed analysis and full proofs, but most of them are simple. Relevant preliminaries can be found in Sections 2.4 (Page 15) and 3.2 (Page 25).

6.1 Motivation

Assume that you observe your neighbours from behind your window, that you know that all of them play the National Lottery each day, and that you want to estimate the probability of winning the lottery from your observations. Assume moreover that most of your neighbours, say, 90 %, have a huge villa and a very expensive car parked in front of it, while the remaining 10 % have no such visible sign of richness (call them "poor").

From time to time, you may observe one of your poor neighbours suddenly having a very expensive car parked in front of his house. Say this happens for 5 % of them over one year. Then the most likely explanation is that they won the lottery, while the other 95 % did not. What about your observations of your "rich" neighbours? You will never observe them moving from a situation *without* an expensive car to one *with* an expensive car, since they all already have one. There are a lot of explanations for this:

- maybe none of them ever won the lottery,
- maybe all of them won the Lottery,
- maybe 5 % of them won the Lottery, and the others did not,
- in general, maybe a fraction p of them won the Lottery, and the others did not, for any $p \in [0, 1]$.

From these observations you can learn the most likely *conditional* probability of winning the lottery (during a year), which is 5 % for "poor" player, and p % for "rich" player, with unknown p.

Is this model accurate? Say that you want to estimate the probability of winning because you would like to buy a very expensive car. If you are poor, then you can balance the 5 % chance of winning with the prize to be won, the price of the car, etc., and finally make the decision to play or not to play accordingly. If you are rich, then you do not have an estimate of your chance to win, but anyway, winning or not will not affect your situation and hence, will not affect the value of the decision to buy the car; after all, may you win or not, you will be rich before and after playing. Hence, your decision to play or not to play can be made optimally with this model, and in this sense the model is accurate.

Now assume that you do not have enough time to observe all your neighbours each day, and you can only afford the time to observe the closest ones. Assume 2 among them are poor, and 18 are rich. Then the statistics gathered over one year will make you infer that the probability of winning is 0, 1/2, or 1 when you are poor, and again some unknown $p \in [0, 1]$ when you are rich.

Clearly, you do not want to trust the statistics on poor people, because your sample is too small. But if you know that the real probability of winning the lottery does not depend on the player being rich or poor, you would like to gather together the statistics on your poor and rich neighbours so as to estimate this unique probability on a 20-person sample, which would be more reliable. The point is that even with this hypothesis, the most likely explanation is of the form $p = (p_p \times 2 + p_r \times 18)/20$, where p_p is the *known* fraction of your poor neighbours who won, but p_r is the *unknown* fraction of your rich neighbours who won. And this time, if you are poor then the real value of p_r will make a difference on your decision to play or not to play, because you will only trust p to be a good estimate of the probability of winning (instead of the known p_p, which you cannot trust to be accurate).

The following sections are devoted to proposing approaches for dealing with this phenomenon, to which we will refer as the *ambiguity* of the effect "winning the lottery" in condition "being rich".

6.2 Related Work

The phenomenon of ambiguous effects has already been noted by several authors, for instance by Safaei and Ghassem-Sani (2007) and Walsh et al. (2009) for grounded settings. It is also implicitely handled by authors interested in learning *relational* models of actions, for instance, by (Pasula et al., 2007). Nevertheless, as far as we know this phenomenon has never been thoroughly addressed. In particular, works about learning relational models propose *heuristic* approaches, with no guarantee on how ambiguities are resolved.

In the propositional setting, there have been a certain number of works concerned with reinforcement learning for DBN representations of actions. The algorithm DBN-E^3 (Kearns and Koller, 1999) learns the DBN parameters (probabilities), provided the set of parents of each variable is known in advance. Later, Degris et al. (2006) removed this hypothesis, by proposing a general approach based on induction of decision trees for finding relevant parents and associated probabilities. Their algorithm (SPITI) is able to handle very large MDPs, but provides no guarantee on the accurary of the model learnt. Strehl et al. (2007) then proposed an algorithm (SLF-Rmax) able to learn both the structure and the probabilities, with formal guarantees, with a sample complexity essentially of the order of $\binom{n}{k}$ where k is an upper bound on the number of parents of each variable (this upper bound must be known in advance). Quite interestingly, their approach allows to automatically uncover the k variables involved in the conditions of each action (the $\binom{n}{k}$ term in the sample complexity comes from the number of such candidate conditions).

Ambiguity is not a concern for approaches which learn DBNs. This is because there are at most $2n$ effects of the form x or $\overline{x}$, of size 1 each. Hence ambiguity can be avoided by learning a distribution for the states satisfying x, and one for the states satisfying $\overline{x}$. To continue with the Lottery example, the algorithms can afford the time to observe sufficiently many poor and sufficiently many rich neighbours, so as to have accurate statistics for each class. However, all approaches mentioned above must restrict to DBNs *without synchronic arcs*. Indeed, in the case of SLF-Rmax for instance, even if x has at most k parents among the unprimed variables, if it is part (and not probabilistically independent of) an effect containing k' variables, then it has up to $k + k' - 1$ parents due to the synchronic arcs.

Hence, if the length and/or number of effects is unbounded, we need to resort to other approaches. Noticeably, Walsh et al. (2009) have proposed KWIK learners for effect distributions. Nevertheless, they need to know in advance the effects which can occur, and they leave open the problem of learning them.

6.3 Our Contribution

This chapter presents an in-depth study of the phenomenon of ambiguous effects. After giving a precise formalization of the problem, we study an unbiased, maximum-likelihood approach to computing plausible effect distributions given observed transitions. As we will see, this approach does not allow one to compute a unique distribution, hence we have to resort to more sophisticated techniques. So we study an approach based on several sets of observations. Our approach gives a complete (but unsound in general) approach for recognizing when several such sets have been generated by the same effect distribution. However, we are also able to use this approach for identifying the exact set of effects which generated the observations, provided a limited form of active queries are available and the effects satisfy some natural closure property.

Because our setting is harder for the agent than reinforcement learning, and because we can identify the real effects which occurred, this work comes in complement to approaches to reinforcement learning which need the set of effects to be known in advance. Hence we draw interesting perspectives in this setting.

6.4 Formalization

We now place ourselves in the formal framework of actions represented by probabilistic STRIPS operators (PSOs). Hence, there is a hidden model of some action a, which we would like to learn accurately enough for making (almost) optimal decisions. In general, a has a PSO representation of the form $\{(c_i, D_i) \mid i = 1, \ldots, k\}$, where for all i, D_i is an effect distribution. We will consider this general setting, but in all sections except for the last two, we will focus on the simpler setting where there is only one condition, namely, the hidden action a can be represented by a unique effect distribution D, of the form $\{(e_i, p_i) \mid i = 1, \ldots, \ell\}$. For instance, in the Lottery example, the action has two effects, namely x_{rich} with probability 0.05, and the empty effect $\emptyset$ with probability 0.95.

6.4.1 Effect Intervals

For technical reasons, we will make the assumption that the probabilities of effects are rational numbers.

Definition 6.1 (target distribution) *We call* target distribution *a distribution $D = \{(e_i, p_i) \mid i = 1, \ldots, \ell\}$ where for $i = 1, \ldots, \ell$, $e_i \in 3^X$ is an effect and $p_i \in \mathbb{Q} \cap [0, 1]$ is a rational probability. We assume $\sum_{i=1}^{\ell} p_i = 1$.*

Ambiguity is a consequence of the following observation.

Proposition 6.2 (effects explaining some transition) *Let s, s' be two states. Then the effects $e \in 3^X$ which provoke a transition from s to s', that is, which satisfy $apply(s, e) = s'$, are exactly those for which $s' \setminus s \subseteq e \subseteq s'$ holds.*

Example 6.3 (effects for transition) *Let $s = 000$ and $s' = 010$. Then the effects[1] which provoke a transition from s to s' are x_2, $\overline{x}_1 x_2$, $x_2 \overline{x}_3$, and $\overline{x}_1 x_2 \overline{x}_3$. That is, we are sure that effect x_2 occurred, but any other literal in s' may have been set to true, or left true, by the action.*

Hence the effects which *may have occurred* when a transition from s to s' is observed form what we call an *effect interval* (*interval* for short), written $[s' \setminus s, s']$. Observe that for any effect e in this interval, we have $[s' \setminus s, s'] = [e \setminus s, apply(s, e)]$.

Effect intervals are much like set intervals. In particular, if $e_{lb}, e_{lb'}$ are two effects consistent together, the intersection of intervals $[e_{lb}, e_{ub}]$ and $[e'_{lb}, e'_{ub}]$ is exactly the interval $[e_{lb} \cup e'_{lb}, e_{ub} \cap e'_{ub}]$. Moreover, like with set intervals, an interval $[e_{lb}, e_{ub}]$ is nonempty if and only if $e_{lb} \subseteq e_{ub}$ holds. A consequence of this is that if $e_{lb}, e_{lb'}$ are not consistent together, the intersection of intervals $[e_{lb}, e_{ub}]$ and $[e'_{lb}, e'_{ub}]$ is empty, whatever the upper bounds e_{ub}, e'_{ub}. In the following, we write $[e_{lb}, e_{ub}] \cap [e'_{lb}, e'_{ub}] = [e_{lb} \cup e'_{lb}, e_{ub} \cap e'_{ub}]$ for the intersection of the effect intervals $[e_{lb}, e_{ub}]$ and $[e'_{lb}, e'_{ub}]$. Observe that the union or difference of two effect intervals is not in general an effect interval. We also write $[e]$ instead of $[e, e]$, and $|o|$ for the number of effects in an interval o (effect intervals will be denoted by $o, o_1, o_i \ldots$, standing for "observation").

Example 6.4 (continued) *The set of effects in Example 6.3 is denoted by $[x_2, \overline{x}_1 x_2 \overline{x}_3]$. Its intersection with $[\overline{x}_1, \overline{x}_1 x_2 \overline{x}_3]$ is $[\overline{x}_1 x_2, \overline{x}_1 x_2 \overline{x}_3]$, and its intersection with $[\overline{x}_1 x_2, \overline{x}_1 x_2 x_3]$ is $[\overline{x}_1 x_2]$. On the other hand, its intersection with $[x_1]$ is empty, because it is $[x_1 x_2, \overline{x}_1 x_2 \overline{x}_3]$ in which any effect should include x_1 because of the lower bound, whereas x_1 is not allowed by the upper bound.*

Most intuitions about the properties of effect intervals which we need here can be caught with (abstract) representations of these by intervals on the line. We will use such representations for most illustrations.

[1] For brevity, we write, e.g., $x_1 x_2 \overline{x}_3$ for the effect $x_1 \wedge x_2 \wedge \overline{x}_3$ in examples.

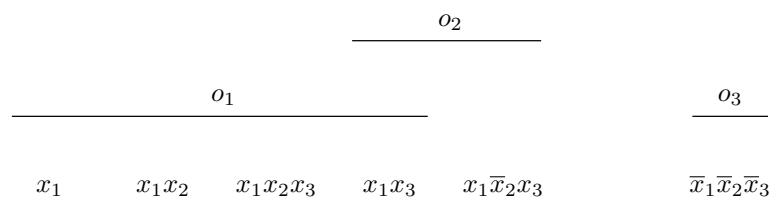


Figure 6.1: An example graphical representation of effect intervals

Example 6.5 (graphical representation) *Figure 6.1 represents the intervals* $o_1 = [x_1, x_1x_2x_3]$, $o_2 = [x_1x_3, x_1\overline{x}_2x_3]$, *and* $o_3 = [\overline{x}_1\overline{x}_2\overline{x}_3]$. *This representation makes explicit, for example, the fact that* o_1 *and* o_2 *intersect (on* x_1x_3*).*

Hence, when trying to learn an action model from observed transitions, the learner will not observe the effect which really occurred, but only be able to deduce an *interval* of effects which possibly occurred (called *explanations* for the corresponding transitions). This ambiguity is not a problem if each possible state can be observed sufficiently often so that the model is learnt independently for each state, as is the case for the lottery example when you observe many poor and many rich neighbours. Indeed, in this case, you are able to learn the action model for each state s, modulo the equivalence relation $\equiv_s$ defined by $e \equiv_s e' \Leftrightarrow \mathit{apply}(s, e) = \mathit{apply}(s, e')$. In the lottery example, even if you have observed sufficiently many rich people (state s), you are unable to learn precisely about their probability of winning (effect e) and that of losing (effect e'), but anyway they are in the same equivalence class, in the sense that winning or losing provokes the same transitions for rich people, namely, they remain rich ($\mathit{apply}(s, e) = \mathit{apply}(s, e') = s$). Clearly then, whatever effect or combination of effects you pick in each equivalence class, the induced transition function T from states to states will be the same, and hence your optimal decisions will be the same.

The problem arises in the framework of PSOs because we don't want a learner to require a sufficient number of sample transitions for each state. Typically, we want the learner to require no more than a number of sample transitions polynomial in the number of variables, conditions, and effects for the hidden action (see Section 6.2 for some examples with DBN representations).

Hence, approaches must be proposed which are able to deal with the ambiguity of effects, when *intervals of effects* are observed in a *small fraction* of the state space, and these observations have to be generalized to the *whole* state space. In the lottery example, this reads "wins and losses are ambiguous in observations of your rich neighbours, you only observe a small fraction of your poor neighbours, you know that the probability of winning is the same for your rich and poor neighbours, and you want to learn this probability".

6.4.2 Learning Problem

We now present our learning model, especially our assumptions about what is observed by the learner. As evoked in the introduction of this chapter, we assume that the learner has no control over the states from which transitions are observed. We even leave the possibility for these states to be chosen *adversarially* (i.e., so that the transitions are minimally informative). On the other hand, we require the environment to sample the transitions fairly to the hidden effect distribution.

Clearly, this setting is more adversarial to the learner than that of reinforcement learning. Indeed, in the latter the same assumption is made that transitions are sampled according to the hidden effect distribution, but in addition the learner has some control over the states in which it observes those transitions. In particular, it can make use of clever *exploration strategies*, for instance the ones in E^3 (Kearns and Singh, 2002) or in R-max (Brafman and Tennenholtz, 2002), or their instantiations for factored MDPs with independent effects, DBN-E^3 (Kearns and Koller, 1999) and SLF-Rmax (Strehl et al., 2007).

Our setting however naturally arises in many settings. For instance, a robot may be allowed to

train in a factory so as to learn accurate models of its actuators. Then the situations (states) in which it is allowed to train may be restricted due to the facilities in the factory. Another setting is that of reinforcement learning, if other agents or exogenous events may intervene at any moment. In this case, the agent may try to follow a chosen trajectory (using, say, its exploration strategy), but the current state may change at any moment due to modifications out of its control.

The following definitions make this setting precise, starting with what couples state/effect the environment is allowed to sample (but these remain hidden as such to the agent).

Definition 6.6 (sample states/effects) *A* multiset of sample states/effects *is any multiset of the form* $\mathcal{T} = \{(s_i, e_i) \mid i = 1, \ldots, m\}$*, where for* $i = 1, \ldots, m$*,* s_i *is a state and* e_i *is an effect.*

Definition 6.7 (induced distribution) *Let* $\mathcal{T} = \{(s_i, e_i) \mid i = 1, \ldots, m\}$ *be a multiset of sample states/effects. The* effect distribution induced by $\mathcal{T}$*, written* $D_{\mathcal{T}}$*, is the distribution* (e, p_e) *of effects occurring in* $\mathcal{T}$*, each associated with the probability* p_e *defined to be the proportion of indices* $i \in \{1, \ldots, m\}$ *such that* $e_i = e$*.*

Definition 6.8 (fairness) *Let* D *be an effect distribution, and let* $\mathcal{T} = \{(s_i, e_i) \mid i = 1, \ldots, m\}$ *be a multiset of sample states/effects. Then* $\mathcal{T}$ *is said to be* ϵ-fair to D *if for* $i = 1, \ldots, m$*,* e_i *has a nonzero probability in* D*, and* $||D_{\mathcal{T}} - D||_1 \leq \epsilon$ *holds.*

In words, a multiset of sample states/effects $\mathcal{T}$ is ϵ-fair to an effect distribution D for some "small" value of ϵ if sampling effects from D yields with some "high" probability $1 - \delta$ the distribution of effects in $\mathcal{T}$. The values of ϵ and δ are linked together by Weissman *et al.*'s bound (Weissman et al., 2003), which extends the well-known Chernoff's bound to multivalued random variables:

$$\delta = Pr(||D_{\mathcal{T}} - D||_1 \geq \epsilon) \leq (2^{\ell} - 2)\mathrm{e}^{-m\epsilon^2/2}$$

where m is the number of samples observed (size of $\mathcal{T}$), and ℓ is the number of effects (with nonzero probability) in D.

Observe that we use L1-distance to measure fairness, but that other distances could be used. Our use of L1-distance is motivated by the fact that approximating the transition probabilities $T(\cdot|s, a)$ in an MDP with respect to this distance provides guarantees on the resulting value functions. Observe that the infinite norm gives similar results (see, e.g., the "simulation lemma" by Kearns and Singh (2002)).

Example 6.9 (sample states/effects) *Let* $D = \{(x_1, 0.5), (x_1\overline{x}_2, 0.3), (\emptyset, 0.2)\}$*, and let* $\mathcal{T} = \{(01, x_1), (00, x_1), (00, \emptyset), (00, x_1\overline{x}_2), (01, x_1)\}$*. Then* $\mathcal{T}$ *is a multiset of sample states/effects in which, for instance, the first element corresponds to state* $s = 01$ *having been chosen, and effect* x_1 *having been sampled. The effect distribution induced by* $\mathcal{T}$ *is* $D_{\mathcal{T}} = \{(x_1, 0.6), (x_1\overline{x}_2, 0.2), (\emptyset, 0.2)\}$*, and hence* $\mathcal{T}$ *is* 0.2*-fair to* D*.*

With these definitions in hand, we now formalize what the agent really observes of the states and effects sampled by the environment. We directly formalize the observation of an effect e in state s using the effect interval $o = [e \setminus s, apply(s, e)]$, because it follows from Proposition 6.2 that this is the same as observing the *transition* from s to s' provoked by e, which is what is presented to the agent's sensors.

Definition 6.10 (induced observations) *Let* $\mathcal{T} = \{(s_i, e_i) \mid i = 1, \ldots, m\}$*, be a multiset of sample states/effects. We call* observation induced by (s_i, e_i) *the couple* $(s_i, [e_i \setminus s_i, apply(s_i, e_i)])$*, and* set of observations induced by $\mathcal{T}$ *the set* $\{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ *of observations induced by the elements of* $\mathcal{T}$*, where each couple* (s_i, o_i) *is associated with the proportion* f_i *of indices* $j = 1, \ldots, m$ *such that the observation induced by* (s_j, e_j) *is* (s_i, o_i)*.*

As should be clear from Proposition 6.2, the intervals which may be induced from a transition from a state s have a particular form. Hence when talking about a set of observations, we will mean a set of observations which have this particular form.

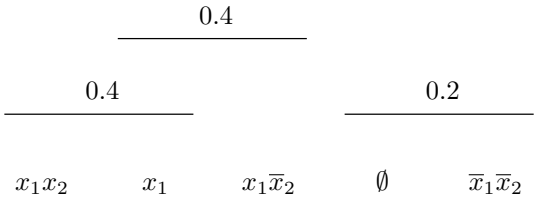


Figure 6.2: An example of induced observations

Definition 6.11 (possible observations) *A set* $\{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ *is said to be a* possible set of observations *if for all* i, s_i *is a state,* $o_i = [e_i \setminus s_i, apply(s_i, e_i)]$ *for some effect* $e_i \in 3^X$, f_i *is a rational number in* $[0, 1]$, *and in addition* $\sum_i f_i = 1$.

As we will see throughout this study, states s_i play a minor role in the learning process. It is thus useful to define the frequency of an observed interval irrespective of the states in which it was observed.

Definition 6.12 (frequency) *Let* $O = \{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ *be a possible set of observations. For each interval* o *occurring in* O, *the* frequency of o in O, *written* $f_{o,O}$ *or simply* f_o, *is defined to be* $\sum_{o_i=o} f_i$.

Definition 6.13 (fairness) *Let* D *be an effect distribution. A set of possible observations* O *is said to be* ϵ-fair to D *if there is a multiset of sample states/effects* $\mathcal{T}$ *such that* O *is induced by* $\mathcal{T}$ *and* $\mathcal{T}$ *is* ϵ*-fair to* D.

Example 6.14 (continued) *The set of observations induced by* $\mathcal{T}$ *in Example 6.9 is*

$$O = \{(01, [x_1, x_1x_2], 0.4), (00, [x_1, x_1\overline{x}_2], 0.2), (00, [\emptyset, \overline{x}_1\overline{x}_2], 0.2), (00, [x_1, x_1\overline{x}_2], 0.2)\}$$

The frequency of interval $[x_1, x_1\overline{x}_2]$ *is* $0.2+0.2 = 0.4$. *The intervals are depicted on Figure 6.2 with the associated frequencies. By construction,* O *is* 0.2*-fair to the distribution* D *in Example 6.9.*

On the other hand, the set of observations $\{(11, [x_1, x_1\overline{x}_2], 1)\}$ *is not a possible set of observations. This is because the effects in the interval do not provoke the same transitions in state* 11.

6.5 Likelihood of Distributions

We start our study with an unbiased maximum-likelihood approach for learning a hidden (PSO) action model from a set of observations. We restrict here to PSOs with only one condition, that is, we want to learn a unique, hidden effect distribution D. As we will see, there is always a "perfectly likely" effect distribution for explaining a given set of observed transitions. Unfortunately, the phenomenon of ambiguous effects implies that in general, there will be many such most likely distributions, which are indistinguishable from each other without an additional bias.

More generally, we propose a measure of how likely it is that *any given* effect distribution indeed generated a given set of observations. This will allow us to further add constraints under which there will not necessarily exist a "perfectly likely" distribution. Such constraints will especially show up in Section 6.6, where we will search for a distribution which explains *several* sets of observations at the same time.

We now propose our measure of likelihood for an effect distribution, given a set of observations. As we will show, this measure will coincide with fairness, and in fact will give a constructive manner of measuring fairness (of a given set of observations to a given distribution). Moreover, this construction will show that the observed *intervals* convey all the information, and that the

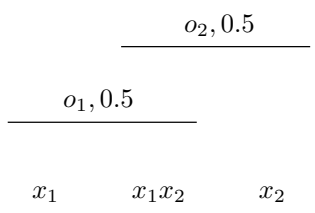


Figure 6.3: An example possible set of observations

states in which they were observed can be forgotten without any loss of information about the likely effect distributions. Observe however that this is natural, since the state s can be retrieved from an interval of the form $[e \setminus s, \mathit{apply}(s, e)]$.

6.5.1 Most Likely Distributions

We start with the most likely distributions. Our approach parallels the well-known maximum-likelihood approach for estimating a distribution of objects from their observed frequencies, but here the objects (effects) are not directly observed.

Proposition 6.15 *Let $O = \{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ be a possible set of observations. Then there is at least one effect distribution D to which O is 0-fair.*

Proof Let n be an integer such that $f_i n$ is an integer for all i; such an n exists because all f_i's are rational by the definition of a possible set of observations. Now for each element $(s_i, o_i, f_i) \in O$, let e_i be any effect in o_i. Define the multiset of sample states/effects $\mathcal{T}$ to contain $f_i n$ times the sample state/effect (s_i, e_i), for all $i = 1, \ldots, m$. Then from Proposition 6.2 we have $o_i = [e_i \setminus s_i, \mathit{apply}(s_i, e_i)]$, hence O is induced by $\mathcal{T}$, hence it is 0-fair to the distribution $D_{\mathcal{T}}$ of effects in $\mathcal{T}$. □

It is clear from the construction in the proof of Proposition 6.15 that there are in general many most likely effect distributions. In fact, each possible choice of one effect in each observed interval leads to a different, yet most likely, effect distribution. Further, in some given observed interval it may be the case that two different effects provoked the corresponding transitions, increasing still more the number of candidate distributions. Still worse, the different most likely distributions may be arbitrarily different from each other (in the sense of the L1-distance).

Example 6.16 (most likely distributions) *Let*

$$O = \{(01, [x_1, x_1x_2], 0.5), (10, [x_2, x_1x_2], 0.5)\}$$

as depicted on Figure 6.3. Then, for example, O is 0-fair to $D_1 = \{(x_1, 0.5), (x_2, 0.5)\}$, to $D_2 = \{(x_1x_2, 1)\}$, and to $D_3 = \{(x_1, 0.4), (x_1x_2, 0.3), (x_2, 0.3)\}$. To see why O is 0-fair to D_3, observe that if effect x_1x_2 has been sampled 0.1 times in state 01 and 0.2 times in state 10, then we indeed get the observations in O.

Now despite O is 0-fair to D_1 and to D_2, the L1-distance between both is maximal, namely 2, and so are the distributions of next-states $T(\cdot|00, D_1)$ and $T(\cdot|00, D_2)$ which they induce for state $s = 00$.

If some additional constraints have to be satisfied by the effect distribution sought for (like accounting for other observations at the same time), in general there will not be any effect distribution to which the observations are 0-fair. Hence we now define a measure of how likely it is that a *given* effect distribution indeed generated the observations. This is formalized by a notion of *deviation* of a set of observations from a distribution; the less the deviation, the more likely it is that the distribution generated the observations.

To understand the construction, observe the following for a sufficiently large set of observations O generated by a distribution D:

- if an effect e induced the interval o, then the observed frequency of o should be close to the probability of e in D,
- it is possible that an effect $e \in D$ with a sufficiently small probability p_e has never been observed, i.e., that O contains no interval o which contains e,
- every interval $o \in O$ *must* be induced by some effect e with nonzero probability in D, i.e., there must be $e \in D$ with probability $p_e > 0$ and $e \in o$,
- if two intervals $o_1, o_2 \in O$ intersect, it may be the case that only one effect $e \in D$ induced both of them (in different starting states), and in this case the observed frequencies of o_1, o_2 should approximately sum up to the probability of e in D,
- dually, if some interval $o \in O$ contains two effects e_1, e_2, it may be the case that both participated in inducing o, and in this case the probabilities of e_1, e_2 should approximately sum up to the observed frequency of o.

Generalizing these observations, we can see that one effect may induce several observed intervals, and dually that one interval may be induced by several effects. We take these possibilities into account by introducing values which we call *contributions of effects to intervals*, written $c_{e,o}$, and reflecting how often, among the times when e was sampled, it induced interval o.

We are now ready to define our notion of deviation. The choice of L1-distance is to agree with the definition of fairness, but again, other distances could be used in both. Note that the states where each effect interval was observed need not be taken into account in the definition.

Importantly, it can be questioned whether deviation is always well-defined, since it is defined as a minimum over all combinations of contributions, which may form an open set due to the constraint $c_{e,o} \neq 0$ for at least one effect e in o. We will show in Proposition 6.19 that the minimum is never taken of an emptyset, and in Proposition 6.23 that it can be defined as the objective value of a linear program, and hence that it is well-defined. Hence we admit its well-definedness for the moment.

Definition 6.17 (deviation) *Let D be an effect distribution, and let $O = \{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ be a set of possible observations.*

If for all intervals $o \in O$ there is an effect e satisfying $e \in o$ and with nonzero probability p_e in D, the deviation of O from D *is defined to be*

$$\Delta(O|D) = \min \left(\sum_{e \in D} \quad |p_D(e) - \sum_{o \in O} c_{e,o}| \right)$$

where the minimum is taken over all combinations of nonnegative values for $c_{e,o}$'s (for all effects e, observed intervals o) which satisfy the three conditions:

$$\begin{array}{ll} \text{if } e \notin o \text{ then } c_{e,o} = 0 & \forall o \in O, e \in D \\ \sum_{e \in D} c_{e,o} = f_o & \forall o \in O \\ \exists e \text{ such that } p_e \neq 0 \text{ and } c_{e,o} \neq 0 & \forall o \in O \end{array}$$

Otherwise, the deviation of O from D is defined to be 2.

The value to be minimized in the definition reflects the L1-distance between the effect distribution D and the frequencies of the realizations of these effects. A particular combination of $c_{e,o}$'s indicates a particular hypothesis about which effect provoked each observed transition, and how often. The constraints read, respectively:

- it cannot be imagined that e provoked the transition represented by o (may it be only once) if e is not in o; this is valid since by construction, o captures exactly the effects which may have caused the transition (Proposition 6.2),
- each time an interval/transition o has been observed, this must be accounted for by exactly one effect $e \in o$ having been sampled,
- each interval/transition must be accounted for by at least one effect with nonzero probability.

The case when for some interval o, there is no effect $e \in o$ with nonzero probability p_e, formalizes the impossible case when a transition has been observed though the environment cannot choose an effect which provokes it. Observe that this is exactly the case when there is no distribution satisfying all constraints, hence when the minimum scopes over the empty set of distributions (this is proved formally in Proposition 6.19 below).

Example 6.18 (continued) *Consider again the set of observations O in Example 6.16 (Figure 6.3). Then the deviation of O from the distribution $D_3 = \{(x_1, 0.4), (x_1x_2, 0.3), (x_2, 0.3)\}$ is $\Delta(D_3|O) = 0$, as witnessed by the contributions $c_{x_1,o_1} = 0.4$, $c_{x_1,o_2} = 0$, $c_{x_1x_2,o_1} = 0.1$, $c_{x_1x_2,o_2} = 0.2$, $c_{x_2,o_1} = 0$, $c_{x_2,o_2} = 0.3$. Now for $D = \{(x_1x_2, 0.9), (\overline{x}_1, 0.1)\}$ we get a deviation of 0.2 with $c_{x_1x_2,o_1} = c_{x_1x_2,o_2} = 0.5$, and for $D = \{x_1, 0.45), (x_2, 0.55)\}$ we get a deviation of 0.1. Finally, for $D = \{x_1, 1)\}$ we get a deviation of 2, since there is no explanation for o_2.*

6.5.2 Properties

We now prove several properties of deviation, and especially that this notion captures fairness exactly. We start with the following simple property, which justifies our choice for a deviation of 2 if there exists an interval which cannot be accounted for by an effect with nonzero probability.

Proposition 6.19 (range of deviations) *Let D be an effect distribution and O be a set of possible observations. Then the deviation of O from D satisfies $\Delta(O|D) \in [0, 2]$. Moreover, any (rational) value in $[0, 2[$ is attained, and $\Delta(O|D) = 2$ holds if and only if there is an interval $o \in O$ such that all effects $e \in o$ have probability $p_e = 0$ in D.*

Proof The fact that $\Delta(O|D)$ ranges in $[0, 2]$ follows from the fact that it is a L1-distance between vectors summing up to 1, namely, D and $(\sum_{o \in O} c_{e,o})_{e \in D}$.

To see why any value in $[0, 2[$ is attained, let $\epsilon \in [0, 2[$. Let $O = \{(0, [x_1], 1)\}$ and let $D = \{(x_1, 1 - \epsilon/2), (\overline{x}_1, \epsilon/2)\}$. Then clearly the deviation of O from D is ϵ.

If there is an unexplainable interval, equality to 2 holds by definition of Δ. Conversely, assume that for all intervals $o \in O$ there is an effect $e_o \in o$ with nonzero probability p_o in D. Then for all intervals o, define $c_{e_o,o} = f_o$, and $c_{e,o} = 0$ for all other effects e. Then clearly this set of contributions witnesses a deviation less than 2. □

We now show that this definition reflects exactly what can be deduced from the observations, in the sense that fairness and deviation coincide.

Lemma 6.20 *Let $\mathcal{T} = \{(s_i, e_i) \mid i = 1, \ldots, m\}$ be a multiset of sample states/effects, and let $O = \{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ be the set of observations induced by $\mathcal{T}$. If $\mathcal{T}$ is ϵ-fair to some effect distribution D, then the deviation of O from D is at most ϵ.*

Proof By definition of fairness, all effects occurring in $\mathcal{T}$ have a nonzero probability in D, hence we are in the first case of Definition 6.17.

Write $D = \{(e, p_e)\}$, and $D_{\mathcal{T}} = \{(e, p'_e)\}$ for the effect distribution induced by $\mathcal{T}$. To show ϵ-deviation, for each $e \in D$ and $o \in O$, we define $c_{e,o}$ to be $f_{e,o}$, where $f_{e,o}$ is the proportion of indices $i \in \{1, \ldots, m\}$ such that $e_i = e$ in $\mathcal{T}$ and $o_i = o$ in O. Clearly enough, all three constraints in Definition 6.17 are satisfied by these contributions. Now for all $e \in D$ we have by construction $\sum_{o \in O} c_{e,o} = p'_e$, hence we get $\Delta(O|D) \leq \sum_{e \in D} |p_e - p'_e|$ which as desired is at most ϵ, since $\mathcal{T}$ is ϵ-fair to D. □

`Variables:`	$c_{e,o}$	$\forall o \in O, e \in O$
`Minimize:`	$\sum_{e \in 3^X} \ \lvert p_e - \sum_{o \in O} c_{e,o} \rvert$	
`Subject to:`		
	$c_{e,o} \geq 0$	$(\forall o \in O, e \in O)$
	$\sum_{e \in o} c_{e,o} = f_o$	$(\forall o \in O)$

Figure 6.4: Linear program for computing the deviation of O from $D = \{(e, p_e)\}$

Lemma 6.21 *Let $O = \{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ be a possible set of observations. If the deviation of O from some effect distribution D is at most $\epsilon < 2$, then O is ϵ-fair to D.*

Proof From $\epsilon < 2$ we know that for all intervals $o \in O$ there is at least one effect e satisfying $e \in o$ and with nonzero probability in D (Proposition 6.19).

Now let $c_{e,o}$, for all $e \in D, o \in O$, be rational numbers which witness the deviation (argmin in Definition 6.17); because the deviation is the optimal solution of a linear program with rational coefficients (Proposition 6.23), such a combination of *rational* contributions must exist.

Let n be an integer such that $f_i n$ and $c_{e,o} n$ are integers for all $i = 1, \ldots, m$ and all $e \in 3^X, o \in O$. Let $\mathcal{T}$ be a multiset of n sample states/effects built as follows. For each observation o occurring in O, let S_o be the set of all states s such that the couple (s, o) occurs in O, and let E_o be the set of all effects e with $c_{e,o} \neq 0$. Then we add $f_o n$ sample state/effects to $\mathcal{T}$, each of the form (s, e) in such a way that each state s occurs as often as in S_o and each effect e occurs proportionnally to $c_{e,o}$. This is possible since $\sum_{e \in D} c_{e,o} = f_o$ is imposed for all intervals o by Definition 6.17.

From the construction and Proposition 6.2 it is clear that O is induced by $\mathcal{T}$. Now the L1-distance between D and $D_{\mathcal{T}}$ is:

$$||D - D_{\mathcal{T}}||_1 = \sum_{e \in D} \ |p_e - \sum_{o \in O} c_{e,o}|$$

which is $\Delta(O|D) \leq \epsilon$ by our choice of $c_{e,o}$'s. This concludes the proof. □

Corollary 6.22 (of Lemmas 6.20, 6.21) *Let O be a possible set of observations. For all $\epsilon < 2$, the effect distributions D with $\Delta(D|O) \leq \epsilon$ are exactly the distributions to which O is ϵ-fair.*

6.5.3 Linearity

Our motivation for introducing the notion of deviation is to provide a way to compute the likelihood of an effect distribution given a set of observations. Indeed, an important feature of Definition 6.17 is that it provides a linear programming approach to computing most likely distributions.

To see this, consider the program on Figure 6.4. Observe that the objective is defined in terms of absolute values. But since it is the sum of these values and because it must be minimized, we can introduce auxiliary variables using the well-known representation of $x = |y - z|$ by $x \geq y - z$ and $x \geq z - y$, which is a valid linear reformulation when x must be minimized.

The rest is a translation of the constraints in Definition 6.17, but without the constraint that for all $o \in O$, there is an effect $e \in 3^X$ with $p_e \neq 0$ and $c_{e,o} \neq 0$ (this constraint would have introduced a *strict* inequality constraint). We now show that the program still correctly computes the deviation of O from D, and moreover that the witnesses of this deviation can be retrieved from its solutions (even from those which do not satisfy the constraint left out).

Observe that the condition in the proposition is equivalent to saying that we are in the first case of Definition 6.17, or, equivalently (Proposition 6.19), that the deviation is less than 2.

Proposition 6.23 (linearity) *Assume that for all $o \in O$ there is an effect $e \in o$ with $p_e > 0$. Then the optimal value of the linear program on Figure 6.4 is exactly $\Delta(O|D)$, and from any optimal solution a combination of $c_{e,o}$'s can be retrieved straightforwardly, which witnesses the deviation $\Delta(O|D)$. In particular, the program computes $\Delta(O|D)$, and this deviation is well-defined.*

Proof We only have to check that given an optimal solution of the linear program, one can build another optimal solution such that for each observation $o \in O$ there is an effect e with nonzero probability and nonzero contribution to o. Indeed, the rest of the program is a direct reformulation of Definition 6.17.

So let $c_{e,o}$, for all $o \in O$, $e \in o$, form a solution of the program with value ϵ, and let $o \in O$ be an interval such that for all effects $e \in o$, either $c_{e,o}$ or p_e is 0. Because of the constraint $\sum_{e \in o} c_{e,o} = f_o$, there must be some effect $e \in o$ with $c_{e,o} > 0$, and hence $p_e = 0$.

On the other hand, by assumption there is an effect $e' \in o$ with probability $p_{e'} > 0$. Now consider the contributions obtained by adding $c_{e,o}$ to $c_{e',o}$ and setting $c_{e,o}$ to 0. Then clearly this new set of contributions now satisfies all constraints in the definition of deviation. Now because p_e is 0, the operations performed make the error $|p_e - \sum_{o \in O} c_{e,o}|$ decrease by an amount of $c_{e,o}$, and by the triangle inequality the error for e' cannot increase by more than this amount, hence the value of the resulting solution is at most ϵ. □

Though the deviation is correctly computed by this linear program, there may indeed be cases where some optimal solution is not a correct witness, as the following example shows.

Example 6.24 *Let $O = \{(01, [x_1, x_1x_2], 0.5), (10, [x_2, x_1x_2], 0.5)\}$ (as depicted on Figure 6.3), and $D = \{(x_1, 0), (x_2, 0.25), (x_1x_2, 0.25), (\overline{x}_1, 0.5)\}$. Observe that the deviation $\Delta(O|D)$ is at least 1, since there is a 0.5-excess on effect $\overline{x}_1$ and a 0.5-shortage on other effects. Hence the set of contributions given by $c_{x_1,o_1} = 0.5$, $c_{x_1x_2,o_1} = 0$, $c_{x_1x_2,o_2} = 0.25$, $c_{x_2,o_2} = 0.25$ (all others being zero) is optimal, though it does not respect the constraint that there is an effect $e \in o_1$ with $p_e > 0$ and $c_{e,o_1} > 0$. However, following the proof of Proposition 6.23, an optimal solution respecting this constraint for all intervals can be retrieved by "redirecting" the contribution of x_1 to o_1 to the effect x_1x_2, which has nonzero probability.*

Note that the linear program is not intended to be used as such. If a most likely distribution is sought for a given set of observations, then the construction in Proposition 6.15 is more straightforward. On the other hand, this linear programming formulation is useful when the distribution must satisfy additional (linear) constraints, in which case a "perfectly likely" distribution will not exist in general. In particular, we will use it for computing variance in the next section.

6.6 Variance of Sets of Observations

We now turn to the general setting where we want to measure how likely it is that several sets of observations $O_1, \ldots, O_q$ are all induced by some effect distribution D. Intuitively, we will use this measure for uncovering the conditions of the action to be learnt, with each set of observations O_i corresponding to some candidate condition.

Naturally, we want to measure this likeliness by the radius of a ball centered at D and containing all O_i's, $i = 1, \ldots, q$. The smaller this radius, the more likely will all sets of observations be induced by D. And the smaller the minimum radius over all distributions D, the less the "variance" of the sets of observations will be.

6.6.1 Definition and Properties

From Corollary 6.22, which shows that our notion of deviation exactly captures the likelihood of an effect distribution given observations, it follows that deviation is the only candidate (unbiased) measure for defining the radius of such balls. The center of such a ball with minimal radius is known as the *Chebyshev center* of the given sets of observations.

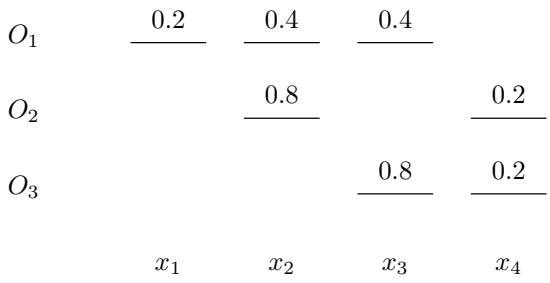


Figure 6.5: Example sets of observations with no Chebyshev center

We have however taken care in our definition of deviation to require that any observed interval is taken into account by at least one effect with nonzero probability. This has the unfortunate consequence that the center sought for needs not exist, even if there is a distribution from which all given sets of observations have deviation less than 2 (as is in fact always the case, as we prove in Proposition 6.27).

Example 6.25 (no Chebyshev center) *Consider the three following sets of observations (states omitted)*[2]*. These observations are also depicted on Figure 6.5.*

$$\begin{aligned} O_1 &= \{([x_1], 0.2]), [x_2], 0.4), ([x_3], 0.4)\} \\ O_2 &= \{([x_2], 0.8), ([x_4], 0.2)\} \\ O_3 &= \{([x_3], 0.8), ([x_4], 0.2)\} \end{aligned}$$

Clearly, the distributions with the smallest variance are distributions only on effects x_i, $i = 1, \ldots, 4$. So let $D = \{(x_1, 1 - p_2 - p_3 - p_4)), (x_2, p_2), (x_3, p_3), (x_4, p_4)\}$ be any such distribution. We show that whatever the combination of p_i's, the deviation of O_2 or O_3 from D must be (strictly) more than 0.8, while for any $\epsilon > 0$ ($\epsilon < 0.2$) there is a combination of p_i's such that the deviation of O_1, O_2, O_3 from D is at most $0.8 + \epsilon$. From this it follows that the center of O_1, O_2, O_3 does not exist (it is the minimum of an open set).

To see the first point, first assume $p_2, p_3 \leq 0.8$ and $p_4 \leq 0.2$. Then it is easily seen that $\Delta(O_2|D) = 2 - 2(p_2 + p_4)$ and $\Delta(O_3|D) = 2 - 2(p_3 + p_4)$ hold. Hence if $\max_{i=1,2,3} \Delta(O_i|D) \leq 0.8$ holds, then in particular $p_2 + p_3 + 2p_4 \geq 1.2$ must hold, hence with the assumption $p_4 \leq 2$ we must have $p_2 + p_3 + p_4 \geq 1$, in contradiction with the constraint $1 - p_2 - p_3 - p_4 > 0$ (so that the interval $[x_1]$ is explained in O_1).

Now assume $p_2 > 0.8$ (hence $p_3 < 0.2$) and $p_4 \leq 0.2$. Then $\Delta(O_3|D) = 2 - 2p_3 - 2p_4 > 1.2$ must hold. The case when $p_3 > 0.8$ and $p_4 \leq 0.2$ hold is dual.

Finally, assume $p_4 > 0.2$. Then we must have $p_2 + p_3 < 0.8$, $\Delta(O_2|D) = 1.6 - 2p_2$, and $\Delta(O_3|D) = 1.6 - 2p_3$. Hence if $\max_{i=1,2,3} \Delta(D|O_i) \leq 0.8$ holds, then in particular $p_2 \geq 0.4$ and $p_3 \geq 0.4$ must hold, a contradiction.

We have thus shown that there is no distribution D such that the deviations of O_1, O_2, O_3 from D are all at most 0.8. On the other hand, let $\epsilon > 0$ ($\epsilon < 0.2$), and let $p_2 = 0.4 - \epsilon/2$, $p_3 = 0.4 - \epsilon/2$, and $p_4 = 0.2$. Then each effect has a nonzero probability, and the deviations of O_1, O_2, O_3 from D are $0.4, 0.8 + \epsilon, 0.8 + \epsilon$, respectively.

We can however define what it means for sets of observations to be ϵ-variant. Observe that our definition allows sets of observations to be 0-variant (and in this case, the Chebyshev center exists, see Proposition 6.32). Also note that the definition could have been with an infimum instead of a minimum, but in our opinion this would have hidden the requirement that every observed interval *must* be explained by some effect with nonzero probability.

[2]Technically, these are not *possible* sets of observations, because the upper bounds do not mention all variables, but simply think of four *disjoint* intervals.

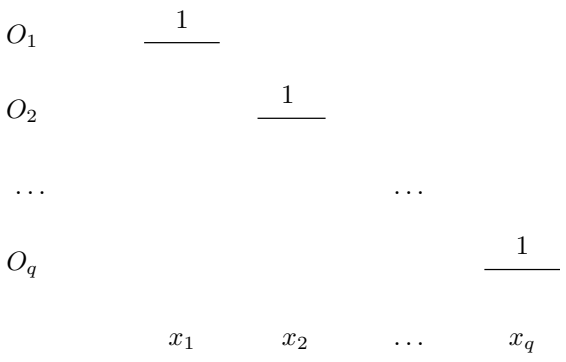


Figure 6.6: Example sets of observations with high variance

Definition 6.26 (variance) *Let $O_1, \ldots, O_q$ be sets of observations, and let $\epsilon \in [0, 2]$. Then $O_1, \ldots, O_q$ are said to be* ϵ-variant *(together) if there is an effect distribution D such that for all $i = 1, \ldots, q$, $\Delta(O_i|D) \leq \epsilon$ holds.*

Even if deviations may equal 2, minimizing over all distributions always gives a variance less than 2.

Proposition 6.27 (range of variance) *Let $O_1, \ldots, O_q$ be possible sets of observations. Then the variance of $O_1, \ldots, O_q$ is in $[0, 2[$.*

Proof Let $o^1, \ldots, o^k$ enumerate all the intervals occurring in $O_1 \cup \cdots \cup O_q$, and let $e_1, \ldots, e_k$ be k effects, with $e_i \in o^i$ $(i = 1, \ldots, k)$ and with probability $1/k$ each. For each effect e_i let $c_{e_i,o^i} = f_{o^i}$, and $c_{e_i,o^j} = 0$ for $j \neq i$. Then clearly this combination of contributions witnesses a variance less than 2. □

Like for deviation, any value $\epsilon \in [0, 2[$ can be attained by some combination of sets of observations (though not, we conjecture, for a fixed number of them), in the sense that there exist set of observations which are *not* ϵ-variant together.

Example 6.28 *Let $q \in \mathbb{N}$, and for $i = 1, \ldots, q$ let $O_i = \{([x_i], 1)\}$ (the same remark as in Footnote 2 on Page 67 applies). These observations are depicted on Figure 6.6. Clearly, for any ϵ any effect distribution witnessing ϵ-variance can be chosen of the form $D = \{(x_i, p_i) \mid i = 1, \ldots, q\}$. Now for a fixed set of p_i's we have*

$$\begin{aligned} \max_{i=1,\ldots,q} \Delta(D|O_i) &= \max_{i=1,\ldots,q} (p_1 + \cdots + p_{i-1} + (1 - p_i) + p_{i+1} + \cdots + p_q) \\ &= \max_{i=1,\ldots,q} ((1 - p_i) + (1 - p_i)) \end{aligned}$$

Due to the constraint that p_i's sum up to 1, this quantity is necessarily at least $2 - 2/q$, which can be made arbitrarily close to 2.

As an immediate consequence of Corollary 6.22 we have:

Proposition 6.29 (small variance) *Let $O_1, \ldots, O_q$ be possible sets of observations. Then there is an effect distribution D to which $O_1, \ldots, O_q$ are all ϵ-fair if and only if $O_1, \ldots, O_q$ are ϵ-variant.*

We also have the following straightforward property. In informal terms, it states that if the sets of observations $O_1, \ldots, O_q$ are likely to be all generated by the same effect distribution D, then D also likely generated *all* observations, that is, the multiset $O = O_1 \cup \cdots \cup O_q$.

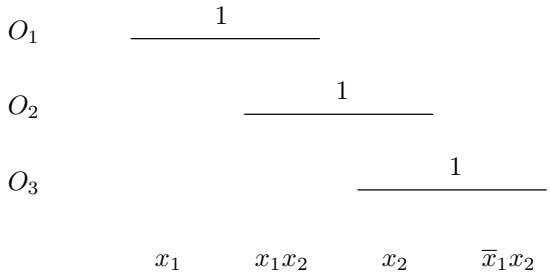


Figure 6.7: Example sets of observations for which the triangle inequality does not hold

Proposition 6.30 *If $O_1, \ldots, O_q$ are ϵ-variant as witnessed by the effect distribution D, then the set of observations $O = O_1 \cup \cdots \cup O_q$ is ϵ-fair to D.*

Proof For each set of observations O_i, $i = 1, \ldots, q$, let $\mathcal{T}_i$ be a multiset of sample states/effects which induces O_i and which is fair to D. Such a multiset exists because by Proposition 6.29 each O_i must be fair to D.

Now let $\mathcal{T}$ be the multiset of sample states/effects defined as $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_q$. Clearly $\mathcal{T}$ induces O, and on the other hand its induced effect distribution $D_{\mathcal{T}}$ is the mean of all distributions $D_{\mathcal{T}_1}, \ldots, D_{\mathcal{T}_q}$. Since each of them is at L1-distance at most ϵ from D, by properties of L1-distance so is $D_{\mathcal{T}}$, which completes the proof. □

Obviously, the contrary does not hold in general. Indeed, whatever the possible sets of observations $O_1, \ldots, O_q$, the combined set of observations $O_1 \cup \cdots \cup O_q$ is always 0-fair to some effect distribution (Proposition 6.15), whereas the variance of sets $O_1, \ldots, O_q$ may be arbitrarily large (Example 6.28).

Another important point is that when restricted to two sets of observations, variance does not define a distance. Intuitively, this is simply because the witness D_{12} for the variance of O_1, O_2 needs not be the same as the witness D_{23} for O_2, O_3. In fact, the triangle inequality can even be made "very false", as the following example shows.

Example 6.31 *Let $O_1 = \{([x_1, x_1x_2], 1)\}$, $O_2 = \{([x_2, x_1x_2], 1)\}$, $O_3 = \{([x_2, \overline{x}_1x_2], 1)\}$, as depicted on Figure 6.7. Then O_1, O_2 are 0-variant (using $D = \{(x_1x_2, 1)\}$), and so are O_2, O_3 (using $D = \{(x_2, 1)\}$). Nevertheless, because the intervals in O_1 and O_3 are disjoint, for any distribution D at least one of $\Delta(O_1|D) \geq 1$ or $\Delta(O_3|D) \geq 1$ must hold, hence O_1, O_3 are not ϵ-variant for any $\epsilon < 1$.*

6.6.2 Linearity

We now build on the linear program for computing deviations, and design a linear program which decides ϵ-variance for given sets of observations. More precisely, the program is not given ϵ as an input, and rather finds the smallest ϵ such that the given sets of observations are $(\epsilon + \alpha)$-variant for any $\alpha > 0$. Proposition 6.32 makes this precise.

This program is given on Figure 6.8. It is essentially a concatenation of instances of the program for deviation, one for each multiset of observations $O_1, \ldots, O_q$, together with additional variables for the probabilities of effects. Similarly to the program for deviation, this program is indeed linear because absolute values and maxima must be minimized.

Proposition 6.32 (linearity) *The optimal value σ of the linear program on Figure 6.8 satisfies $\sigma \leq \epsilon$ if and only if the given sets of observations are $(\epsilon + \alpha)$-variant for any $\alpha > 0$.*

Proof The "if" direction directly follows from the fact that any effect distribution D witnessing $(\epsilon + \alpha)$-variance gives a feasible solution with value $(\epsilon + \alpha)$ for the linear program, hence there must be a feasible solution with value $\lim_{\alpha \to 0}(\epsilon + \alpha) = \epsilon$.

`Variables:`	p_e	$(\forall e \in 3^X)$
	$c^i_{e,o}$	$(\forall i = 1, \ldots, q, o \in O_i, e \in o)$
`Minimize:`	$\max_{i=1,\ldots,q} \sum_{e \in 3^X} \ \lvert p_e - \sum_{o \in O_i} c^i_{e,o} \rvert$	
`Subject to:`	$p_e \geq 0$	$(\forall e \in 3^X)$
	$c^i_{e,o} \geq 0$	$(\forall i = 1, \ldots, q, o \in O_i, e \in o)$
	$\sum_{e \in 3^X} p_e = 1$	
	$\sum_{e \in o} c^i_{e,o} = f_o$	$(\forall i = 1, \ldots, q, o \in O_i)$

Figure 6.8: Linear program for computing variance

Conversely, let p_e, for all $e \in 3^X$, be the probabilities of effects in a solution of value ϵ for the linear program, and let $\alpha > 0$ ($\alpha < 1/3^n$). We show that the given sets of observations are $(\epsilon + \alpha)$-variant.

Let e be an effect with $p_e \geq 1/3^n$; there must be one since there are 3^n effects and their probabilities must sum up to 1 in any feasible solution. Now for all intervals o in any of the given sets of observations, such that there is no $e \in o$ with $p_e > 0$ and $c_{e,o} > 0$, let e_o be an effect in o with $c_{e_o,o} > 0$ (there must be one by the constraint $\sum_{e \in o} c_{e,o} = f_o$). We define a new feasible solution for the linear program by setting $p_{e_o} = \alpha/3^{n+1}$ for all such e_o's, and for each of them decreasing p_e by the same amount. The resulting solution (with contributions unchanged) is clearly feasible for the linear program, and for all intervals o there is an effect e_o with nonzero probability and nonzero contribution to o. Now the value of this resulting solution has increased by less than α, since we added $\alpha/3^{n+1}$ to at most 3^n effect probabilities p_{e_o}, and removed less than $3^n \alpha/3^{n+1} = \alpha/3$ from p_e, hence by the triangle inequality the deviation from the solution of any of the given sets of observations has increased by at most $2\alpha/3$. □

We are even able to prove that the linear program correctly decides 0-variance.

Proposition 6.33 *If $O_1, \ldots, O_q$ are ϵ-variant for all $\epsilon > 0$, then they are 0-variant. In particular, if there is an optimal solution with value 0 to the program on Figure 6.8, then $O_1, \ldots, O_q$ are 0-variant.*

Proof The second part follows from the first one together with Proposiion 6.32 ("only if" direction). Hence we show the first part.

Let $\epsilon = \frac{1}{2} \min \frac{f_o}{|o|}$, where the minimum is taken over all observations o in any of the given sets, and let p_e ($e \in 3^X$), $c_{e,o}$ ($o \in O_1 \cup \cdots \cup O_q$) constitute a solution of the linear program with value at most ϵ. From the constraint $\sum_{e \in o} c_{e,o} = f_o$ it follows that for all intervals o, there is an effect e with $c_{e,o} \geq \frac{f_o}{|o|} \geq 2\epsilon$. Hence this effect must have probability p_e at least ϵ.

From this it follows that we can add the linear constraint $\sum_{e \in o} p_e \geq \frac{1}{2} \min \frac{f_o}{|o|}$ $(\forall o \in O_1 \cup \cdots \cup O_q)$ to the linear program on Figure 6.8 without changing its set of solutions with value less than ϵ. It follows that the value $\lim_{\epsilon \to 0} \epsilon = 0$ is attained by some feasible solution, which satisfies that for all intervals o, there is an effect $e \in o$ with nonzero probability and (with the same reasoning as in Proposition 6.23) nonzero contribution to o. □

Open Question 3 *As Example 6.25 shows, Proposition 6.33 is not true if $\epsilon = 0 + \epsilon$ is replaced with $\sigma + \epsilon$. We however leave the question open whether it may be the case that there exist optimal solutions with value σ in which not all intervals are explained, but there exist other optimal solutions in which all of them are.*

6.7 Identifying the Effects

So far, we have studied the problem of determining likely effect distributions given observations, or sets of observations, and given efficient algorithms for doing so. However, we have shown that with an unbiased approach, there are in general many candidate distributions which are most likely.

We now investigate the extent to which this number of candidate distributions can be restricted, and especially how the *real* effects in the target distribution can be determined (independently from their relative probabilities). For this we will need to give more power to the learner, by allowing it to partially influence the set of states in which transitions are observed. Hence our approach gives hints about how an exploration strategy should be designed, if the underlying setting is a reinforcement learning problem.

6.7.1 Influencing the Choice of States

We propose to allow the learner to fix a bounded number of literals (a *term* t), and ask the environment to sample states/effects of the form (s, e) with s satisfying the term t (and, as before, the effect e being sampled from the target distribution D).

The basic idea is that the learner will ask a number of transitions for each possible choice of a (bounded-size) term t, and in this manner it will be able to disambiguate a number of effects in the observed transitions. The choice of imposing a bound k on the terms chosen by the learner is to make sure that the set of all possible choices of a term t has a reasonable size (more exactly, a polynomial size), namely, there are $\binom{n}{k}2^k$ such choices, where n is the number of variables. In this section, we write 3^k for the set of all terms of size k (the underlying set of variables will always be clear from the context).

Example 6.34 *Assume $X = \{x_1, x_2, x_3, x_4\}$ and $k = 2$. Then there are $\binom{4}{2}2^2$ possible choices of terms t for the learner, namely, any choice of $k = 2$ variables and any choice of polarities for them. If the learner chooses $t = \overline{x}_2 \wedge x_3$, then the environment must present a transition of the form (s, s') with s any state assigning 0 to x_2 and 1 to x_3 (but otherwise possibly adversarially chosen), and $s' = apply(s, e)$ for some effect e sampled i.i.d. from the target, hidden distribution of effects.*

What we show next is that the learner can get information about the real effects (with nonzero probability) in the target distribution, if it obtains sufficiently many sample states/effects per possible choice of a term t. We will use the following definition.

Definition 6.35 (projection) *Let $O = \{(s_i, o_i, f_i) \mid i = 1, \ldots, m\}$ be a possible set of observations, and let t be a term. The* projection of O onto t*, written O^t, is the possible set of observations $\{(s_i, o_i, f_i^t) \mid i = 1, \ldots, m, s_i \models t\}$, where $f_i^t = f_i / \sum\{f_i \mid i = 1, \ldots, m, s_i \models t\}$ denotes the frequency of (s_i, o_i) in O^t.*

Example 6.36 *Let*

$$O = \{(01, [x_1, x_1x_2], 0.4), (00, [x_1, x_1\overline{x}_2], 0.2), (00, [\emptyset, \overline{x}_1\overline{x}_2], 0.2), (00, [x_1, x_1\overline{x}_2], 0.2)\}$$

be the possible set of observations in Example 6.14 (Page 61). Then the projection of O onto $t = \overline{x}_2$ is

$$O^t = \{(00, [x_1, x_1\overline{x}_2], 1/3), (00, [\emptyset, \overline{x}_1\overline{x}_2], 1/3), (00, [x_1, x_1\overline{x}_2], 1/3)\}$$

and its projection onto $t' = \overline{x}_1 \wedge x_2$ is $\{(01, [x_1, x_1x_2], 1)\}$.

6.7.2 Closure of Sets of Effects

Quite obviously, the greater the bound k on the size of the terms which the learner can fix, the easier it is to identify the effects in the target distribution. We will make this formal by showing that it is possible to efficiently identify any *k-closed* set of effects using terms of size k.

Given a term $t = \ell_1 \wedge \cdots \wedge \ell_k$, we write $\overline{t}$ for the term $\overline{\ell}_1 \wedge \cdots \wedge \overline{\ell}_k$.

Definition 6.37 (closure) *Let E be a set of effects, and let $k \in \mathbb{N}$. An effect $e \in 3^X$ is said to be in the* k-closure *of E if for all terms t of size k, there exists an effect e^t in E such that e^t and e are consistent together and $e^t \cap \bar{t} = e \cap \bar{t}$ holds. A set of effects E is said to be* k-closed *if it is equal to its k-closure.*

Observe that E is always included in its k-closure, since e is a witness of its own membership for all terms $t \in 3^k$.

Example 6.38 *Consider the set of effects*

$$E = \{x_1\overline{x}_2x_4, x_1\overline{x}_2x_3, x_3x_4\overline{x}_5\}$$

Then $e = x_1\overline{x}_2x_3x_4$ is in the 2-closure of E. For instance, considering $\bar{t} = x_1\overline{x}_2$ we find that e matches the first effect in E on this term. Considering $\bar{t} = x_3x_5$, we find that it matches the second effect. Considering $\bar{t} = x_4\overline{x}_5$, we find that it matches the first effect (but not the third one), and so on.

On the other hand, the effect $e' = x_1x_3$ is not *in the 2-closure of E, because considering the term $\bar{t} = x_1\overline{x}_2$ we cannot find an effect in E which contains x_1 but not $\overline{x}_2$, while so does e'.*

The property of k-closure is a rather natural property for a set of effects. Consider for instance $k = 3$. Then the main intuition is that a set of effects is 3-closed if an effect e may occur (i.e., is in E) if *all* its subeffects of size 3 may also occur in effects not contradicting e, and dually for its "noneffects" (terms $\bar{t}$ with $\bar{t} \cap e = \emptyset$). Note that this is simply for intuition, since the definition requires even more for an effect to be in the k-closure, in particular about the terms which "overlap" it.

As an example, consider the set of effects $E = \{x_1x_2, x_1x_3, x_1x_4\}$. Then the effect $e = x_1$ is in the 2-closure of E, partly because for the terms $\bar{t}_i = x_1x_i$, there is an effect which contains x_1 but not x_i, just as e does. On the other hand, it is not in its 3-closure, because there is no effect which agrees with it on the term $\bar{t} = x_2x_3x_4$ (all effects in e contain one of these variables, while e contains none).

A natural family of effect sets which are k-closed are as follows. For any $n \in \mathbb{N}$, it can be checked that the set of effects $\{x_1, x_1x_2, \ldots, x_1x_2 \ldots x_n\}$ is k-closed for any $k \in \mathbb{N}$. Intuitively, these are sets of effects corresponding to an action for which succeeding in the effect x_{i+1} is conditioned on success for x_i. For instance, trying to open a door may fail, but if it works there may be someone right behind, and if there is someone this might be one whom the agents knows, etc. (observe that it is typically very difficult to distinguish between the direct effects of an action and its ramifications when learning from plain transitions in the environment).

We are now ready to give the main result of this section.

Proposition 6.39 *Let O be a set of observations induced by an effect distribution D, and let $k \in \mathbb{N}$. Write E for the set of all effects with nonzero probability in D, and for all terms t of size k, write $E^t = \bigcup\{o \mid (s, o, f) \in O^t\}$ for the set of all effects which occur in at least one interval observed in states which satisfy t. Then any effect $e \in \bigcap_{t \in 3^k} E^t$ is in the k-closure of E.*

Proof By construction, for all terms $t \in 3^k$, e occurs in an observed interval of the form $o^t = [e^t \setminus s^t, \mathit{apply}(s^t, e^t)]$.

Let $t \in 3^k$ be such a term. From $e^t \setminus s^t \subseteq e$ and $s^t \models t$ we get $e^t \cap \bar{t} \subseteq e$ and hence $e^t \cap \bar{t} \subseteq e \cap \bar{t}$. Now from $e \subseteq \mathit{apply}(s^t, e^t)$ we get $e \cap \bar{t} \subseteq \mathit{apply}(s^t, e^t) \cap \bar{t}$, and since $s^t \models t$ holds, we get $e \cap \bar{t} \subseteq e^t \cap \bar{t}$. Finally, from $e \subseteq \mathit{apply}(s^t, e^t)$ we get that e and e^t are mutually consistent. Hence e^t is a witness for membership of e in the closure of E. □

Example 6.40 *Let $E = \{x_1, x_2\}$ (e.g., with distribution $D = \{(x_1, 0.5), (x_2, 0.5)\}$), let*

$$O = \{(01, [x_1, x_1x_2], 1/3), (11, [\emptyset, x_1x_2], 1/3), (10, [x_2, x_1x_2], 1/3)\}$$

as obtained for instance from the multiset of sample states/effects

$$\mathcal{T} = \{(01, x_1), (01, x_1), (11, x_1), (11, x_2), (10, x_2), (10, x_2)\}$$

and let $k = 1$*. Then we have:*

$$\begin{aligned} O^{x_1} &= \{(11, [\emptyset, x_1x_2], 0.5), (10, [x_2, x_1x_2], 0.5)\} \\ O^{\overline{x}_1} &= \{(01, [x_1, x_1x_2], 1)\} \\ O^{x_2} &= \{(01, [x_1, x_1x_2], 0.5), (11, [\emptyset, x_1x_2], 0.5)\} \\ O^{\overline{x}_2} &= \{(10, [x_2, x_1x_2], 1)\} \end{aligned}$$

Then the effect x_1x_2 *occurs in at least one interval in each* O^t*, and indeed, it is in the* 1*-closure of* E *(as witnessed by* $x_1 \in E$ *over* $t = x_1$ *and* $t = \overline{x}_1$*, and by* $x_2 \in E$ *over* $t = x_2$ *and term* $= \overline{x}_2$*).*

We also have the following straightforward result, which we will use a little later (the notation is the same as in Proposition 6.39).

Proposition 6.41 *Let* O *be a set of observations induced by an effect distribution* D*, and let* $k \in \mathbb{N}$*. Let* $e \in 3^X$ *be an effect with nonzero probability in* D*. If for all terms* $t \in 3^k$*,* e *has been sampled at least once in a state satisfying* t*, then* e *is in* $\bigcap_{t \in 3^k} E^t$*.*

Proof This follows directly from the fact that if (s, e) is a sample state/effect with $s \models t$, then e is in the induced interval $[e \setminus s, \mathit{apply}(s, e)]$ and hence, in E^t. □

6.7.3 PAC-Learning Sets of Effects

We now derive two results about PAC-learning sets of effects (let apart their probabilities). Though they take place in two different models, the idea in both cases is that the learner is indeed able to acquire a sufficient number of observations for each possible term t.

Hence in this section, *learning* a hidden set of effects $E \subseteq 3^X$ refers to the problem of identifying E from *examples*, where these examples are transitions provoked by these effects (equivalently, effect intervals).

For the following result, define for all $k \in \mathbb{N}$ the *k-partially specified state oracle* (k-PSS-oracle) to be the query $PSS^k(\cdot)$ which the learner can use with a term $t \in 3^k$ of its choice, and such that $PSS^k(t)$ returns a sample transition (s, s') with s chosen arbitrarily but satisfying t, and s' is $\mathit{apply}(s, e)$ for an effect e sampled i.i.d. from D.

Proposition 6.42 *The class of all* k*-closed sets of effects is PAC-learnable by a learner which uses a number of calls to a* k*-PSS oracle polynomial in* $n^k, 1/\epsilon, 1/\delta$*, where* n *is the number of variables over which states are defined. Precisely, given* $\epsilon, \delta > 0$*, a hidden* k*-closed set of effects* E*, and a hidden distribution* $D = \{(e, p_e)\}$ *on* E*, the learner outputs a set of effects* $\hat{E}$ *such that* $\hat{E} \subseteq E$ *holds and with probability at least* $1 - \delta$*,* $\sum\{p_e \mid e \in E \setminus \hat{E}\} < \epsilon$ *holds.*

Proof Using Propositions 6.39 and 6.41, it is enough for the learner to ensure that all effects in E have been sampled in O^t for all terms t, except maybe for a proportion at most $\epsilon / \binom{n}{k} 2^k$ of them (as measured by the hidden distribution D). Indeed, in this case, modulo a union bound on confidences, a proportion of effects at most $\binom{n}{k} 2^k \times \epsilon / \binom{n}{k} 2^k = \epsilon$ has been missed in E over all terms t. Now Chernoff's bound ensures that a polynomial number $m_{\delta,\epsilon}$ of sample state/effects per term t (that is, of calls to the PSS-oracle) is enough to ensure this. □

The second formalization assumes a uniform distribution on states in place of access to a k-PSS oracle, but the intuitions are the same.

Proposition 6.43 *The class of all* k*-closed sets of effects is PAC-learnable by a learner which uses a polynomial (in* $n^k, 1/\epsilon, 1/\delta$*) number of examples of the form* (s, s')*, where* s *is sampled i.i.d. from*

the uniform distribution on 2^X*, and* s' *is apply*(s, e) *for an effect* e *sampled from* D*. Precisely, given* $\epsilon, \delta > 0$*, a hidden* k*-closed set of effects* E*, and a hidden distribution* $D = \{(e, p_e)\}$ *on* E*, the learner outputs a set of effects* $\hat{E}$ *such that* $\hat{E} \subseteq E$ *holds and with probability at least* $1 - \delta$*,* $\sum\{p_e \mid e \in E \setminus \hat{E}\} < \epsilon$ *holds.*

Proof We simply show that a polynomial number of examples is enough to obtain $m_{\delta,\epsilon}$ examples (s, s') with s satisfying t, for all terms t, where $m_{\delta,\epsilon}$ is as in the proof of Proposition 6.42.

For this we consider the (nondisjoint) events $(s \models t)_{t \in 3^k}$. Clearly, since states are chosen acoording to the uniform distribution, also these events are uniformly distributed. Moreover, each state s realizes $\binom{n}{k}$ of these (all those corresponding to terms t satisfied by s). Hence the probability of each of them is $1/2^k$. Then using the bound by Weissman et al. (2003) on this distribution of events, we can guarantee with a polynomial number of sample states/effects m that with probability at least $1 - \delta/\binom{n}{k}2^k$, each of these events occurred with a frequency at least $1/\binom{n}{k}2^k - \epsilon_0$ for some ϵ_0. Now a polynomial number of sample state/effects is also enough to ensure that m is large enough for this proportion to be at least $m_{\delta,\epsilon}$. A union bound on confidences concludes the proof. □

6.8 Perspective: Learning Conditions of Actions

We now propose to use the tools developed in Section 6.6, especially the efficient linear programming approach for computing variance, to give a heuristic approach for identifying the conditions of an action from observed transitions.

We place this discussion in essentially the same setting as the one used by Strehl et al. (2007). Precisely, we assume that there is a hidden *Probabilistic STRIPS Operator* consisting of (mutually inconsistent) conditions associated with distributions of effects, of the form $\{(c_i, D_i)\}$, of which the conditions, the effects, and their probabilities are unknown. Nevertheless, we assume that there is a known bound k on the number of variables used in the conditions of the hidden action (that is, $|\cup \mathit{Vars}(c_i)| \leq k$).

Our proposal mimicks the study by Strehl et al. (2007), but the price to pay (at least for the moment) for ambiguous effects is that our approach is heuristic, while theirs is provably correct. Observe that their approach, SLF-Rmax, also observes *transitions*, but it learns the effects on each variable independently (as Dynamic Bayesian Networks), hence it relies on the assumption that these are probabilistically independent. Their approach could in principle be augmented to deal with correlated effects, but this would not respect the bound k assumed on the number of parents of each variable. Indeed, if a variable is correlated with k' other variables in the effects, then its number of parents becomes $k + k'$, while our approach is not dependent on the size of effects (i.e., on their amount of correlation).

The basic principle of SLF-Rmax is to gather statistics on each candidate combination of parents and each instantiation of them, that is, to gather observations for each term $t \in 3^k$, just as we did in Section 6.7.3. For reasons which will become clear soon, they also gather statistics on terms over $2k$ variables.

The following observations allow the learner to distinguish between the correct set of k variables (written X_c) and others:

1. if t is a term on exactly the correct set of variables X_c, then the observations $O^{t \wedge t'}$ over $t \wedge t'$, for all other terms t', will be similar to each other; this is because per definition, the distribution induced by the target distribution D on the states satisfying $t \wedge t'$ is the distribution associated to the condition $c_i = t$, whatever t';

2. conversely, if t is a term of size k such that the observations $O^{t \wedge t'}$ over $t \wedge t'$, for all other terms t', are similar to each other, then either t is over the correct set of variables X_c, or t_0 is the correct term and we have the following: O^t is similar to $O^{t \wedge t_0}$ (it is the center of a ball with small radius, per assumption), and because t_0 is over the correct set of variables, $O^{t \wedge t_0}$ is similar to O^{t_0}, which is itself similar to the correct distribution (for condition $c_i = t_0$); the

triangle inequality allows to conclude that the observations on t are similar to the correct distribution.

From these two observations, Strehl et al. (2007) derive an algorithm able to learn a correct set of variables (more precisely, a correct term of size k for each state), hence to deduce the structure of a correct DBN for the hidden action (precisely, for its effect on each variable). The algorithm can then estimate the DBN parameters using a maximum likelihood approach (observed frequencies of effects x and $\overline{x}$).

Clearly enough, the above observations hold whatever the underlying model for actions. In particular, they hold when the action is assumed to be represented as a PSO with at most k variables occurring in the conditions. Using our definition of variance of sets of observations and the results presented in Section 6.6, we propose the following approach.

Given a state s (think of the state currently occupied by the agent in a reinforcement learning setting), and assuming that sufficiently many observations O^t have been gathered for each term t satisfied by s, the agent computes the variance of these sets of observations using the linear programming approach of Section 6.6 (restricted to candidate effects occurring in all sets of observations, as in Section 6.7.3). It follows directly from the analysis presented that the variance of the O^t's is a lower bound of their "real" variance, that is, of the radius of the smallest ball centered at the target distribution D and containing all O^t's. Hence, Condition 1 above is still satisfied by the term t on the correct variables. On the other hand, and this is why our approach can only be heuristic, for a term t to satisfy Condition 2 does not entail that the corresponding observations are indeed similar to the target distribution. This is because the triangle inequality does not hold for the measure of deviation with ambiguous effects, as demonstrated in Example 6.31.

Hence our approach is only heuristic, namely, complete (the "real" term will be recognized as a candidate), but not sound. Some preliminary experiments have nevertheless demonstrated that it works well on several (reasonably sized) benchmarks, that is, that it converges to optimal behaviour as fast as a provably correct approach such as Rmax. In fact, it even converges much faster, because of the generalization induced by knowing that the effect distribution is conditioned on the values of k variables only (Lesner and Zanuttini, 2010a, in French).

We nevertheless leave this study as a perspective for future work. Indeed, it still has to be tested on greater state spaces and more challenging problems than the one used for our preliminary experiments. Moreover, as for other factored approaches, it is still not clear whether planning can be performed efficiently (using the factored model learnt) as soon as enough statistics about the current state have been gathered. We refer the reader to the discussion about this problem by Walsh et al. (2009).

To conclude, note that a natural variant of the approach outlined above is a *sound but incomplete* approach (instead of a complete but unsound version). For this, we should use an *upper bound* on the variance of the sets of observations, rather than a lower bound. This would correspond to a *pessimistic* approach, where we assume that the variance of the sets of observations is the worst one over all candidate distributions. This approach still has to be experimented with, but observe that it requires to *maximize* the objective of the linear program of Section 6.6 over variables p_e's. Because these variables occur in maxima in the objective, the program becomes a *mixed integer program*, with as many 0/1 values as candidate effects. Hence computation will certainly be much slower than with the "optimistic" approach.

6.9 Perspective: Reinforcement Learning of PSOs

Putting together all the analyses and results in this chapter, we raise the perspective of a complete approach to the reinforcement learning problem for actions described as Probabilistic STRIPS Operators. By this, we mean the reinforcement learning problem with sample complexity measured as a function of the size of the smallest descriptions of actions as PSOs.

As already mentioned, Walsh et al. (2009) proposed an approach for this problem when the set of all effects is given (and raised the problem of handling unknown effects). Hence our results

might complement theirs by allowing the learner to identify the correct set of effects (or at least its k-closure).

Things are not so obvious, however. We first have to extend our approach for identifying the effects (Section 6.7.3) to the reinforcement learning setting. As already evoked, our formalization with partially specified state oracles might give hints about how to design an exploration strategy. Second, even with an effective exploration strategy designed, we would still have to be able to *implement it*. Indeed, knowledge of the "real" effects does not entail knowledge of their relative probabilities. Our approach for computing most likely distributions (Section 6.5) may help, but only in a heuristic manner. Hence it may be difficult for a learner to have an accurate model of the portion of the state space already explored, and hence to plan in order to implement a given exploration strategy.

A possible approach to this problem, however, is to extend our results about the identification of effects to the KWIK framework (independently from the reinforcement learning setting), and then use generic results as given by Li et al. (2011) to combine such results with KWIK algorithms for learning the probabilities of *known* effects, into a global approach to this problem. We leave this as an important perspective of our study.

Chapter 7

Learning the User's Preferences

Contents

This chapter explores issues related to agents interacting with humans. We consider agents which *serve* the user, in the sense that they act with the overall goal of satisfying their preferences at best. Sticking to our general concern of designing agents which are not specifically tailored to a given user, we consider agents which must *learn* the user's preferences. Moreover, motivated by the fact that it is easier for humans to compare objects to each other than to give each of them a numerical value, and that for many applications an ordinal ranking of the objects is enough, we are interested here in *symbolic* preferences, in particular in representations by *conditional preference networks* (CP-nets).

We first describe some joint work with Frédéric Koriche about learning such networks by observing and querying the user (Koriche and Zanuttini, 2009, 2010). We then present two research projects, one about more natural interaction protocols, and one about interleaving decision and preference elicitation. This last project benefits from very useful discussions with Laëtitia Matignon and Abdel-Illah Mouaddib. Preliminaries about the concepts used in this chapter are given in Sections 2.3 (Page 12) and 3.2 (Page 25).

7.1 Protocols for Learning Ordinal Preferences

Recall from Section 2.3 that a preference relation on a set of objects is a binary relation $\succ$ on this set, which is transitive and irreflexive. We will be concerned here with preference relations over combinatorial sets of objects, namely, we will compare to each other *assignments* μ, μ' to a set of propositional variables X.

Our study will focus on preference relations which can be represented by conditional preference networks (CP-nets, Boutilier et al. 2004). Recall that a CP-net N (on X) is given by a directed graph G with set of vertices X, and for each variable $x \in X$, a conditional probability table (CPT)

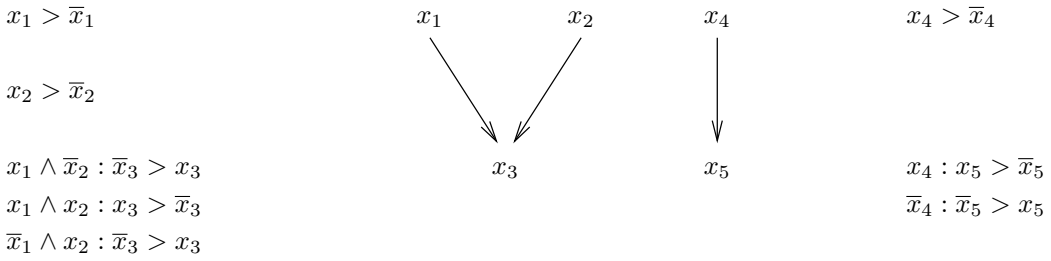


Figure 7.1: An example CP-net

which tells whether x is preferred to $\overline{x}$, or the converse, for each term t over the parents $Pa(x)$ of x in the dependency graph G.

Two objects μ, μ' which differ only on x, called *swaps* (on x), are ordered according to the preference expressed by the CPT on the values of x given the assignment to $Pa(x)$ in μ, μ' (which is the same by assumption). The relation $\succ_N$ induced by the CP-net N is defined to be the transitive closure of this set of preferences on swaps. Also recall that this relation may not be irreflexive (hence not be a preference relation), but that it is always so if the dependency graph G is acyclic. Finally, we also consider *incomplete* CP-nets, that is, CP-nets for which not all variables x need have a CPT, and the CPT for a variable x needs not give a preference on $x, \overline{x}$ for all terms over $Pa(x)$. If such a preference relation is missing, corresponding swaps cannot be ordered by the CPT.

These definitions are illustrated by the following running example.

Example 7.1 *Consider the CP-net N depicted on Figure 7.1. This CP-net is acyclic but not tree-structured (x_3 has two parents). Moreover, it is incomplete, since there is no CP-rule for x_3 and the term $t = \overline{x}_1 \wedge \overline{x}_2$ over $Pa(x_3) = \{x_1, x_2\}$.*

For learnability issues, we place ourselves in the setting where the user has in mind a preference relation $\succ$ over the assignments to X, and this relation is hidden to the learner. Naturally, we see $\succ$ as a *concept*, for which the *positive examples* are the couples of assignments $(\mu, \mu') \in 2^X \times 2^X$ for which $\mu \succ \mu'$ holds, and the *negative examples* are the couples $(\mu, \mu') \in 2^X \times 2^X$ for which $\mu \succ \mu'$ does not hold (also written $\mu \not\succ \mu'$). It is very important to keep in mind that $\mu \not\succ \mu'$ is *not* equivalent to $\mu' \succ \mu$: both assignments may be incomparable. On the other hand, $\mu \succ \mu'$ *does* entail $\mu' \not\succ \mu$. This is by the definition of a preference relation as a transitive and irreflexive relation.

Example 7.2 (continued) *Let $\succ$ be the relation represented by the CP-net N depicted on Figure 7.1. Then* $(11111, 10001)$ *is a positive example for $\succ$. On the other hand,* $(00100, 00000)$ *is a negative example for $\succ$, since these assignments are not comparable to each other, and so is* $(10001, 11111)$*, since the converse preference* $11111 \succ 10001$ *holds.*

With these definitions in hand, we can directly adapt the basic definitions of concept learning (with queries). In particular, a *membership query* (or *dominance query*) to the user, written $MQ(\mu, \mu')$, concerns two assignments μ, μ' and prompts the user to answer "yes" if $\mu \succ \mu'$ holds, and "no" if $\mu \not\succ \mu'$ holds. Given a class $\mathcal{R}$ of representations of preference relations over X (in our case, a subclass of the class of all CP-nets), an *equivalence query* to the user, written $EQ(\hat{N})$, concerns a hypothesis $\hat{N} \in \mathcal{R}$, and prompts the user to answer "yes" if the preference relation $\succ_{\hat{N}}$ represented by $\hat{N}$ is precisely $\succ$, and to give a counterexample (μ, μ') otherwise. Counterexamples (μ, μ') such that $\mu \succ \mu'$ holds (and hence $\mu \not\succ_{\hat{N}} \mu'$ holds) are called *positive counterexamples*, and counterexamples (μ, μ') such that $\mu \not\succ \mu'$ holds (and hence $\mu \succ_{\hat{N}} \mu'$ holds) are called *negative counterexamples*.

We will also use queries *restricted to swaps*, that is, to couples (μ, μ') such that μ and μ' differ on exactly one variable. Observe that restricting the *membership* queries to swaps makes the *learner*'s task more difficult, since it properly restricts the queries it is allowed to ask, while so restricting *equivalence* queries is more demanding for the *user*, since it requires that she finds a *swap* counterexample instead of an arbitrary one.

Example 7.3 (continued) *Consider again the relation $\succ$ represented by the CP-net N on Figure 7.1. The answer to the membership query $MQ(11111, 10001)$ is "yes", meaning that the user prefers the assignment* 11111 *to the assignment* 10001*. The query $MQ(00100, 00000)$ is a membership query on swaps and its answer is "no". Finally, the answer to the query $MQ(10001, 11111)$ is also "no".*

Now for equivalence queries, consider the hypothesis $\hat{N}$ which is a separable CP-net (that is, $Pa(x_i) = \emptyset$ for $i = 1, \ldots, 5$), and in which the CP-rule for all $i = 1, \ldots, 5$ is $x_i > \overline{x}_i$. Then the answer to the equivalence query $EQ(\hat{N})$ may be $(00100, 00000)$, which is a swap negative counterexample to the query (because $00100 \succ 00000$ does not hold while $00100 \succ_{\hat{N}} 00000$ does hold). The answer may also be $(01011, 01110)$, which is a nonswap positive counterexample.

Like for concept learning, we define the notion of a *query learning* algorithm.

Definition 7.4 (learning preference relations) *Let $\mathcal{C}$ be a class of preference relations, let $\mathcal{R}$ be a class of representations for preference relations, and let $s : \mathcal{R} \to \mathbb{N}$ be a function measuring the size of representations. An algorithm L is a* query learning algorithm *for $\mathcal{C}$ by $\mathcal{R}$ if there are two polynomials p_t and p_q such that for any concept N^* in $\mathcal{C}$ over n variables, after $p_q(n, s(N^*))$ queries and total running time in $O(p_t(n, s(N^*)))$, L outputs a representation $\hat{N} \in \mathcal{R}$ such that the preference relation $\succ_{\hat{N}}$ represented by $\hat{N}$ is exactly $\succ$.*

In the following, we will make explicit some conditions which we impose on the queries used and on the polynomial p_q whenever needed.

It is instructive to figure out what kind of interaction equivalence and membership queries formalize in real-world scenarios. Recall from the discussion in Section 3.2 that passive queries formalize *passive observation* of the user by the learner. A concrete realization of this in the setting of ordinal preference relations is as follows.

Assume that at some point, the learner has a hypothesis about the user's preference relation on a set of objects, say, on the dates proposed for a given meeting. Then the learner can present the user a linear ordering of these dates (or of the k dates which it thinks are most preferred, for some "reasonable" k), from the one which it thinks she will prefer down to the one which it thinks she will like the least. For hypotheses represented as CP-nets, finding such an ordering is not well-defined, because the induced preference relation is a *partial* order, but an ordering can be found efficiently, which ranks no assignment μ lower than a less preferred assignment μ' (Boutilier et al., 2004). Formally, the ordering so computed satisfies that if $\mu \succ \mu'$ holds, then μ is ranked higher than μ'.

Given such a suggested ordering, the user can be prompted to choose her preferred date. If the hypothesis is correct (at least, about the most preferred date), then she will choose the highest ranked one (denote it by μ_1). Dually, assume that she chooses, say, the one in the second place (μ_2). Because the easiest choice for her was the first presented, this can be interpreted as her preferring this second date to the first one. Thus this provides a positive counterexample to the learner's hypothesis. Moreover, in case $\mu_1 \succ_{\hat{N}} \mu_2$ held according to the hypothesis $\hat{N}$ (instead of simply $\mu_2 \not\succ_{\hat{N}} \mu_1$), this choice also provides a *negative* counterexample.

It is certainly easier to figure out how membership queries can be implemented in real contexts, since they directly correspond to questions asked to the user about her preference among two objects. Nevertheless, we raise some issues about such queries, and corresponding perspectives for future work, in Section 7.5.

7.2 Related Work

A recent book has appeared on the general problem of learning preferences (Fürnkranz and Hüllermeier, 2010), which can serve as a reference for the interested reader, hence we only report on the works closest to our setting here.

Very few works have addressed the problem of learning purely ordinal preference relations so far[1]. In the passive setting, Lang and Mengin (2009) consider the special case of *separable* CP-nets, that is, preference networks in which all variables have an empty set of parents. Their main result is that the problem of finding a separable CP-net which is consistent with a given set of examples can be solved in time polynomial in the number of variables, which entails PAC-learnability of this class since its VC-dimension is polynomial (Chevaleyre et al., 2010).

In a different setting, Dimopoulos et al. (2009) investigate the learnability issue of acyclic (complete) CP-nets under specific distributions. An assumption of their work is that the learner receives examples which are *transparently entailed* by the target preference relation $\succ$ (we refer the reader to the paper for the definition). Under this assumption, they show that acyclic CP-nets are identifiable with a time complexity of $\Theta(n^k)$, where k is the in-degree of a CP-net representing $\succ$. It can be shown that swap examples, as we use in this chapter, are always transparently entailed by acyclic, complete CP-nets, and hence their result complements the one which we present in Section 7.4.2 by assuming a bounded degree but using only examples (no active queries).

Finally, in the active setting, a work close to ours is due to Sachdev (2007), who investigates the general issue of learning preference logics with equivalence and membership queries. However, although encouraging, his results do not take computational cost into account, and the query complexity of his algorithms is exponential in the size of the target relation.

7.3 The Impossibility to Learn CP-Nets from Observations Only

We first show that a learner L cannot exist for some concept classes $\mathcal{C}$ of CP-nets, which uses (proper) equivalence queries only. In other words, these classes of CP-nets cannot be learnt from a polynomial number of queries without querying the user.

The proof uses the notion of *approximate fingerprints*, developed by Angluin (Angluin, 1990). Informally, a concept class $\mathcal{C}$ which has approximate fingerprints allows an adversary to provide highly noninformative counterexamples to equivalence queries asked by the learner. Here, "noninformative" means that the counterexample provided allows the learner to rule out only a superexponential fraction of the set of hypotheses. Hence, after only a polynomial number of equivalence queries the learner cannot have ruled out all but one hypotheses, as would be required for exactly identifying the target concept.

Formally, a concept class $\mathcal{C}$ has *approximate fingerprints* if there is some subset $\hat{\mathcal{C}}$ of $\mathcal{C}$ such that for all polynomials p and for sufficiently many variables n, there are at least two hypotheses in $\hat{\mathcal{C}}$, and for all hypotheses $\hat{c} \in \mathcal{C}$ of size at most $p(n)$, the adversary can generate a counterexample (μ, μ') to $\hat{c}$ which is also a counterexample to a fraction less than $1/p(n)$ of the hypotheses in $\hat{\mathcal{C}}$ (that is, there are less than $1/p(n)$ of the hypotheses $\hat{c}' \in \hat{\mathcal{C}}$ such that $\mu \succ_{\hat{c}'} \mu'$ if the counterexample is negative, or such that $\mu \not\succ_{\hat{c}'} \mu'$ if the counterexample is positive).

Example 7.5 (adapted from Angluin 1990, Section 3) *Let $\hat{\mathcal{C}} = \mathcal{C}$ be the set of all terms t with $Vars(t) = X$. Then any* negative *example $\mu \in 2^X$ is an approximate fingerprint of $\hat{\mathcal{C}}$. This is because there is exactly one term for which μ is* not *a counterexample, that is, $2^X - 1$ hypotheses in $\hat{\mathcal{C}}$, over 2^X, are consistent with the fact that μ is a negative example (only one hypothesis over 2^X is removed).*

[1] It is interesting to note that a entire session was dedicated to learning CP-nets at the IJCAI 2009, with three independent works presented, while to the best of our knowledge not a single article had been previously published on this topic in any international journal, conference, or workshop.

Using this notion, we were able to prove the following (the reader is referred to the published article for the proofs). Observe that the result tells impossiblity for *proper* equivalence queries. Said otherwise, nothing prevents *a priori* a query-efficient learner to use only equivalence queries, but such a learner would necessarily use a class of hypotheses different from the target class.

Proposition 7.6 (Koriche and Zanuttini 2010, Theorem 1, Corollary 1) *The class of all complete acyclic CP-nets has approximate fingerprints, and hence it is not learnable with proper equivalence queries only. The result holds even if the user is required to give counterexamples which are swaps.*

Proposition 7.7 (Koriche and Zanuttini 2010, Theorem 3, Corollary 2) *The class of all tree-structured (possibly incomplete) CP-nets has approximate fingerprints, and hence it is not learnable with proper equivalence queries only.*

Observe that Proposition 7.6 does not imply that the class of all *possibly incomplete* acyclic CP-nets is not learnable with equivalence queries only. It might be the case that allowing the user to provide hypotheses in this larger class gives it strictly more power, even if the target class itself is larger. We leave this as an open question, but conjecture that this class is not learnable either with equivalence queries only. Similarly, we conjecture that the class of complete tree-structured CP-nets is not learnable with equivalence queries only, though this cannot be a direct consequence of Proposition 7.7.

Open Question 4 *Are complete acyclic CP-nets efficiently learnable with (improper) equivalence queries only? Same question for possibly incomplete tree-structured CP-nets.*

Open Question 5 *Is the class of all possibly incomplete acyclic CP-nets efficiently learnable with equivalence queries only?*

7.4 Learning CP-Nets with Active Queries

We have seen in Section 7.3 that the classes of acyclic and tree-structured CP-nets cannot be learnt in polynomial time with equivalence queries only. We now describe algorithms which learn them with the additional help of membership queries. Moreover, these algorithms are *attribute-efficient* and *nearly optimal*.

7.4.1 Determining Parents

Both algorithms use a common subroutine which constitutes their core. This routine allows the learner to identify the *structure* of the target CP-net, namely, to identify relevant parents of variables. The routine is called with some variable x for which a parent will be identified, and with two assignments μ_p, μ_n such that $\mu_p[x/1] \succ \mu_p[x/0]$ but $\mu_n[x/1] \not\succ \mu_n[x/0]$ hold, that is, the couples $(\mu_p[x/1], \mu_p[x/0])$ and $(\mu_n[x/1], \mu_n[x/0])$ are a positive and a negative example of the target concept N^*, respectively. More precisely, they are examples of the preference relation induced by N^* on the values of x. We will soon see how such examples are acquired by the algorithms.

This routine is a variant of a technique by Blum et al. (1995). It identifies a parent for x in the target concept N^* using the following chain of assignments from of μ_p to μ_n (where p_i denotes the value assigned by μ_p to x_i, n_i the value assigned by μ_n to x_i, and the value of x is omitted):

$$\begin{aligned}
\mu_p = \mu_0 &= p_1 p_2 p_3 \dots p_{n-1} p_n \\
\mu_1 &= n_1 p_2 p_3 \dots p_{n-1} p_n \\
\mu_2 &= n_1 n_2 p_3 \dots p_{n-1} p_n \\
&\dots \\
\mu_{n-1} &= n_1 n_2 n_3 \dots n_{n-1} p_n \\
\mu_n &= n_1 n_2 n_3 \dots n_{n-1} n_n
\end{aligned}$$

Because of $\mu_0[x/1] \succ \mu_0[x/0]$ but $\mu_n[x/1] \not\succ \mu_n[x/0]$, there must be an index i in this chain which satisfies $\mu_i[x/1] \succ \mu_i[x/0]$ but $\mu_{i+1}[x/1] \not\succ \mu_{i+1}[x/0]$, and it can be immediately deduced that the only variable which changes values between both, namely x_{i+1}, is a parent of x in the target (acyclic) CP-net.

Example 7.8 (continued) *We still consider the CP-net N on Figure 7.1. Searching a parent for $x = x_5$ can be done using $\mu_p = 10111$ (because $10111[x_5/1] = 10111 \succ 10110 = 10111[x_5/0]$ holds) and $\mu_n = 00001$ (because $00001 \not\succ 00000$ holds, in fact even $00000 \succ 00001$ holds). Writing down the chain of assignments and the corresponding preferences, we get:*

$$\begin{array}{rcl} \mu_p = 10111 & \succ & 10110 \\ 00111 & \succ & 00110 \\ 00011 & \succ & 00010 \\ \mu_n = 00001 & \not\succ & 00000 \end{array}$$

and we can deduce that x_4 is a parent of x_5.

Our routine performs exactly this search, using a membership query for deciding $\mu_i[x/1] \succ \mu_i[x/0]$ for all $i = 1, \ldots, n$. Using a dichotomic search in this sequence, we get a procedure which correctly identifies a parent of some variable x using a number of queries which is logarithmic in the number of variables (attribute-efficiency will come from this property).

7.4.2 Learning Acyclic CP-Nets

We now give a learner for the class of all acyclic CP-nets. The only restriction is that the user is required to provide *swap* counterexamples to negative queries. We conjecture that it is impossible to remove this hypothesis; indeed, in this case the learner could not even decide efficiently whether the counterexample provided satisfies its current hypothesis, that is, it could not distinguish positive from negative counterexamples in polynomial time, assuming P $\neq$ NP (Domshlak and Brafman, 2002).

Our algorithm incrementally builds both the dependency graph and the conditional preference tables, using the counterexamples received for its equivalence queries (as soon as an equivalence query is answered "yes", learning has succeeded with the current hypothesis). At any moment, the dependency graph is a subgraph of the graph G in the target CP-net N^*, that is, at any moment and for all variables, the learner uses a set of parents included in or equal to the correct one[2]. Initially, the graph is empty.

When it receives a positive counterexample of the form $\mu[x/\ell] \succ \mu[x/\overline{\ell}]$, where ℓ is a literal over variable x, the learner builds a new rule for taking it into account. Namely, it uses the current set of parents $Pa(x)$ for x, and adds to its current hypothesis the rule $\mu_{|Pa(x)} : \ell > \overline{\ell}$. In this manner, the current hypothesis takes into account at least this counterexample. Most importantly, the learner stores the *context assignment* μ together with the new rule.

Example 7.9 (continued) *We still consider the CP-net N depicted on Figure 7.1. Assume that the agent currently believes $Pa(x_3) = \{x_2\}$ and receives the positive counterexample* $(11111, 11011)$. *Then it adds the rule $x_2 : x_3 > \overline{x}_3$ to its hypothesis, and stores* 11111 *as the context assignment of this rule.*

In general, such positive counterexamples $\mu[x/\ell] \succ \mu[x/\overline{\ell}]$ may come out for one of two reasons: either the learner's hypothesis was previously unable to compare $\mu[x/\ell]$ to $\mu[x/\overline{\ell}]$, or it entailed the converse preference $\mu[x/\overline{\ell}] \succ \mu[x/\ell]$. In the former case, no more work has to be done, but in the latter case the learner now must "revise" the rule which prescribed the wrong preference. This is done exactly as if the *negative* counterexample $(\mu[x/\overline{\ell}], \mu[x/\ell])$ had been received (meaning

[2]Precisely, in the smallest correct one, which is guaranteed to be unique for acyclic CP-nets (Koriche and Zanuttini, 2010, Lemma 2).

$\mu[x/\overline{\ell}] \not\succ \mu[x/\ell]$). Note that this is indeed a negative example, since $(\mu[x/\ell], \mu[x/\overline{\ell}])$ is a positive one.

We now describe how such negative counterexamples $(\mu[x/\overline{\ell}], \mu[x/\ell])$ are handled in general. Because negative counterexamples are such that $\mu[x/\overline{\ell}] \succ \mu[x/\ell]$ is entailed by the current hypothesis, there must be a rule of the form $\mu_{|Pa(x)} : \overline{\ell} > \ell$. By construction, the learner must have received a previous *positive* example for this rule, namely, there is an assignment μ_p such that $(\mu_p)_{|Pa(x)} = \mu_{|Pa(x)}$ and $\mu_p[x/\overline{\ell}] \succ \mu_p[x/\ell]$ holds in the target concept N^*. Moreover, still by construction such a context assignment μ_p has been stored together with the rule.

Hence we have two assignments μ_p and $\mu_n = \mu$ which are a positive and a negative example, respectively, of the same conditional rule. Because the target concept is known to be acyclic, it follows from this that some parent of x is missing, precisely one which discriminates both assignments from each other. At this point the algorithm simply calls the routine described in Section 7.4.1 for identifying such a parent x'. Then it expands the graph, and it expands the body of each rule in the conditional table of x with this variable. Precisely, it replaces each rule of the form $t : \ell > \overline{\ell}$, where ℓ is a literal over x, with the rule $t \wedge \ell' : \ell > \overline{\ell}$, where ℓ' is the value of the new parent x' in the context assignment stored for the rule. In this manner, the invariant is maintained that each rule is built on the same set of parents and is supported by its context assignment.

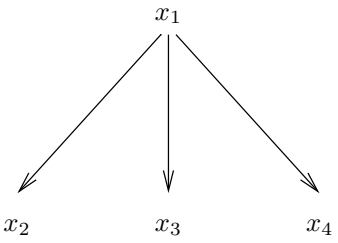
Example 7.10 (continued) *Assume as in Example 7.9 that the agent believes $Pa(x_3) = \{x_2\}$, and assume that its hypothesis currently contains the CP-rules $x_2 : x_3 > \overline{x}_3$ (with context assignment 11111) and $\overline{x}_2 : \overline{x}_3 > x_3$ (with context assignment 10000). If the agent receives the negative example $(01111, 01011)$, this contradicts its first rule above. Then using $\mu_p = 11111$ (context of the guilty rule) and $\mu_n = 01111$ (example just received), the agent finds that x_1 is also a parent of x_3. Then it updates the first rule above to $x_1 \wedge x_2 : x_3 > \overline{x}_3$ (because the context of this rule assigns 1 to x_1), and similarly it updates the second rule above to $x_1 \wedge \overline{x}_2 : \overline{x}_3 > x_3$.*

The algorithm is depicted on Figure 3. We state its correctness and complexity without proof

Algorithm 3: Learner for acyclic CP-nets

```
N̂ ← ∅
while EQ(N̂) ≠ "yes" do
    let (μ[x/ℓ], μ[x/ℓ̄]) be the counterexample returned by EQ(N̂)
    if (μ[x/ℓ], μ[x/ℓ̄]) is a model of some rule r in N̂ then
        /* The counterexample is negative */
        let μ_r be the context assignment of r
        x' ← a new parent of x (using μ_r and μ)
        Pa(x) ← Pa(x) ∪ {x'}
        expand the body of each rule r' with context assignment μ_r' in the table of x with
        the literal of x' in μ_r'
    else
        /* The counterexample is positive */
        add the rule μ_Pa(x) : ℓ > ℓ̄ to N̂ with context assignment μ
        if (μ[x/ℓ̄], μ[x/ℓ]) is a model of some rule r in N̂ then
            let μ_r be the context assignment of r
            x' ← a new parent of x (using μ_r and μ)
            Pa(x) ← Pa(x) ∪ {x'}
            expand the body of each rule r' with context assignment μ_r' in the table of x
            with the literal of x' in μ_r'
return N̂
```

($s(N^*)$ denotes the size of the target concept — total number of occurrences of values in rules, e



$\overline{x}_1 > x_1$

$x_1 : \overline{x}_2 > x_2$
$\overline{x}_1 : x_2 > \overline{x}_2$

$x_1 : \overline{x}_3 > x_3$
$\overline{x}_1 : x_3 > \overline{x}_3$

$x_1 : \overline{x}_4 > x_4$
$\overline{x}_1 : x_4 > \overline{x}_4$

Figure 7.2: An example tree-structured CP-net

denotes the number of edges in the dependency graph G, and $n = |X|$ denotes the number of variables).

Proposition 7.11 (Koriche and Zanuttini 2010, Theorem 2) *Acyclic CP-nets are attribute efficiently identifiable, when the user is restricted to answer equivalence queries with swap counterexamples. Algorithm 3 uses at most $s(N^*) + 1$ equivalence queries and $e\lceil \log n \rceil$ membership queries (also restricted to swaps).*

7.4.3 Learning Tree-Structured CP-Nets

Our algorithm for learning tree-structured CP-nets exploits the same ideas as for acyclic CP-nets, but takes advantage of the fact that per definition, there is at most one parent to learn for each variable. This makes it possible to learn tree-structured CP-nets with *arbitrary* equivalence queries, that is, the user may give any counterexample to them, while still asking membership queries on swaps only.

Recall that we do not know *a priori* which variables are relevant for describing the target CP-net. Our learner uses counterexamples to equivalence queries to identify these, using the observation that if $\mu \succ \mu'$ holds in the target CP-net (positive counterexample), then each variable with a different value in μ, μ' must have at least one rule, hence be relevant. The algorithm then handles each of these variables in turn, using membership queries to find its parent (if any) and its CP-table. Then, so as to be consistent with the counterexample received, it recurses on the parents found. It can be easily shown that considering only these variables is enough for the resulting CP-net to entail the positive example received. Moreover, by construction the hypothesis maintained by the learner will always entail a subset of the target preference relation, so that no negative counterexample can ever be received.

Now given a variable x, in order to find the parent of x the learner uses the following observation. If x' is the parent of x, then in the target concept we must have $\mu_{\vec{0}}[x/1] \succ \mu_{\vec{0}}[x/0]$ and $\mu_{\vec{1}}[x/1] \not\succ \mu_{\vec{1}}[x/0]$, or vice-versa, where $\mu_{\vec{0}}$ assigns all variables to 0, and $\mu_{\vec{1}}$ assigns all variables to 1. Using membership queries this can be worked out. If the preference over values of x is the same in $\mu_{\vec{0}}$ and $\mu_{\vec{1}}$, then x has no parent, and otherwise the routine for finding parents can be used with $\mu_p = \mu_{\vec{0}}$ and $\mu_n = \mu_{\vec{1}}$, or vice-versa. Once the parent found, since there are at most four rules on x, all of these can be determined using a constant number of membership queries.

Example 7.12 (tree-structured CP-nets) *Consider the tree-structured CP-net N depicted on Figure 7.2, and assume that the agent receives the positive example* $(1001, 1111)$. *Then it computes the CP-tables for x_2 and x_3. For x_2 it asks the question $MQ(0000[x_2/1], 0000[x_2/0])$, i.e., $MQ(0100, 0000)$, and similarly the question $MQ(1111, 1011)$. The first query is answered "yes" and the second is answered "no", so the agent knows that x_2 has a parent. Using $\mu_p = 0000$*

and $\mu_n = 1111$ it finds x_1. Then it asks the queries $MQ(0100, 0000)$ (which is answered "yes"), $MQ(0000, 0100)$ ("no"), $MQ(1111, 1011)$ ("no"), and $MQ(1011, 1111)$ ("yes"). From the answers it can then conclude that the CP-table for x_2 is $\{x_1 : \overline{x}_2 > x_2, \overline{x}_1 : x_2 > \overline{x}_2\}$. Then it goes on with x_3 and x_1 (recursive call).

The algorithm is depicted on Figure 4. Again, we state its correctness and complexity without

Algorithm 4: Learner for tree-structured CP-nets

$\hat{N} \leftarrow \emptyset$
while $EQ(\hat{N}) \neq$ "yes" do
  let (μ, μ') be the (positive) counterexample returned by $EQ(\hat{N})$
  for each x with different value in μ, μ' do HANDLEVARIABLE(x)
end
return $\hat{N}$

Procedure HANDLEVARIABLE(x)

if x has already been handled then return
foreach $\mu \in \{\mu_{\vec{0}}, \mu_{\vec{1}}\}$ and $\ell \in \{x, \overline{x}\}$ do $pref(\mu, \ell) \leftarrow MQ(\mu[x/\ell], \mu[x/\overline{\ell}])$
if $pref(\mu_{\vec{0}}, \ell) = pref(\mu_{\vec{1}}, \ell) =$ "yes" for some $\ell \in \{x, \overline{x}\}$ then
  $Pa(x) \leftarrow \emptyset$
end
else if $pref(\mu_{\vec{0}}, \ell) =$ "yes" for some $\ell \in \{x, \overline{x}\}$ then
  $Pa(x) \leftarrow$ the parent of x (using $\mu_{\vec{0}}, \mu_{\vec{1}}$)
end
else if $pref(\mu_{\vec{1}}, \ell) =$ "yes" for some $\ell \in \{x, \overline{x}\}$ then
  $Pa(x) \leftarrow$ the parent of x (using $\mu_{\vec{0}}, \mu_{\vec{1}}$)
else
  $Pa(x) \leftarrow \emptyset$
end
add to $\hat{N}$ a table for x over $Pa(x)$ and with the rules validating the answers $pref(\mu, \ell)$
if $Pa(x) = \{x'\}$ then HANDLEVARIABLE(x')

proof ($s(N^*)$ denotes the size of N^*, m the number of relevant variables in it — number of CPTs, e the number of edges in the dependency graph G, and $n = |X|$ the total number of variables).

Proposition 7.13 (Koriche and Zanuttini 2010, Theorem 4) *Tree-structured CP-nets are attribute-efficiently identifiable. Algorithm 4 uses at most $s(N^*)+1$ (arbitrary) equivalence queries and $4m + e\lceil \log n \rceil$ membership queries (restricted to swaps).*

If attribute-efficiency is not a concern, it is interesting to note that a straightforward modification of Algorithm 4 shows that tree-structured CP-nets are learnable *from membership queries only.* Indeed, it suffices to call the routine HANDLEVARIABLE for each variable used to describe the assignments. From a practical perspective, this property allows the user to install its software agent, answer a reasonable number of questions (membership queries) in a row, and then have the agent immediately available for optimal behaviour.

However, in real settings, not being attribute-efficient means asking membership queries for each of thousands of variables even if, say, only ten are relevant. Hence we state the result only for the record.

Proposition 7.14 *The class of all tree-structured CP-nets is learnable with membership queries only, with a learner which uses $4n + e\lceil \log n \rceil$ queries (restricted to swaps).*

7.4.4 Optimality

The algorithms presented above share the interesting feature of being *attribute-efficient*, that is, they require only a polylogarithmic effort (answers to queries) from the user on irrelevant variables, without the need to know them in advance. We have also shown that their active behaviour is necessary, because CP-nets are not learnable with polynomially many (proper) equivalence queries only.

We are also able to show that they are almost optimal, in terms of numbers of queries. We do not detail the proof here, but it works by first determining the VC-dimension of particular classes of CP-nets, then using a standard result by Auer and Long (1999, Theorem 2.2). Again, the VC-dimensions (taking into account the fact that not all variables need be relevant) are obtained using a variant of a technique by Blum et al. (1995).

Proposition 7.15 (Koriche and Zanuttini 2010, Theorem 6) *The class of all acyclic CP-nets with in-degree at most k and at most e edges, over n variables, has a VC-dimension at least $VC_T = \lfloor \frac{e}{k} \rfloor (\lfloor \log \frac{n - \lfloor e/k \rfloor}{k} \rfloor + 1)$ for $k = 1$ (that is, for tree-structured CP-nets), and at least $VC_A = \lfloor \frac{e}{k} \rfloor (2^k + k(\lfloor \log \frac{n - \lfloor e/k}{k} \rfloor - 1) - 1)$ for $k \geq 2$.*

Corollary 7.16 *Any algorithm which identifies tree-structured CP-nets (resp. acyclic CP-nets) must use at least $\log \frac{4}{3} VC_T$ queries (resp. $\log \frac{4}{3} VC_A$) queries.*

It follows directly that our algorithms are almost optimal, namely, that they are optimal up to a term of order $e \log n$.

7.5 Perspective: Restricting Membership Queries

Recall from the discussion in Section 7.1 that a membership query on swaps (μ, μ') (say, with $\mu \models x$ and $\mu' \models \overline{x}$) can be viewed as asking the user whether in the context of $\mu_{|X \setminus \{x\}} = \mu'_{|X \setminus \{x\}}$, she prefers x to $\overline{x}$. Even more naturally, this can be viewed as asking whether the user would prefer the *object* μ with its attribute x changed to $\overline{x}$.

This is fine if all objects μ make sense for the user. For instance, assume that the learner sells kitchens, and wants to elicitate a customer's preferences on various combinations of elements. In this case, typically it will be able to present the user a 3D-view of any combination μ of elements using his favourite software, and to ask a membership query on (μ, μ') by presenting the view of μ, then that of μ'. Chances are that the costumer easily expresses her preferences using these views.

Now assume to the contrary that the learner sells music albums. Clearly, it will be able to ask a membership query on objects such as "a French group playing Blues with electric guitars in the Nineties" and "an English group playing Rock'n'Roll with electric guitars in the Seventies", by presenting an example of each. But clearly, not all combinations of attributes can be so easily presented to the user (think of the combination μ corresponding to "an Italian singer singing Rap with beatbox in the 19th Century"). However, our algorithms *a priori* suffer no restriction on the membership queries that they can ask (apart from being on swaps).

In the general case, these examples call for algorithms which are able to elicitate preferences using membership queries only on objects μ, μ' taken from a given database $\mathcal{D}$, or which are solutions of a given set of constraints.

An important comment on this problem is that two alternative formalizations are reasonable. In the first one, the learner is required to elicitate the preferences which concern precisely the given set of objects, that is, to identify the relation $\succ_{|\mathcal{D}}$ induced by $\succ$ on the pairs of objects $(\mu, \mu') \in \mathcal{D} \times \mathcal{D}$. In the second formalization, specific to CP-nets, it is required to elicitate the CPT entries which are witnessed by at least two assignments $\mu, \mu' \in \mathcal{D}$ forming a swap. As the following very simple example shows, these two problems are different from each other in general.

Example 7.17 *Consider the (separable) CP-net with rules $x_1 > \overline{x}_1$, $x_2 > \overline{x}_2$, and $x_3 > \overline{x}_3$, and the database of objects $\mathcal{D} = \{000, 001, 111\}$. Then in the first formalization above the agent must*

learn $111 \succ 001 \succ 000$. *Contrastingly, in the second formalization above it must learn* $x_3 > \overline{x}_3$, *because this is witnessed by* $(001, 000)$ *in* $\mathcal{D}$. *Observe that in the second formalization the learner will not be able to compare* 111 *to* 001, *nor* 111 *to* 000, *but it will be able to compare* 101 *to* 100, *even while these assignments are not in* $\mathcal{D}$.

Note that the second formalization yields a problem which is easier for the learner, in the sense that all CPT entries that must be learnt in this setting must also be learnt in the setting induced by the first formalization (if preferences are learnt as a CP-net).

We also want to point out that this problem of restricting membership queries to a database of objects (given in extension or in intension) is not specific to preference elicitation, and can be formulated for the general problem of concept learning. For some insights into this generic problem in a setting different from ours, the reader is referred to Boutilier et al. (2010).

7.6 Perspective: Learning Preferences while Acting

Assume you just installed your new software agent, which is designed (and sold) for helping you to organize your daily tasks, meetings, and more generally your schedule. You will certainly accept that during an initial period the agent asks you some questions about your preferences so that it gets more helpful to you. On the other hand, you will certainly not accept that after several years monitoring your actions and asking questions to you from time to time, it still takes appointments for you which do not suit you at all, or keeps on asking questions to you every day. Dually, you will certainly not accept that it asks you questions during a whole week at the time you install it, even if, it claims, this will allow it to make only very few mistakes thenafter.

What you will probably prefer is that installation takes only a few minutes, possibly that you answer a few questions at this time, and that only during the few first months the agent asks questions (from time to time) and makes some mistakes about your preferences (from time to time as well). And at any time during this kick-off period, you would certainly appreciate if the agent was already able to make decisions which suit you, even if on the easiest tasks only.

What this calls for is agents which are able to elicitate the user's preferences only when needed, while able to act at any moment according to the preferences elicited so far. What this also calls for is agents which are able to converge reasonably fast to an optimal behavior according to *all* the (initially hidden) user's preferences.

We have seen in Section 7.4 that it is indeed possible for an agent to elicitate the user's preferences efficiently, and hence to converge reasonably fast to full knowledge of them, while asking a reasonable number of questions (ant not all at once). We also know of algorithms which allow a user to make optimal decisions when the dynamics of the environment and the (numerical) preferences on states or actions are known.

The question now is whether it is possible to design techniques for coupling such optimal strategies. We believe that the most natural formalization of this problem is as a reinforcement learning problem.

More precisely, in our proposal formalization there is an underlying MDP defined by the set A_o of *ontic actions* available to the agent, that is, a set of actions which provoke changes on the environment. We may assume that the transition probabilities between states under each ontic action are known to the agent, as well as the costs and rewards associated to taking these actions.

What is initially unknown to the agent is the *reward function* R on the states of this underlying MDP, precisely because it is defined by the user's preferences on states. The states are thus defined to contain all the information necessary to compare decisions about which ontic actions to perform *and* to evaluate the user's preferences.

In this setting, the agent must be a *reinforcement learner* able to learn the reward function/user's preferences while acting and making a small number of suboptimal decisions from the very beginning. We are thus left with how this reward function can be acquired.

If the user's preferences are represented on a numerical scale, and the user is aware of this, it is reasonable to assume that upon reaching a state s, the agent will be informed of the value

assigned to s by the user. If this scale is uncommensurable with the one used for expressing rewards and costs of ontic actions, then standard techniques from multicriteria decision can be used for aggregating both and hence comparing trajectories in the underlying MDP (Boussard et al., 2010). Then the problem as formulated is a standard reinforcement problem. Factored approaches can be imagined if some structure is assumed in the representation of the user's preferences, e.g., if they are assumed to be representable by a GAI-network with bounded degree.

Things get more complicated if the user's preferences are ordinal. The first problem comes with the fact that, even if they are completely known, comparing trajectories when they are "valued" by ordinal preferences is a nontrivial problem. Weng gives interesting solutions based on rationality assumptions in case the preferences consist of a total order on the set of states, including some representation theorems (Weng, 2011). Hence we may forget this problem for the moment, as far as linear preference relations are concerned. As for partial preference relations, though some ideas by Weng may be lifted to this more general case, to the best of our knowledge there has been no proposal yet in the literature for comparing trajectories and hence, defining optimal policies.

The second problem which comes with ordinal preferences is what feedback the learner receives from the user about the reward functions. By definition, ordinal preferences do not assign a value to a single state, contrary to the case of reinforcement learning settings.

Assume for the moment that all ontic actions available to the agent are deterministic, and that the user knows that. What we propose is to study user's feedback of the form "you should have done that", or more precisely "you should have taken that action, because it would have led you to a state better than the one where you are now". More formally, we propose to formalize the user's feedback as follows.

Definition 7.18 (user's feedback for ordinal preferences) *Let $\succ$ be a preference relation on the set S of all states, let A be a set of deterministic actions, and let the function $next : S \times A \to S$ indicate to which state $s' = next(s, a)$ the environment moves from state s via action a. Given a state s and an action a, the* user's feedback for decision (s, a) *is any action $a' \in A$ satisfying $next(s, a') \succ next(s, a)$, if one exists, and "yes" otherwise.*

Example 7.19 *Assume the user prefers state s_1 to s_2, and s_2 to s_3. Moreover, assume that in some state s, the agent has three actions available, namely, for $i = 1, 2, 3$ an action a_i which leads it deterministically to s_i. If it takes action a_3, then the user's feedback can be a_1 or a_2. If it takes action a_2, then it must be a_1, and finally, if it takes action a_1 the user's feedback must be "yes".*

An additional requirement may be that the user's feedback is an action which is *nondominated* under $\succ$ among the valid feedbacks. Clearly, this would give more information to the learner. In any case, we think Definition 7.18 gives a reasonable account of what a user will typically do, that is, complain about the fact that its agent did not help her as well as it could have.

An important remark is that Definition 7.18 assumes a short-sighted user. Her complaining about the agent taking such action does not mean that the agent did not follow the optimal policy. In fact, it exactly means, by definition, that the agent did not follow the optimal policy *at horizon* 1.

With this setting in hand, the objective of this research project is to design techniques for learning the user's preferences from such user's feedback, generalize this knowledge to other, unseen states, and design exploration strategies for learning fast. We believe that an efficient learning algorithm for some class of preferences can be used to design efficient exploration strategies where, in a nutshell, membership queries would translate to promising states to explore, while counterexample to equivalence queries would translate to user's feedback on suboptimal behaviors. This however remains a widely open research project.

Chapter 8

Reasoning about the Complexity of Algorithms

Contents

This chapter makes a detour in function algebra and clone theory, precisely by discussing the use of Post's lattice by agents and proposing a related research project, concerned with the exact complexity of algorithms. More precisely, we introduce the notion of a frozen partial co-clone, to capture problems which are interreducible by reductions which preserve the number of variables, hence, to some extent, the exact complexity of algorithms. Though our main concern here is the Boolean domain, we formulate notions and results in the general, finite-domain setting as often as possible.

This notion has been introduced in a conference article (Nordh and Zanuttini, 2009), in collaboration with Gustav Nordh. The research project which follows has also been elaborated with Magnus Wahlström. The preliminaries relevant to this chapter are given in Section 2.2 (Page 9).

8.1 The Use of Post's Lattice by Intelligent Agents

Recall that Post's lattice (of co-clones) organizes all Boolean relations in classes which are closed under implementation and hence, under polynomial-time (logspace) reductions for many decision problems. As we have shown in Section 2.2, this is of particular interest for being able to give a

complete picture of the complexity of such problems, when their inputs are restricted to Γ-formulas. We now argue that Post's lattice can also be used directly by an agent.

The first possible use of Post's lattice by agents lies in using complexity classifications for deciding dynamically which piece of knowledge to use for some reasoning tasks. Indeed, assume that the agent has as its disposal several approximations of some concept c, one in each of several languages $\Gamma_1, \Gamma_2 \ldots$, as in the setting of Chapter 4. Now assume it is faced with some reasoning task for which it must use the concept c. Then, using Post's lattice and a lookup table for the complexity of the task depending on the language, the agent can decide at execution time which approximation of c to use for the reasoning task, depending on the time allowed and by the quality of inference required (the lower the language of the approximation in Post's lattice, the faster the reasoning algorithms but the less accurate the answer).

In a more general setting, assume that the agent faces a reasoning task (about the concept c) for which it has several candidate representations of c. Then, because Γ-formulas can be recognized efficiently for any finite Γ and any reasonable presentation of the constraints, the agent can find out dynamically in which language each representation falls, and decide which one to use accordingly, again depending on the imposed tradeoff between time and quality.

An agent can also use Post's lattice and lookup tables for the complexity of problems in order to decide on a language over which to *learn* its concepts. For instance, if the agent is often faced with abduction problems which it has a reasonable but not infinite time to solve, then it could decide to learn approximation of concepts in the class of, say, all Horn formulas, because this language has "only" an NP-complete abduction problem, as opposed to the general case which is Σ_2^{P}-complete.

Finally, as shown by Khardon and Roth (1996) for Horn formulas, and further generalized by del Val (1995) and Zanuttini (2002b), approximations of concepts in languages can guarantee soundness and completeness for some sets of queries. For instance, the *Horn* formulas which are entailed by a (minimal upper) Horn approximation of some concept c are exactly those which are entailed by c. It follows again that, using Post's lattice for recognizing which queries correspond to which approximation language, an agent can save an important computational effort by choosing the easiest approximation which is complete for the queries at hand.

8.2 Motivation for Studying Frozen Co-Clones

As we have argued in Section 8.1, agents which use logical knowledge can make use of Post's lattice for deciding which piece of knowledge and/or algorithm to use when facing a particular reasoning problem. However, this approach has risks not to be so useful in general, due to the fact that most implementations of languages by others do *not* preserve the number of variables n in the instance. Hence, though there may exist an algorithm for some reasoning problem with complexity $O(f(n))$ for some language Γ_1, when using implementations in Post's lattice the problem may have, for instance, complexity $O(f(n+m))$ for some other language Γ_2, where m is the size of the input formula, even though Γ_1 and Γ_2 determine the same co-clone. In general, the reductions map a conjunction of m constraints on n variables to a conjunction of bm constraints on $n+am$ variables, where a and b are constants which depend on the languages Γ_1 and Γ_2.

To be more concrete, consider the complexity of 3-SAT and 1/3-SAT. These problems are exactly the problems $\mathrm{SAT}(\Gamma_3)$ and $\mathrm{SAT}(\Gamma_{1/3})$, where Γ_3 consists of all relations corresponding to clauses on three variables or less, and $\Gamma_{1/3}$ consists of the unique relation $\{001, 010, 100\}$, which forces exactly one of three variables to be assigned 1. Clearly, in Post's lattice we have $\langle\Gamma_3\rangle = \langle\Gamma_{1/3}\rangle = \mathrm{BR}$. In particular, $\mathrm{SAT}(\Gamma_3)$ and $\mathrm{SAT}(\Gamma_{1/3})$ are polynomial-time equivalent (both turn out to be NP-complete).

However, 3-SAT is "only" known to be solvable in time $O(1.473^n)$, where n is the number of variables in the instance (Dantsin et al., 2002; Brueggemann and Kern, 2004), while 1/3-SAT is known to be solvable in time $O(1.1003^n)$ (Byskov et al., 2005). Hence, though no lower bound is formally proven for 3-SAT, it seems like 1/3-SAT is a much easier problem. However, polynomial-time reductions, and hence membership in the same co-clone in Post's lattice, do not account for this.

We try to remedy this by developping in the forthcoming section the notion of a *frozen co-clone*, which refines the notion of a co-clone.

8.3 Related Work

Several refinements of the structure defined by Post's lattice have already been proposed, which hence may be candidates for studying the exact complexity of algorithms. By "refinements", we mean that these notions define closed classes of relations of the form $\langle\Gamma\rangle_X$, which are naturally organized in a lattice, and that these notions decompose further each co-clone in Post's lattice, that is, if $\langle\Gamma_1\rangle_X = \langle\Gamma_2\rangle_X$ holds, then $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle$ also holds.

Remark 8.1 *Throughout this chapter, when we need to distinguish implementations in the sense of co-clones from other types of implementations (e.g., from the frozen implementations which we introduce), we will talk of* regular *implementations.*

8.3.1 Partial Strong Co-Clones

One of the refinements of Post's lattice is given by the notion of a *partial strong co-clone*. The partial strong co-clone of a language Γ, written $\langle\Gamma\rangle_{ps}$, is the set of all relations R such that $R(X)$ is logically equivalent to at least one conjunction of constraints from $\Gamma \cup \{R_=\}$ (over the same set of variables X). In other words, the partial co-clone of Γ is the set of all relations which can be expressed from Γ (and equality) using only conjunction. Clearly enough, when two finite languages Γ_1, Γ_2 determine the same partial strong co-clone ($\langle\Gamma_1\rangle_{ps} = \langle\Gamma_2\rangle_{ps}$), we have a translation between Γ_1-formulas and Γ_2-formulas which preserves satisfiability *and the number of variables*, thus meeting our requirements. Such reductions preserve in fact much more: the set of variables and the set of models, which makes them suited for reductions between, e.g., counting problems or circumscriptive inference problems, contrary to usual co-clones.

The problem with partial strong co-clones is that their structure is largely unknown. In particular, it turns out that there are an uncountable number of them, even on the Boolean domain (Lau, 2006, Chapter 20). Despite this, partial strong co-clones come with a nice theory. In particular, there is a Galois connection between their lattice and that of partial strong clones, which organize partially defined functions in closed classes just as (total) functions are organized into clones. Moreover, each (usual) co-clone C has at least two known bases in the sense of partial strong implementations.

The first basis is called a *weak basis*, and is one which can be expressed via partial strong implementations from any language Γ_1 satisfying $\langle\Gamma_1\rangle = C$; equivalently, which generates the smallest partial strong co-clone contained in C (and in no proper subco-clone). There is moreover a standard construction for such a weak basis, starting from any relation R with $\langle\{R\}\rangle = C$, and any such weak basis consists of only one relation. The notion and the construction have been introduced by Schnoor and Schnoor (2008).

The second basis, for each co-clone C, is called a *plain basis*, and is one which generates all of C via partial strong implementations. We introduced this notion together with a complete list of plain bases, one per co-clone, in a journal article (Creignou et al., 2008b). Observe that these bases are not finite in general.

Example 8.2 *The relation R defined by $R(x_1, x_2, x_3, x_4) \equiv (\overline{x}_1 \vee \overline{x}_2) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4)$ is in the partial strong co-clone of $\Gamma = \{R_1, R_2\}$, where R_1 is $\{0,1\}^3 \backslash \{111\}$ and R_2 is $\{0,1\}^3 \backslash \{110\}$. This is because the constraint $R(x_1, x_2, x_3, x_4)$ is logically equivalent to the Γ-formula $R_1(x_1, x_2, x_2) \wedge R_2(x_2, x_3, x_4)$.*

Now a plain basis of IE_2*, that is, the co-clone of all Horn relations, is the infinite set of all Horn clauses, since by definition a Horn relation is one which is equivalent to a Horn CNF.*

To conclude on this point, partial strong co-clones meet our requirements for preservation of the exact complexity of algorithms, but they yield a probably too fine-grained classification of languages.

8.3.2 Faithful Implementations

Another proposed classification of languages is the one induced by *faithful implementations*, a notion especially designed for counting problems. In a nutshell, a language Γ *faithfully implements* a relation R if there is a (regular) implementation of $R(X)$ by $\exists X'\varphi$, where φ is a Γ-formula over $X \cup X'$, such that *every assignment in* $R(X)$ *is uniquely extended over* X' *into a model of* φ. The link to counting problems is immediate, since clearly the number of models of φ is exactly the number of assignments in $R(X)$ (very little extra work is enough to show that this holds for *conjunctions* of constraints).

Unfortunately, faithful implementations do not preserve the number of variables in general. Moreover, they do not preserve important properties such as the one of being a minimal model.

Example 8.3 *The formula* $\exists x_4(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (x_4 \vee x_3)$ *is* not *a faithful implementation of* $(\overline{x}_1 \vee \overline{x}_2 \vee x_3)$ *(we abuse words by confounding the relation and the constraint). Indeed, the model* 001 *of the latter formula can be extended to two models of* $(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (x_4 \vee x_3)$*, namely* 0010 *and* 0011*. Contrastingly, the formula* $\exists x_4(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (x_4 \vee x_3) \wedge (\overline{x}_4 \vee \overline{x}_3)$ is *a faithful implementation of* $(\overline{x}_1 \vee \overline{x}_2 \vee x_3)$*.*

Another point to note is that, to the best of our knowledge, faithful implementations do not come (yet) with a theory of polymorphisms which would help in classifying languages according to their "faithful expressiveness". Still, they prove useful in classifying the complexity of many problems (Creignou et al., 2001).

8.4 Frozen Partial Co-Clones

We now introduce the notion of a frozen implementation, from which we will deduce that of a frozen partial co-clone. It is less demanding for an implementation to be frozen than to be "partial strong", because we allow a restricted form of existential quantification. On the other hand, it is more demanding than to be faithful, in the sense that if an implementation is frozen, then it must be faithful. This further restriction is necessary since, as we have seen, faithful implementations are not demanding enough for our complexity-preserving requirements.

8.4.1 Frozen Implementations

The basic idea is to define a language Γ_1 to freezingly implement a language Γ_2 if any relation in Γ_2 can be defined using the relations in Γ_1 and at most two additional variables, one bound to 0 and the other to 1. In this manner, when representing a Γ_2-formula as a Γ_1-formula, the auxiliary variables introduced for implementing each relation in Γ_2 can all be consistently identified to only two of them (intuitively, "false" and "true"), so that the number of variables increases by at most 2.

The definitions and results in this section are general, so we state them for languages over an arbitrary, finite domain D.

Definition 8.4 (frozen variable) *Let* φ *be a formula over a set of variables* X*, and let* $x \in X$ *be a variable. Then* x *is said to be* frozen *in* φ *if there is a value* $v \in D$ *such that* φ *entails* $(x = v)$*. In other words,* x *is frozen in* φ *if it is assigned the same value by all its models.*

Definition 8.5 (frozen implementation) *Let* Γ *be a set of relations and* R *be relation. Then* Γ freezingly implements R *if the constraint* $R(X)$ *is logically equivalent to* $\exists X'\varphi$ *for some* Γ*-formula* φ *over* $X \cup X'$ *such that every variable in* X' *is frozen in* φ*.*

In words, frozen implementations are regular implementations in which auxiliary variables must be frozen.

Example 8.6 *Consider the relations* $R = \{000, 001, 110\}$ *and* $R' = \{00, 10\}$*. Then* R' *is freezingly implemented by* $\{R, R_T\}$*, as can be seen from* $R'(x_1, x_2) \equiv \exists x_3 R(x_2, x_2, x_3) \wedge R(x_2, x_2, x_1) \wedge R_T(x_3)$ *(where* x_3 *is frozen to* 1*).*

As another example, consider $R_1 = \{01, 10\}$ *and* $R_2 = \{0100, 0110, 1000, 1111\}$*. Then* $R_3 = \{010, 011, 100\}$ *is freezingly implemented by* $\{R_1, R_2\}$*, as shown by* $R_3(x_1, x_2, x_3) \equiv \exists x_4 R_1(x_1, x_2) \wedge R_2(x_1, x_2, x_3, x_4)$ *(*x_4 *is frozen to* 0 *by the conjunction).*

8.4.2 Frozen Partial Co-Clones

We now introduce the notion of a frozen partial co-clone, and show that these co-clones are organized in a lattice.

Definition 8.7 (frozen partial co-clone) *Let* Γ *be a language. The* frozen partial co-clone *generated by* Γ*, written* $\langle\Gamma\rangle_f$*, is the set of all relations which can be freezingly implemented by* Γ*.*

Definition 8.8 (frozen basis) *Let* F *be a frozen partial co-clone. Any language* Γ *which generates* F *(*$\langle\Gamma\rangle_f = F$*) is called a* frozen basis *of* F*.*

Using standard arguments, it is relatively easy to show that frozen partial co-clones define closed classes for frozen implementations (the only nontrivial point is to show transitivity, that is, that frozen implementations compose together into implementations which are also frozen), and that they are ordered by inclusion in a lattice. We state these propositions without their (tedious but simple) proof.

Proposition 8.9 *Frozen co-clones define equivalence classes for frozen implementations, in the sense that if* Γ_1 *and* Γ_2 *define the same frozen co-clone (*$\langle\Gamma_1\rangle_f = \langle\Gamma_2\rangle_f$*), then* Γ_1 *freezingly implements all relations in* Γ_2*, and vice-versa.*

Proposition 8.10 *Frozen partial co-clones ordered by inclusion form a lattice with* $F_1 \sqcap F_2 = F_1 \cap F_2$ *defining its meet operation, and* $F_1 \sqcup F_2 = \langle F_1 \cup F_2\rangle_f$ *defining its join operation.*

It is easily seen that the lattice of frozen partial co-clones is refined by that of partial strong co-clones, but that it refines that induced by faithful implementations (which itself refines Post's lattice). For faithfulness, this is because in an implementation of $R(X)$ by a formula $\exists X'\varphi$, the assignments in $R(X)$ can be extended to models of φ only by assigning 0 (resp. 1) to each variable in X' which is frozen to 0 (resp. 1). Hence, frozen co-clones define a structure allowing for the study of problems modulo reductions which preserve the number of variables, while, we hope, being more manageable that the lattice of partial strong co-clones (this hope however is vain for some co-clones, see Proposition 8.17).

The following result, which follows directly from the construction, shows that frozen co-clones define a grain sufficiently small for our needs. In other words, that for classifying the exact complexity of problems such as satisfiability, inference by circumscription, abduction, etc., according to a restriction of the input formulas to a finite language, it is enough to determine the complexity for an arbitrary finite language in each frozen co-clone.

Proposition 8.11 *Let* Γ_1, Γ_2 *be two finite languages with* $\langle\Gamma_1\rangle_f = \langle\Gamma_2\rangle_f$*. Then there is a transformation between* Γ_1*-formulas and* Γ_2*-formulas which is faithful and preserves the number of variables (up to a constant term equal to the size of the domain* D*).*

8.4.3 Frozen Boolean Partial Polymorphisms

We now introduce briefly the notion of a frozen partial polymorphism, and exhibit a connection between frozen co-clones and these polymorphisms, in the same spirit as the connection between co-clones and clones (via regular polymorphims), or between partial strong co-clones and partial strong clones (via partial polymorphisms).

An important restriction in this section is that the results which we give are valid only for the Boolean domain; we discuss why at the end of the section.

We start with an important definition.

Definition 8.12 (freezable) *A value $v \in \{0,1\}$ is said to be* freezable *by a finite Boolean language Γ if there is a Γ-formula φ in which some variable x is frozen to v, or, equivalently, if the unary relation $\{v\}$ is in the (ordinary) co-clone $\langle\Gamma\rangle$ of Γ.*

We can now introduce the notion of a frozen partial polymorphism. Observe that, contrary to the case of usual and partial polymorphisms, there is no absolute notion of a frozen partial *function*, because the notion is dependent on the considered constraint language Γ, which defines the relevant (freezable) values.

Definition 8.13 (frozen) *Let Γ be a Boolean constraint language. A partial function f is said to be* frozen *(for Γ) if for all values $v \in \{0,1\}$ such that v is freezable in Γ, f is defined on the tuple of arguments $(v, v, \ldots, v)$ and satisfies $f(v, v, \ldots, v) = v$.*

Note that this is different from requiring f to be idempotent, since, e.g., $f(0, \ldots, 0)$ can be defined and be 1 if 0 is not frozen for Γ.

Definition 8.14 (frozen polymorphism) *A k-ary partial function f is said to be a* frozen polymorphism *of a finite Boolean language Γ if it is frozen for Γ and is a partial strong polymorphism of it, that is, for all relations $R \in \Gamma$ and any choice of k tuples $t_1, \ldots, t_k$ in R, either $f(t_1[i], \ldots, t_k[i])$ is undefined for at least one coordinate i, or the tuple $f(t_1, \ldots, t_k)$ (coordinate-wise application) is in R.*

The set of all frozen polymorphisms of Γ is written $fPol(\Gamma)$.

Example 8.15 *Let R be the ternary relation $\{001, 010, 111\}$ and let Γ be the singleton language $\{R\}$. Then 1 is freezable in Γ (by $R(x,x,x) \equiv x$), but 0 is not since Γ is 1-valid. Let f be the binary partial function undefined on the tuple $(0,0)$, and otherwise defined by $f(x_1, x_2) = x_1 \wedge x_2$. Note that f is frozen for Γ since it is defined on $(1,1)$, and $f(1,1)$ is 1. Then f is a frozen polymorphism of R. Indeed, it is undefined on the couple of tuples $(001, 010)$, and as for the other couples we have $f(001, 111) = f(111, 001) = 001 \in R$ and $f(010, 111) = f(111, 010) = 010 \in R$. The cases when a tuple is repeated and f is defined (e.g., $f(111, 111)$) also pass the test due to the idempotency of conjunction.*

On the other hand, if f was also defined on $(0,0)$ (and equal to 0 on it), then it would not be a polymorphism of R, as follows from $001 \wedge 010 = 000 \notin R$.

We can now give the correspondence between frozen co-clones and sets of frozen partial polymorphisms. The proof is technical but uses rather standard ideas (the reader is referred to the article).

Proposition 8.16 (Nordh and Zanuttini 2009, Theorem 12) *Let Γ_1, Γ_2 be two Boolean languages. Then $\langle\Gamma_1\rangle_f \subseteq \langle\Gamma_2\rangle_f$ is equivalent to $fPol(\Gamma_2) \subseteq fPol(\Gamma_1)$.*

We do not claim that there is a Galois correspondence, because we do not have a notion of closed class for frozen polymorphisms. This is because for a function, being "frozen" is relative to a particular constraint language. Using the usual notion of closure by superposition would by definition yield the partial co-clone of a set of polymorphims, which cannot be correct. Hence the notion of closure should certainly depend on the projections allowed, but we leave this as an open problem.

Open Question 6 *Define a notion of closure on partial functions, which takes into account the freezable values, and allows to define a Galois correspondence with frozen partial co-clones through the operator fPol.*

As an important consequence of Proposition 8.16 together with the definition of a freezable value, we have the following unfortunate result. Intuitively, it simply says that if no constant is freezable, then existential quantification can never be used in implementations, so that frozen implementations collapse with partial strong implementations.

Proposition 8.17 *If C is a (regular) Boolean co-clone containing neither of the unary relations $R_F = \{0\}$ and $R_T = \{1\}$, then the frozen partial co-clones F with $\langle F\rangle = C$ (that is, which are included in C but in none of its subco-clones) are exactly the partial strong co-clones P with $\langle P\rangle = C$.*

Hence, for co-clones such as IE, IV, IM, etc., there is no hope that analyzing problems modulo frozen implementations is easier than using partial strong implementations. However, we still hope that the structure of frozen co-clones is simpler for other co-clones and, as we will see in Section 8.6, this is indeed the case for at least some of the bottommost of them (in Post's lattice).

To conclude this section, recall that the main result given here (Proposition 8.16) is valid only for Boolean languages. Technically, this is because for the Boolean case, we use the fact that $v \in \{0,1\}$ is freezable by Γ if and only if the relation $\{v\}$ is in $\langle\Gamma\rangle_f$ (in addition to being in $\langle\Gamma\rangle$, as required by the definition). This fact is an essential step in our proofs.

Example 8.18 *Let $D = \{0,1,2\}$ and consider the relation $R = \{(0,1),(0,2)\}$. Then the constant value 0 cannot be freezingly implemented by R, though it can be implemented by it (via $x = 0 \equiv \exists x' R(x,x')$).*

We leave the possibility to lift our results to general, finite domains D as an open question. An idea (see Example 8.18) is to define a value $v \in D$ to be *antifreezable* if there is a formula in which some variable x is forced to be *different from* v, and allowing existential quantification over such variables in frozen implementations. Clearly, in the Boolean case both versions collapse, since v is antifreezable in this sense if and only if $1 - v$ is freezable, but this definition may open new perspectives in generalizing the notion of frozen polymorphisms to general domains.

Open Question 7 *Generalize the notion of a frozen implementation and/or of a frozen polymorphism to general finite domains D, in such a way that the exact complexity of problems remains preserved but Proposition 8.16 holds.*

8.5 Techniques for Determining Frozen Partial Co-Clones

In Section 8.6 we will give results about the structure of frozen partial co-clones in some parts of Post's lattice. Rather than giving the proofs in detail there, we describe here a few techniques which can be used for determining that a given set of relations Γ freezingly implements, or does not freezingly implement, a given relation R. Most results presented in Section 8.6 indeed rely on these techniques, but these may also be useful for determining the structure of further parts of Post's lattice.

Because they rely on the correspondence with partial polymorphisms, these techniques are directly available for Boolean languages, but not for languages on general finite domains (see Open Question 7).

We start with a technique directly derived from Proposition 8.16 (and illustrated in the proof of Proposition 8.28).

Corollary 8.19 (of Proposition 8.16) *Exhibiting a partial function f which is frozen in Γ and R, which is a (partial) polymorphism of Γ but which does not preserve R is enough for showing $R \notin \langle\Gamma\rangle_f$.*

Conversely, the second technique allows to show that a language Γ *does* implement a relation R. It uses a CNF representation of some Γ-formula φ.

For a CNF formula $\varphi = \bigwedge_{i \in I} c_i$ and a subset X_0 of its variables, we denote by φ_{-X_0} the CNF formula obtained from φ by removing every c_i containing only variables from X_0, and every literal over a variable in X_0 from every other clause c_i (we write $(c_i)_{-X_0}$ for the clause so obtained). Moreover, for an assignment μ to a set of variables X_1, we write $\varphi(\mu)$ for the formula $\bigwedge_{x \in X_1, \mu \models x} x \wedge \bigwedge_{x \in X_1, \mu \models \overline{x}} \overline{x}$.

Example 8.20 *Let $\varphi = (x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_3 \vee x_4)$ and let $X_1 = \{x_1, x_2\}$. Then $\varphi_{-X_1} = (x_3) \wedge (x_3 \vee x_4)$.*

Interestingly, the assignment which satisfies φ_{-X_0} in the following statement is precisely an *autark assignment* of φ, as used in the study of satisfiability problems (Monien and Speckenmeyer, 1985).

Proposition 8.21 *Let R be a relation and let $\varphi \equiv R(X)$ be a CNF formula. Let $X_0 \subseteq X$ be a set of variables, and let φ_{X_0} be the set of all clauses of φ that contain only variables of X_0. If φ_{-X_0} is satisfiable (resp. satisfied by the all-0 assignment $\mu_{\vec{0}}$, by the all-1 assignment $\mu_{\vec{1}}$), then the constraint language $\Gamma = \{R, R_T, R_F\}$ (resp. $\{R, R_F\}$, $\{R, R_T\}$) freezingly implements the relation R_{X_0} defined by $R_{X_0}(X_0) \equiv \varphi_{X_0}$.*

Proof Let μ be a model of φ_{-X_0}. It is easy to see that $\varphi \wedge \varphi(\mu) \equiv \varphi_{X_0} \wedge \varphi(\mu)$ holds, and by construction every variable in $X_1 = \mathit{Vars}(\varphi(\mu)) = X \setminus X_0$ is frozen in this formula. Now since $X_0 \cap X_1 = \emptyset$ holds by construction, we have:

$$\exists X_1(\varphi \wedge \varphi(\mu)) \equiv \exists X_1(\varphi_{X_0} \wedge \varphi(\mu)) \equiv \varphi_{X_0}$$

which concludes the proof. The restriction to the case where $\mu = \mu_{\vec{0}}$ or $\mu = \mu_{\vec{1}}$ is obvious. □

Example 8.22 (continued) *Let $\varphi = (x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_3 \vee x_4)$, and let $X_0 = \{x_1, x_2\}$. Then $\varphi_{-X_0} = (x_3) \wedge (x_3 \vee x_4)$ has $\mu = 10$ as one of its models. Thus $\exists x_3, x_4, \varphi \wedge T(x_3) \wedge F(x_4)$ is a frozen implementation of the relation R_{X_0} defined by $R_{X_0}(x_1, x_2) \equiv (x_1 \vee x_2)$ (i.e., $R_{X_0} = \{01, 10, 11\}$), as can be checked.*

8.6 The Frozen Structure of Some Co-Clones

We now investigate some parts of Post's lattice, and determine the corresponding structure of frozen co-clones. We only give two example proofs which illustrate the use of the techniques presented in Section 8.5, because the other proofs are rather technical and tedious. Most proofs also use the plain bases known for each co-clone (Creignou et al., 2008b).

Interestingly, it is known that there are 25 partial strong co-clones included in (or equal to) IM_2, and 33 in ID_1 (Haddad and Simons, 2003). Contrastingly, we show here that there are only 8 frozen partial co-clones in IM_2 and 6 in ID_1. This suggests that the lattice of frozen partial co-clones is indeed less complex than that of partial strong co-clones.

Nevertheless, as could be expected this lattice is more complex than that of ordinary co-clones. In particular, we show that the ordinary co-clone ID_2 splits into 13 frozen partial co-clones. Note however that, to the best of our knowledge, the structure of *partial strong* co-clones in ID_2 is not known. It is possibly significantly more complex than that of frozen co-clones, or possibly closely related to it, in which case our study may help in determining it.

8.6.1 Frozen Partial Co-Clones which Coincide with Co-Clones

We start our study by exhibiting quite a number of co-clones for which there is a single frozen partial co-clone, that is, co-clones C such that for all languages Γ with $\langle\Gamma\rangle = C$, $\langle\Gamma\rangle_f = C$ also holds. Equivalently, if $\langle\Gamma\rangle = C$, then any relation which can be implemented by Γ can also be *freezingly* implemented by Γ.

The co-clones which we could identify to have a single frozen partial co-clone are the ones depicted in grey on Figure 8.1.

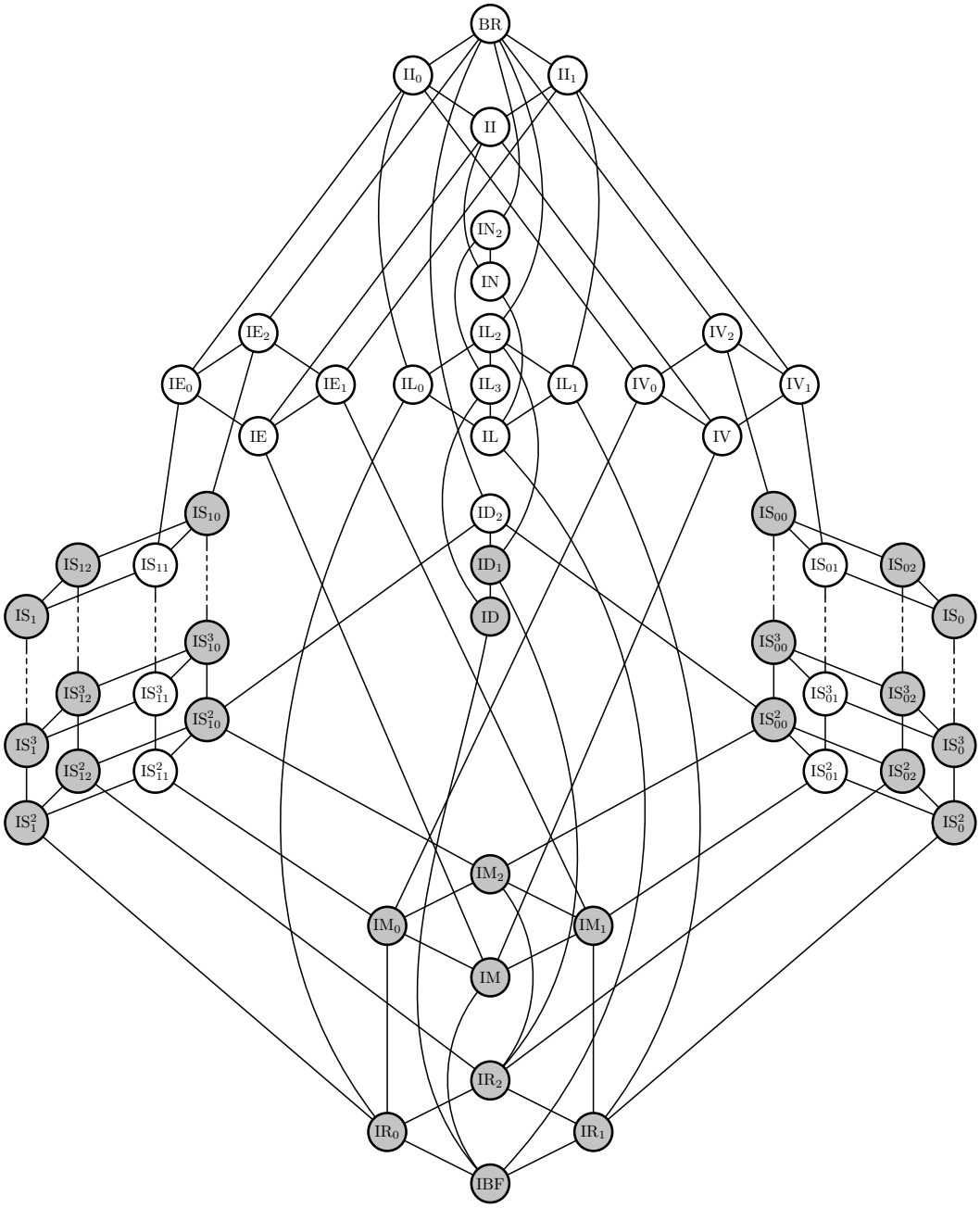


Figure 8.1: Boolean co-clones with single frozen partial co-clone.

Proposition 8.23 *For each co-clone C depicted in grey on Figure 8.1, any finite language Γ with $\langle\Gamma\rangle = C$ satisfies $\langle\Gamma\rangle_f = C$ as well. Equivalently, there is exactly one frozen co-clone F with $\langle F\rangle = C$, namely, $F = C$.*

We also conjecture that this list is complete, though we have not been able to prove this yet.

Open Question 8 *Determine whether for all co-clones C not depicted in grey on Figure 8.1, there are at least two different frozen partial co-clones F_1, F_2 with $\langle F_1\rangle = C$ and $\langle F_2\rangle = C$. We conjecture that the answer is affirmative.*

We now give the proof for $\mathrm{IS}_0^{\mathrm{n}}$, which illustrates the second technique in Section 8.5 and the use of weak bases.

Proposition 8.24 (monotone relations) *For all $n \in \mathbb{N}$, any finite language Γ with $\langle\Gamma\rangle = \mathrm{IS}_0^{\mathrm{n}}$ freezingly implements all relations in $\mathrm{IS}_0^{\mathrm{n}}$.*

Proof By results about plain bases (Creignou et al., 2008b) the relations R in $\mathrm{IS}_0^{\mathrm{n}}$ are exactly those for which $R(X)$ is equivalent to a conjunction φ of positive clauses each containing at most n literals. Hence, because $\langle\Gamma\rangle$ is $\mathrm{IS}_0^{\mathrm{n}}$, there must be at least one relation R in Γ which is equivalent to a conjunction of positive clauses among which at least one, say $(x_1 \vee \cdots \vee x_n)$, contains exactly n variables and is prime in φ.

Write X_0 for $\{x_1, \ldots, x_n\}$ and φ_{X_0} for the conjunction of all clauses of φ which contain only variables in X_0. Now consider the formula φ_{-X_0}. This formula is positive, thus it is satisfied by the all-1 assignment $\mu_{\overline{1}}$ to $X \setminus X_0$. By Proposition 8.21, it follows that $\{R, R_T\}$ freezingly implements φ_{X_0}. But since the clause $(x_1 \vee \cdots \vee x_n)$ is prime in φ, and φ contains only positive clauses, there can be no other clause in φ over X_0, that is, $\varphi_{X_0} = (x_1 \vee \cdots \vee x_n)$. Thus $\{R, R_T\}$ freezingly implements $(x_1 \vee \cdots \vee x_n)$. Finally, since the relation R_T is in $\mathrm{IS}_0^{\mathrm{n}}$ we have that Γ freezingly implements R_T (Nordh and Zanuttini, 2009, Proposition 8), and hence Γ freezingly implements $(x_1 \vee \cdots \vee x_n)$ (by transitivity of frozen implementations). Because identification of variables is allowed in Γ-formulas, and hence in frozen implementations, it follows that Γ freezingly implements every positive clause of length *at most* n, hence it freezingly implements every relation in $\mathrm{IS}_0^{\mathrm{n}}$. □

8.6.2 Frozen Structure of Bijunctive Languages

We now exhibit the structure of frozen co-clones in the co-clone ID_2 (containing exactly the bijunctive relations). Recall that a relation is *bijunctive* if it can be defined by a CNF formula with at most two literals per clause, or, equivalently, by a conjunction of binary constraints.

We start by introducing the basic relations which serve to define these frozen co-clones, then we define the frozen co-clones themselves.

Definition 8.25 (relations in ID_2) *We denote by $R_2^p, R_2^n, R_2^i, R_2^{\neq}, R_3^p, R_3^n, R_4^p$ the 7 relations defined as follows:*[1]

$$\begin{aligned}
R_2^p(x_1, x_2) &\equiv (x_1 \vee x_2)\\
R_2^n(x_1, x_2) &\equiv (\overline{x}_1 \vee \overline{x}_2)\\
R_2^i(x_1, x_2) &\equiv (\overline{x}_1 \vee x_2)\\
R_2^{\neq}(x_1, x_2) &\equiv (x_1 \neq x_2)\\
R_3^p(x_1, x_2, x_3) &\equiv (x_1 \vee x_2) \wedge (x_1 \neq x_3)\\
R_3^n(x_1, x_2, x_3) &\equiv (\overline{x}_1 \vee \overline{x}_2) \wedge (x_1 \neq x_3)\\
R_4^p(x_1, x_2, x_3, x_4) &\equiv (x_1 \vee x_2) \wedge (x_1 \neq x_3) \wedge (x_2 \neq x_4)
\end{aligned}$$

[1] The mnemonics are: 2 stands for "binary", "p" for "positive", "n" for "negative", and "i" for "implicative".

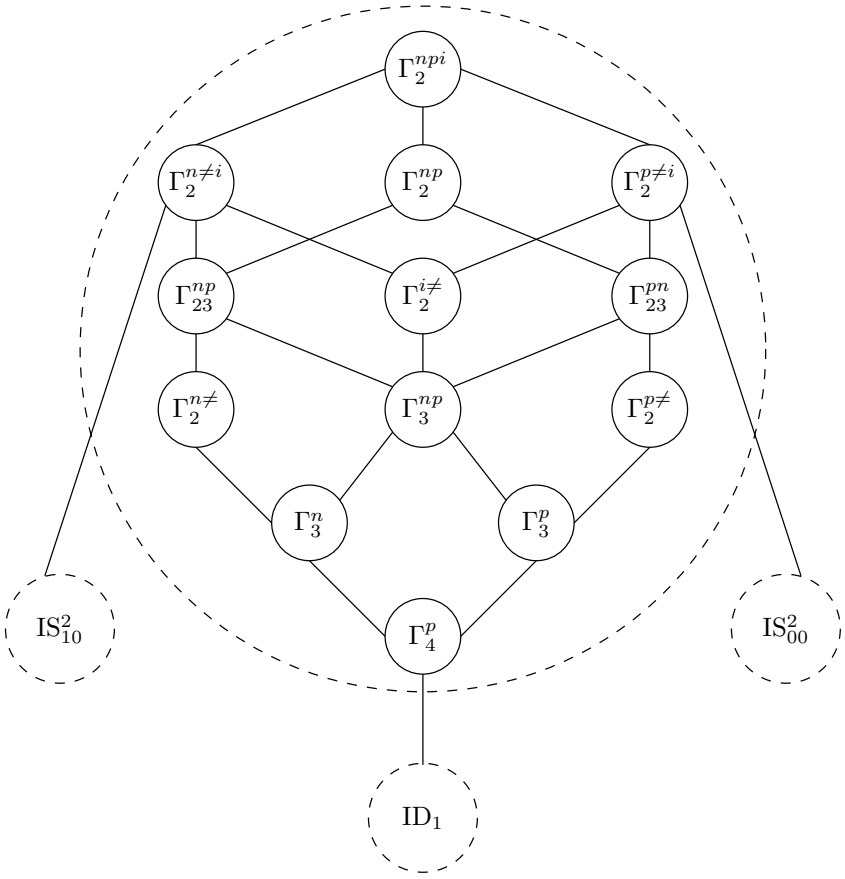


Figure 8.2: Frozen partial co-clones in ID_2.

Definition 8.26 (frozen partial co-clones in ID_2) *We denote by Γ_4^p, Γ_3^p, Γ_3^n, Γ_3^{np}, $\Gamma_2^{n\neq}$, $\Gamma_2^{p\neq}$, $\Gamma_2^{i\neq}$, Γ_2^{np}, Γ_{23}^{np}, Γ_{23}^{pn}, $\Gamma_2^{n\neq i}$, $\Gamma_2^{p\neq i}$, and Γ_2^{npi} the 13 frozen partial co-clones defined as follows:*[2]

- $\Gamma_4^p = \langle R_4^p \rangle_f$,
- $\Gamma_3^p = \langle R_3^p \rangle_f$, $\quad \Gamma_3^n = \langle R_3^n \rangle_f$, $\quad \Gamma_3^{np} = \langle R_3^n, R_3^p \rangle_f$,
- $\Gamma_2^{n\neq} = \langle R_2^n, R_2^{\neq} \rangle_f$, $\quad \Gamma_2^{p\neq} = \langle R_2^p, R_2^{\neq} \rangle_f$, $\quad \Gamma_2^{i\neq} = \langle R_2^i, R_2^{\neq} \rangle_f$, $\quad \Gamma_2^{np} = \langle R_2^n, R_2^p \rangle_f$,
- $\Gamma_{23}^{np} = \langle R_2^n, R_3^p \rangle_f$, $\quad \Gamma_{23}^{pn} = \langle R_2^p, R_3^n \rangle_f$,
- $\Gamma_2^{n\neq i} = \langle R_2^n, R_2^{\neq}, R_2^i \rangle_f$, $\quad \Gamma_2^{p\neq i} = \langle R_2^p, R_2^{\neq}, R_2^i \rangle_f$,
- $\Gamma_2^{npi} = \langle R_2^n, R_2^p, R_2^i \rangle_f$.

The inclusion structure among these frozen partial co-clones and the direct subco-clones of ID_2 is given on Figure 8.2.

Proposition 8.27 *Figure 8.2 gives the exact structure of the frozen partial co-clones F satisfying $\langle F \rangle = \mathrm{ID}_2$.*

[2] The mnemonics are: subscripts represent the arities of the relations in the frozen basis, and superscripts represent the nature of these relations, in the same order ("p" stands for "positive", etc.).

The proof of Proposition 8.27 involves showing that all the given inclusion relations hold. Showing $F_1 \subseteq F_2$ is done by exhibiting a frozen implementation of each relation in the frozen basis of F_1 by the relations in the frozen basis of F_2.

It also involves showing that the frozen partial co-clones are the only existing ones. The proof goes by showing that any language Γ which is in some frozen coclone F but in none of its direct children freezingly implements all the relations in the basis of F. For this we use techniques similar to the ones illustrated for Proposition 8.24.

Finally, the proof is complete if we show that all frozen co-clones are indeed distinct. For this we use the first technique presented in Section 8.5, namely, for showing $F_1 \neq F_2$ we exhibit a frozen polymorphism of one which is not a polymorphism of the other. We illustrate this point by the proof for Γ_4^p, Γ_3^p.

Proposition 8.28 *The frozen partial co-clones Γ_4^p and Γ_3^p are distinct.*

Proof Let f be the partial ternary function which is undefined when its tuple of arguments is $(0,1,0)$ or $(0,0,1)$, and otherwise is defined by $f(0,0,0) = 0, f(1,1,1) = 1$, $f(1,0,0) = 1$, and $f(0,1,1) = f(1,0,1) = f(1,1,0) = 0$ (f returns its minoritary argument). Observe that whenever an argument appears twice, f returns the third one. Hence when trying to show that f preserves some relation, this is clear for any choice of tuples where one is repeated.

Observe that the relation R_4^p in the basis of Γ_4^p is exactly the relation $\{0110, 1001, 1100\}$. For any choice of three different tuples in R_4^p (that is, all three in some order), it can be seen that f is undefined on at least one of the two last components (e.g., $f(0110[4], 1001[4], 1100[4]) = f(0,1,0)$ is undefined). Hence f preserves R_4^p, and it follows from Proposition 8.16 that f preserves all relations in Γ_4^p.

On the other hand, observe that the relation R_3^p in the basis of Γ_3^p is $\{011, 100, 110\}$. We have $f(011, 100, 110) = (f(0,1,1), f(1,0,1), f(1,0,0)) = (0,0,1) \notin R_3^p$, hence f does not preserve R_3^p.

Since f is frozen on both values $(0,1)$ which are freezable in ID_2, by Corollary 8.19 we get $R_3^p \not\subseteq \Gamma_4^p$, as desired. □

Interestingly, observe that the least frozen partial co-clone in some co-clone C (in our case, Γ_4^p) does not include in general the direct subco-clones of C (in our case, IS_{10}^2 or IS_{00}^2). It would be interesting then to have the inclusion relationship between *all* frozen co-clones on Figure 8.2 and *all* subco-clones of ID_2 (which are exactly its *frozen* subco-clones by the results presented on Figure 8.1, except maybe for IS_{11}^2 and IS_{01}^2).

Open Question 9 *For each frozen co-clone F on Figure 8.2, determine the maximal co-clones which are included in F.*

8.7 Perspective: Generalizing Techniques for Using Frozen Implementations

Despite our effort in identifying the frozen structure of ID_2, we think it is vain, if interesting at all, to try to determine the frozen structure of all Post's lattice. The structure exhibited for ID_2 should rather be viewed as a proof of concept.

If one is concerned with classifying problems $\textsc{problem}(\Gamma)$ parameterized by a finite language Γ according to their exact complexity, then for sure a complete description of the lattice of frozen partial co-clones would be of great help. Nevertheless, exact complexity for problems over Boolean formulas is essentially concerned with exponential time complexities: apart from some degenerate cases, solving a problem requires reading its input, which requires time proportional to the length of the instance (formula). For polynomial time complexities, this will dominate the number of variables, because in general there is no polynomial bound on the length of formulas as a function of their number of variables. Hence, for such problems frozen implementations will not help more than ordinary co-clones, since they give no better guarantees on the increase in formula size through the reductions.

On the other hand, classifying the exact complexity of some problem (in the polynomial hierarchy) means obtaining results such as "the problem is solvable in time $O(f(n))$ for such and such language Γ, and in all other cases it has complexity at least $O(g(n))$ for some functions f, g with $f(n) \in o(g(n))$". But if g is exponential, obtaining such results would be a proof of P $\neq$ NP (or a separation between P and some level of the polynomial hierarchy).

Where frozen co-clones may be much more useful is for classifying the complexity of some problems whose complexity is not preserved by regular implementations. As already evoked, inference by circumscription is such a problem. Solution counting is also such a problem, though faithful implementations are enough. For such problems, frozen co-clones can be helpful even if we are only concerned with classifying the problem in the different levels of the polynomial hierarchy, whereas regular co-clones do not help.

Again, a complete picture of all frozen co-clones would help for such tasks. However, general-purpose techniques may be enough: usually a complexity classification modulo polynomial-time reducibility spans few complexity classes, and typically one has an idea of where the hard and easy problems are. It is enough then for the purpose of classification to be able to show that some languages Γ_1 and Γ_2 for which the problem is known to be, say, in P and NP-complete, respectively, are such that $\langle\Gamma_1\rangle_f$ is a direct child of $\langle\Gamma_2\rangle_f$. This is exactly, in fact, the technique used by Schaefer (with regular implementations) for its seminal result on the complexity of SAT, and by Creignou *et al.* in their monograph on complexity classifications (Creignou et al., 2001), with (among others) regular and faithful implementations.

Hence what we see as an interesting perspective of this work is to generalize the techniques presented in Section 8.5. An interesting case study would be to revisit the known complexity classification for circumscription (Durand et al., 2009) along the lines above, by justifying that the classification is complete directly by frozen implementations.

8.8 Perspective: Insights into the Exact Complexity of Satisfiability

To moderate the claims in Section 8.7, an interesting perspective of this work is in complexity theory, precisely in the exact complexity of SAT.

From results by Schnoor and Schnoor (2008), we know the two candidate "easiest" NP-complete SAT(Γ) problems, namely, the cases when Γ is the smallest partial strong co-clone satisfying $\langle\Gamma\rangle =$ BR or $\langle\Gamma\rangle =$ IN. Indeed, by definition of partial strong co-clones, SAT(Γ_1) is at least as hard as SAT(Γ_2) when $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle =$ BR and Γ_2 is the smallest such language, and similarly for IN. On the other hand, by Schaefer's result any language Γ with $\langle\Gamma\rangle \neq$ BR, IN has a polynomial SAT(Γ).

Hence we have two candidate "easiest" NP-complete SAT problems. From the viewpoint of complexity theory it would be interesting to know which one is easier, if any. It would also be interesting to determine the frozen co-clones generated by these partial co-clones since, by definition, SAT has the same exact complexity for all the so encompassed languages (we assume here that the complexity for SAT(Γ) is expressed as a function of the number of variables only). For this, frozen implementations could of course be used, and showing which are the easiest among different problems (instead of determining their complexity) avoids the need to prove P $\neq$ NP. Indeed, results such as "SAT(Γ_1) is (strictly) easier than SAT(Γ_2)" can be obtained using, for instance, reductions from SAT(Γ_2) to SAT(Γ_1) which divide the number of variables by 2.

It would also be very interesting to make a parallel between this study and the known "easiest" NP-complete problem from the viewpoint of descriptive complexity (Barbanchon and Grandjean, 2004).

Chapter 9

Concept Learning in Post's Lattice

Contents

We finish this document with a prospective discussion about issues in concept learning. We propose two main directions of research, both related to Post's lattice. The first direction concerns a "learnability map" for the logical languages in Post's lattice, and more general questions about transferring learnability results between languages. The second direction is the study of learning algorithms able to automatically change their bias.

The first direction benefits from many useful discussions with Frédéric Koriche. The preliminaries relevant to this chapter can be found in Sections 2.2 (Page 9) and 3.2 (Page 25).

9.1 Biases in Concept Learning

Recall that from a high-level point of view, Boolean concept learning is the problem of computing (an approximation of) a hidden Boolean concept $c^* \subseteq 2^X$, given positive and negative examples of this concept (assignments $\mu \in c^*$ and $\mu \notin c^*$, respectively) received passively, and possibly given access to oracles which enable the learner to ask questions of its choice about c^*. Note that there are still more general settings, the first of which is the setting of *agnostic learning*, in which it is not even assumed that there is a hidden concept at all (Kearns et al., 1994). We will however not consider such models here.

Of the greatest importance in concept learning is the notion of a *bias*. In a learning process, a bias is typically defined to be "any basis for choosing one generalization over another, other than strict consistency with the observed training instances" (Mitchell, 1980). In our setting of Boolean concept learning, we will focus on biases as defined by a language for *hypotheses*. For simplicity, we will not make the difference here between a hypothesis (that is, a subset of 2^X) and its representation in a given language (that is, a formula), but keep in mind the following:

- the class of all subsets of 2^X which are allowed as hypotheses defines to what extent the learner can *generalize* from examples; for instance, if any subset of 2^X is allowed as a hypothesis, then given sets of positive and negative examples, any concept consistent with them can

be a generalization; contrastingly, if only *monotone* (increasing) subsets of 2^X are valid hypotheses, then given, say, the assignment 0011 as a positive example, any generalization allowed must also classify 1011, 0111, and 1111 as positive;

- a class of *representations* for hypotheses defines a class of subsets of 2^X (those which are the set of models of at least one allowed representation), and hence to what extent the learner can generalize from examples, but it also imposes *computational requirements*; in particular, we typically measure sample and computational complexity as a function of the size of the smallest representation of the target concept (or a best approximation of it) in the class of representations chosen.

Definition 9.1 (bias) *We define a* learning bias *to be any class of Boolean formulas. Given a Boolean concept learning problem* LEARNING *and a bias B, an algorithm for* LEARNING *is said to* use hypotheses from B *if it returns a formula $\hat{c} \in B$, and, in case equivalence queries are allowed, if it calls $EQ(\hat{c})$ only with hypotheses $\hat{c} \in B$.*

Example 9.2 *In the setting of exact identification with queries, a learner using the class of all Horn CNF formulas as its bias can ask equivalence queries $EQ(\hat{c})$ only with hypotheses $\hat{c}$ which are Horn CNF formulas, and in the end it must output a Horn CNF formula $\hat{c}$ which is logically equivalent to the target concept c^*.*

In the online or KWIK models (Littlestone, 1988; Li et al., 2011), we also intend a bias B to force the learner to predict any example μ using a hypothesis $\hat{c} \in B$, that is, to predict the example to be positive (resp. negative) if $\mu \models \hat{c}$ holds (resp. $\mu \not\models \hat{c}$ holds). We will however not discuss these models in details in this chapter.

Given the observations above, the choice of a bias for learning some hidden concept is essential to the efficiency and success of the learning process. As it turns out, whatever the precise setting, a partial order can be defined on biases, from which we will be able to (partly) compare them.

Definition 9.3 (strength) *A learning bias B_1 is said to be* stronger *than a bias B_2 if any concept $c^* \subseteq 2^X$ which can be represented by a formula $\varphi_1 \in B_1$ (i.e., $c^* = M(\varphi_1)$) can also be represented by a formula $\varphi_2 \in B_2$, and B_2 can represent at least one more concept than B_1. We also say that B_2 is* weaker *than B_1.*

Example 9.4 *The bias consisting of all Horn CNF formulas is weaker than the bias consisting of all monotone CNF formulas. This is because monotone CNF formulas (no negative literals) are special cases of Horn CNF formulas (at most one negative literal per clause).*

Finally, it is essential for this chapter to make clear the distinction between the *learning bias* and the assumptions about the *target concept*. Typically, a learning problem LEARNING is parameterized by a class of target concepts, written $\mathcal{C}$. Then the problem LEARNING($\mathcal{C}$) is the problem of computing a concept $\hat{c}$ equivalent, or "close to", a hidden target concept c^*, *given that c^* is in the class $\mathcal{C}$*. This is to be distinguished from the bias for hypotheses. Also observe that the representation of the target concept is irrelevant to the learning problem (only its set of models, i.e., the target subset of 2^X, is relevant).

Definition 9.5 (learning a class by another) *Let $\mathcal{C}$ be a class of subsets of 2^X, and let B be a learning bias. A* learning problem for $\mathcal{C}$ by B *is a Boolean concept learning problem in which the target concept c^* is in $\mathcal{C}$ (and the learner knows this), and the learner can only use hypotheses from B. If the set of concepts $M(\varphi)$ representable by formulas $\varphi \in B$ is exactly $\mathcal{C}$, the learning problem is said to be* proper.

In general, for learning a concept class $\mathcal{C}$ one uses a bias B which can represent *at least* the concepts in $\mathcal{C}$, but this is not a definitive requirement (agnostic learning — Kearns et al. 1994 — is an extreme counterexample of this assumption). In fact, when the bias cannot represent all concepts in $\mathcal{C}$, learning runs the risk of collapsing, even in the absence of noise in the examples.

Definition 9.6 (collapse) *Let c^* be a target concept, let E^+, E^- be a set of positive (resp. negative) examples of c^*, and let B be a bias. Learning with hypotheses from B is said to* collapse *on E^+, E^- if there is no hypothesis $\hat{c} \in B$ such that all positive examples $e \in E^+$ satisfy $\hat{c}$ and all negative examples $e \in E^-$ falsify $\hat{c}$.*

Example 9.7 *Let the target concept c^* be $\{01, 10\}$ (equivalent to $x_1 \neq x_2$). Then learning with monotone hypotheses collapses on $E^+ = \{01, 10\}, E^- = \{11\}$, since by definition any monotone formula satisfied by 01 or 10 must be satisfied by 11 as well.*

9.2 Learning Problems in Post's Lattice

Recall that Post's lattice organizes constraint languages in classes (co-clones) which are closed under implementations. From such classes one can naturally define an equivalence relation between languages, namely, languages are equivalent if they implement each other. Intuitively, two equivalent languages Γ_1, Γ_2 are such that formulas of the form $\exists X \varphi_1$, with φ_1 a conjunction of constraints over Γ_1, are able to represent exactly the same Boolean concepts as formulas $\exists X \varphi_2$ with φ_2 over Γ_2.

In our opinion, Post's lattice offers a framework of choice for investigating Boolean concept learning problems, for at least three reasons:

- any Boolean concept c^* can be represented by a C-formula (that is, a conjunction of constraints with relations in $C \cup \{R_=\}$) for some co-clone C,
- co-clones define natural biases for learning algorithms, and the structure of these biases (i.e., their relationships through strength) is completely known by Post's lattice,
- the collection of biases induced by Post's lattice offers a rich choice for a learning problem, especially when the concept to be learnt is intended to be used in some computational problem whose complexity is classified in Post's lattice.

To make this last point clear, let as assume that an agent attempts to learn a target concept c^*, and that its guess $\hat{c}$ about c^* will serve as the input to, say, abduction problems. For instance, it may want to learn the description of child diseases by experience, and further use the acquired knowledge for (abductive) medical diagnosis. Then the learner can search the results about the complexity of the problem ABDUCTION(Γ) (Nordh and Zanuttini, 2008) for a language Γ which is as expressive as possible while allowing for efficient abduction, and then decide to learn c^* using hypotheses which are Γ-formulas.

We now define precisely what we intend by learning in Post's lattice. The forthcoming definitions make a crucial distinction between two kinds of hypotheses for some co-clone C. Recall that a Γ-*formula* is a conjunction of constraints all having their relation in $\Gamma \cup \{R_=\}$. Contrastingly, we call *existential* Γ-*formula* any formula of the form $\exists X \varphi$, where φ is a Γ-formula.

Example 9.8 *Let Γ_3 be the set of all ternary relations R such that $R(X)$ is equivalent to a clause of length 3 (there are essentially 4 such relations, depending on the number of negative literals in the clause). Then Γ_3-formulas can be seen as CNF formulas with at most 3 literals per clause. Observe that not all concepts $c \subseteq 2^X$ can be represented by a Γ_3-formula, e.g., the concept $M(x_1 \vee x_2 \vee x_3 \vee x_4)$ cannot. Contrastingly, it is well-known that any concept $c \subseteq 2^X$ can be represented by a formula of the form $\exists X' \varphi$ where φ is (essentially) a CNF with at most 3 literals per clause, that is, by an* existential Γ_3-*formula.*

We give the following definitions in a generic setting. To make things concrete, example learning problems are PAC-learning attribute-efficiently with membership queries, or exact identification with equivalence queries only. Similarly, we define learning problems with respect to arbitrary constraint languages, but we will focus on those which are co-clones (i.e., which are closed under implementation).

Definition 9.9 (learning problems for languages) *Let* LEARNING *be a learning problem, that is, a protocol of interaction for the learner with the environment together with success and performance criteria, and let* Γ_t, Γ_h *be Boolean constraint languages. The* learning problem for Γ_t by Γ_h*, written* LEARNING$(\Gamma_t|\Gamma_h)$*, is the learning problem* LEARNING *with target concepts* c^* *restricted to be* Γ_t*-formulas, and the learner forced to use only hypotheses* $\hat{c}$ *which are* Γ_h*-formulas.*

Definition 9.10 (existential learning problems) *In definition 9.9 we may write* $\exists\Gamma_t$ *(resp.* $\exists\Gamma_h$*) instead of* Γ_t *(resp.* Γ_h*), in which case target concepts (resp. hypotheses) are only restricted to be* existential Γ_t*-formulas (resp. existential* Γ_h*-formulas).*

When relevant, and in particular when Γ_h is infinite, we will make precise what representations we assume for (existential) Γ_h-formulas used as hypotheses (cf. the discussion in Section 2.2.1).

Example 9.11 *Recall that* IE_2 *is the language consisting of all Horn relations, and that* IS_0 *is the language consisting of all monotone relations (both are in fact co-clones). Then the problem* LEARNING$(\mathrm{IE}_2|\exists\mathrm{IS}_0)$ *is the problem of learning a target concept* c^* *known to be the set of models of some Horn CNF formula using hypotheses of the form* $\exists X\varphi$*, where* φ *is a conjunction of monotone constraints.*

As Example 9.8 demonstrates, the learning problems for $\exists\Gamma_t$ and/or $\exists\Gamma_h$ are in general not the same as for Γ_t and/or Γ_h. However, when the target languages Γ_t are supposed to be co-clones, by definition they boil down to the same problem.

Proposition 9.12 *Let* C_t *be a co-clone, let* Γ_h *be a constraint language, and let* LEARNING *be a learning problem. Then* LEARNING$(C_t|\Gamma_h)$ *is the same problem as* LEARNING$(\exists C_t|\Gamma_h)$*, and similarly,* LEARNING$(C_t|\exists\Gamma_h)$ *is the same problem as* LEARNING$(\exists C_t|\exists\Gamma_h)$*.*

Proof Obvious since target concepts are considered only as sets of models, that is, relations, and per definition of a co-clone as a set closed under implementation, the relations in $\exists C_t$ and in C_t are exactly the same. □

As concerns languages for hypotheses, the same can be said about the *sample complexity* of the problems, but not necessarily about their time complexity.

Proposition 9.13 *Let* C_h *be a co-clone, let* Γ_t *be a constraint language, and let* LEARNING *be a learning problem. Then the learning problems* LEARNING$(\Gamma_t|C_h)$ *and* LEARNING$(\Gamma_t|\exists C_h)$ *have the same sample complexity, and similarly,* LEARNING$(\exists\Gamma_t|C_h)$ *and* LEARNING$(\exists\Gamma_t|\exists C_h)$ *have the same sample complexity.*

Proof Clearly, an algorithm using hypotheses over C_h uses in particular hypotheses over $\exists C_h$, hence the sample complexity of LEARNING$(\Gamma_t|C_h)$ is at least as high as that of LEARNING$(\Gamma_t|\exists C_h)$. Conversely, if a learner uses a hypothesis of the form $\hat{c} = \exists X\varphi$ over $\exists C_h$ at some point, then by definition of a co-clone there is a logically equivalent hypothesis of the form $\hat{c}' = \varphi'$ for some C-formula φ' (this is at least true with $\varphi' = R_{\hat{c}}(\mathit{Vars}(\hat{c}))$ where $R_{\hat{c}}$ is $M(\hat{c})$). Hence if computation time is not a concern, the learner can present a C-formula instead of an existential C-formula as its hypothesis (or final result). □

9.3 Perspective: Learnability Map in Post's Lattice

We now present our first research project about learning in Post's lattice, namely, the project of determining a *learnability map* of co-clones. By a learnability map, we mean a systematic set of (un)learnability results and learning algorithms, for all co-clones C_t, C_h as languages for target concepts and hypotheses, and all standard computational learning problems. We borrow the term from the *knowledge compilation map* by Darwiche and Marquis (2002).

The motivation for establishing such a map is twofold. First, given a precise application with specific algorithmic tasks, efficiency requirements, and means of interaction with the environment (including humans), such a map can be used by designers to choose a precise knowledge representation language for their agents, in conjunctions with maps for other algorithmic tasks. The second motivation is the design of agents which are able to decide themselves on a bias for concept learning, depending on the specific task at hand, the means of interaction available, and the efficiency requirements (cf. the example of learning about child diseases in Section 9.2).

This project also calls for general reducibility results among computational learning problems (parameterized by the same co-clones) and among co-clones (for the same learning problem). Such results typically make it possible to determine large parts of the map from only a few results specific to some problems and co-clones (just as, e.g., the complexity of $\text{SAT}(\Gamma)$ can be classified from only 7 specific results, cf. Section 2.2.2). We now give a flavour of the type of results which can be (easily) obtained in this direction.

The first result is straightforward.

Proposition 9.14 *If C_1, C_2 are two co-clones with $C_2 \subseteq C_1$, then learnability of C_1 by any language Γ_h (resp. $\exists\Gamma_h$) entails learnability of C_2 by Γ_h (resp. $\exists\Gamma_h$) with the same protocol.*

Proof This follows directly from the fact that the problem for C_2 is the same as for C_1 but with more assumptions about the target concept. □

It is tempting to believe that the "dual" result holds, i.e., that for $C_2 \subseteq C_1$, learnability of Γ_t by C_2 entails learnability of Γ_t by C_1. However, this is not true for all protocols. For instance, PAC-learnability of CNF formulas (by CNF formulas) with membership queries is a long-standing open problem (Alekhnovich et al. (2008) give recent results on this point). Contrastingly, PAC-learning CNF formulas by the stronger bias of bijunctive CNF formulas, that is, learning the bijunctive CNF formula closest to the target concept, can be done using a simple version space approach, without even the need for membership queries (Valiant, 1984).

The next results build on plain bases of co-clones, and draw an important parallel between learnability problems for co-clones, which benefit from a strong structure, and the corresponding problems for classes of CNF formulas, which are more natural in applications. Moreover, most (un)learnability results about specific classes are given for classes of CNF formulas in the literature, hence this correspondence makes most of these reusable in our context.

Recall that a *plain basis* for a co-clone C is a set of relations pB such that all relations $R \in C$ can be defined by $R(X) \equiv \varphi$ for some pB-formula φ (without using existential quantification). We gave a plain basis for each co-clone, and most of these turn out to be sets of (relations defined by) clauses (Creignou et al., 2008b).

Proposition 9.15 *Let C be a co-clone, and let pB be a plain basis for C. Then learnability of C by any language Γ_h (resp. $\exists\Gamma_h$) is equivalent to learnability of pB by Γ_h (resp. $\exists\Gamma_h$) with the same protocol.*

Proof By definition of a plain basis, since target concepts are considered only up to logical equivalence. □

We now give some (partial) results about the corresponding transformations for *biases*. Note that these results consider biases of the form C (not $\exists C$).

Proposition 9.16 *Let C be a co-clone, and let pB be a plain basis for C. Then if some language Γ_t (resp. $\exists\Gamma_t$) is learnable by C with relations represented in extension in hypotheses, it is also learnable by pB-formulas, with the same protocol.*

Proof From results about plain bases (Creignou et al., 2008b) and about computing prime CNF formulas of relations given in extension (Zanuttini and Hébrard, 2002), it follows that whenever the learner uses a C-formula as a hypothesis, it can compute efficiently an equivalent pB-formula and hence, use it as its hypothesis. □

Proposition 9.17 *Let C be a co-clone, and let pB be a plain basis for C. Then if some language Γ_t (resp. $\exists\Gamma_t$) is learnable by pB-formulas, it is also learnable by C with the same protocol.*

Proof Obvious since pB-formulas and C-formulas define exactly the same concepts, and pB-formulas form a particular representation for C-formulas. □

Given these results and similar ones, which can be easily obtained, or found in the literature, it seems reasonable to hope that a large part of the learnability map could be obtained rather easily, apart from some long-standing open problems, like PAC-learnability of CNF formulas with membership queries. We will not go into the details of the extremely rich, 30 years old literature on these topics, but we simply point now at a few important studies directly related to this project.

Valiant (1984) gives a version space approach to learning CNF formulas with a bounded number of literals per clause, which is easily seen to work for any *finite* language and hence, in our case, for any co-clone with a finite plain basis. Natarajan (1991) gives a number of results about PAC-learning (without queries) of Boolean concept classes. His main focus is on classes which are closed under conjunction (intersection of concepts), which is the case of co-clones, and hence his results are particularly relevant to our learnability map. Bshouty (1995) and, based on his work, Khardon and Roth (1996) develop general results on the *characteristic models* of formulas (with respect to a class of formulas), which in case the class is a co-clone C, are essentially the models of the formula from which the others can be deduced by closure under the polymorphisms of C. Bshouty derives learning algorithms from this study (in the setting of exact identification with queries).

More directly about co-clones, Kavvadias and Sideri (1998) study the *inverse satisfiability problem* (for a language Γ), which asks whether a given relation can be represented by a Γ-formula, a problem related to *consistency learning* (Natarajan, 1991). Finally, we end this brief tour of the most relevant references by the work which is certainly the closest to ours, namely the work by Dalmau (1999) about the learnability of quantified Boolean formulas. Dalmau gives a complete classification of this problem (for all co-clones) for some range of learning models, which determines quite a large part of our learnability map. Nevertheless, there are still many open questions. In particular, Dalmau considers *improper* learning only, and he does not consider attribute-efficiency. Also wide open are most problems in the recently proposed KWIK model (Li et al., 2011).

9.4 Perspective: Incremental Change of Logical Bias

We now propose a second research project related to learning in Post's lattice. In a few words, we propose to study the problem of automatically *changing bias* when required by the examples received for some target concept c^*.

9.4.1 Why Change Bias?

In this presentation, we will focus on learning problems without noise, in which the learner is required to output a concept which is consistent with all the (positive and negative) examples received. Precisely, at any point during learning, if E^+ denotes the set of all positive examples received and E^- that of all negative examples received, the learner is required to output a hypothesis $\hat{c}$ such that for all examples $e \in E^+$, $e \models \hat{c}$ holds, and for all $e \in E^-$, $e \not\models \hat{c}$ holds. If the learning problem is one of *prediction*, as in the online and KWIK models, the learner is required to predict the examples received with such a hypothesis.

When examples are assumed to be noise-free, that is, correctly labelled as positive or negative according to the target concept, consistency of hypotheses is a natural requirement. In particular, it is a necessary requirement if we forbid the learner to make a mistake on some example for which it has already been told the "correct answer".

In this setting, the following can be observed for two biases B_1, B_2 such that B_1 is stronger than B_2 (these observations are valid in general, not specifically when biases are co-clones):

- if there is a correct hypothesis in B_1 (and hence in B_2), then it can be learnt from less examples using B_1 than B_2; to see this, consider the extreme case where B_1 contains only two hypotheses, say one logically equivalent to 1 (tautological) and one logically equivalent to 0 (unsatisfiable), B_2 is able to represent any concept, and the target concept c^* is logically equivalent to 1; then on receiving the first positive example, a learner using B_1 already decides (correctly) that the target concept is the tautological one, while a learner using B_2 still has 2^{n-1} consistent hypotheses[1];
- for the same reason, learning with B_1 induces more generalization from the examples than learning with B_2; again, as an extreme case, when B_2 can represent any concept, then there is *a priori* no reason not to learn the positive examples by heart (not doing so means using some other bias, like, for instance, a preference on hypotheses with short descriptions);
- if there are hypotheses consistent with the examples both in B_1 and in $B_2 \setminus B_1$, then reasoning with a hypothesis in B_1 will be at least as easy as with an arbitrary hypothesis in B_2; still considering our extreme case, if no negative example was received and some positive examples were, it is most often trivial to perform logical inferences from the (correct) tautological hypothesis in B_1, while it is not from an arbitrary hypothesis (say, an arbitrary CNF) in $B_2 \setminus B_1$ consistent with the positive examples.

In the absence of other reasons for choosing a bias (like domain-specific knowledge, for instance), these observations suggest to always choose the strongest possible bias. This can also be seen as an instanciation of Occam's razor, in favour of the simplest (strongest) biases, in addition to the more common idea of favouring the simplest *hypotheses* for a fixed bias (which is provably a good choice, in some formal sense — Natarajan 1991).

On the other hand, it is clear that choosing too strong a bias makes the learner run the risk of collapsing, that is, having no consistent hypothesis at all. Considering again our extreme example, with the bias B_1 containing only tautologies and unsatisfiable formulas, learning collapses as soon as one positive and one negative examples have been received. Hence all these observations call for some way of handling the tradeoff between simplicity and expressivity of the bias.

9.4.2 Changing Bias in Post's Lattice

From now on we assume that the learner uses only biases which are co-clones. As we already evoked, this is not as restrictive as it seems, because any concept can be represented as a C-formula for at least one co-clone C (may it be BR, the co-clone containing all relations), and because co-clones offer a rich choice of biases, as witnessed in particular by the complexity classes spanned by problems parameterized by co-clones. For instance, abduction problems are Σ_2^{P}-complete, NP-complete, coNP-complete, or in P depending in particular on the co-clone used for expressing knowledge bases (Nordh and Zanuttini, 2008).

What we intend by "changing bias in Post's lattice" is best explained by the learning algorithm depicted on Figure 5 (recall that IBF is the bottommost co-clone, containing only the equality relation and its powers). This (naive) algorithm simply tries to learn the target concept using the strongest possible bias, namely using hypotheses which are IBF-formulas. If it succeeds, then the process stops, otherwise the learner tries to "minimally" weaken its bias by choosing a co-clone directly above the current one in Post's lattice, resumes the learning process with this new bias, and so on.

Example 9.18 *Consider a learning problem from the beginning, hence with* $B = \mathrm{IBF}$ *for the algorithm. Assume that the first example received is* 10 *and is labelled as positive. Then the only conjunction of equality constraints consistent with this is the empty (tautological) conjunction. Now if the learner receives the negative example* 11*, learning collapses because the target concept cannot be tautological.*

[1] Note that this general observation can be stated formally, using for instance the VC-dimension of B_1, B_2 together with generic results, notably by (Auer and Long, 1999).

Algorithm 5: A naive algorithm for changing bias in Post's lattice

```
begin
    B ← IBF;
    while true do
        use an algorithm for learning c* using bias B;
        if learning succeeds then
            return current hypothesis ĉ;
        end
        if collapse occurs then
            B ← some direct superco-clone of B;
        end
    end
end
```

Hence the learner changes bias, say, to IR_0 *(*IR_0*-formulas are equivalent to conjunctions of negative literals and equalities). Now there is a consistent hypothesis with the examples received so far, namely,* $R_F(x_2) \equiv \overline{x}_2$*, and the learning process can continue.*

The good point of this algorithm is that it is guaranteed to succeed (in finding a hypothesis consistent with the examples). This is because at worst, it will weaken its bias until reaching the co-clone BR of all relations, in which, per definition, any set of examples labelled without noise from some target concept can be consistently represented. We also wish to note that, despite our presentation in "batch-mode", this algorithm can be adapted easily to online settings. In such settings, collapse is detected when a prediction turns out to be a mistake while it was *entailed* by the examples received, given the current bias.

There are nevertheless many problems with this algorithm, each of which raises questions in this research project.

The first problem is how to select a superco-clone when changing bias, since in general there are several co-clones directly above a given one, which by definition have incomparable expressivity. An idea is to maintain several biases at the same time and use the examples received for learning in each of these simultaneously. This may be reasonable in practice, since the largest antichain in Post's lattice consists of 11 co-clones. Also note that the lattice of co-clones is defined with union (plus closure) as its join operator, which could certainly be exploited when changing from a set of biases to some common superco-clone of them (e.g., their join).

A second problem arises with the infinite chains in Post's lattice. If the learner is embarked in changing bias from, say, IS^2_{11} to IS^3_{11}, then from IS^3_{11} to IS^4_{11}, etc., we need a criterion to make it stop. An obvious criterion consists in *not* moving to IS^{n+1}_{11} when the target concept has n variables (because such a move will not give any additional expressivity), but more refined criteria should be sought for, in particular if one wants *attribute-efficient* algorithms.

The last, most important problem with this algorithm is that it resumes learning for each change of bias. If the examples received so far can be reused, this is harmless in terms of sample complexity but requires the learner to memorize all the examples received. Anyway, as we will see in Section 9.4.4, this provides a first (basic) solution to the problem of changing bias in passive settings.

The really interesting research question arises with *active* algorithms. Indeed, if a learner asks questions to the user, it is not reasonable to resume the series of questions each time the bias is changed. Hence there is a need for being able to reuse the examples received for stronger biases (may they have been acquired passively or actively). Again, we give some preliminary results about this question in Section 9.4.4.

We wish to point out that there are not two separate research projects, one for passive and one for active algorithms. Indeed, a learner which is able to change bias as Algorithm 5 will always start with biases so strong that learning will be possible without the need for active queries, and most of the time end up with a bias much weaker, for which active queries will be necessary in order

to converge (or detect collapse) reasonably fast. As an example, with IM_2 as a bias (conjunctions of unit clauses x or $\overline{x}$ and implications $x_1 \to x_2$), only equivalence queries are needed, because this is a finite language (see Section 9.4.4), while for IS_0 (monotone relations) both equivalence *and* membership queries are needed (Angluin, 1990; Bshouty, 1995).

9.4.3 Related Work

Despite the importance of selecting a good bias in learning, to the best of our knowledge few studies have considered the problem of *dynamically* choosing one, as we propose here.

Mitchell (1980, 1982) discusses in depth the notion of a bias, and some typical ways to choose one, but once and for all for the learning process. Utgoff (1986) seems to be the only author to have really addressed the question, as we do here. He discusses the problem in details, gives some generic ways to dynamically augment the current bias with new hypotheses (for concept learning), and considers related questions in settings quite different from concept learning. However, his approach is not guided by a precise set of available biases, hence it is not clear how to learn in the rather unstructured biases created by his approach.

Gordon and Perlis (1995) explore the problem of shifting bias in a consistency-based approach. Nevertheless, their approach is limited to biases defined by the variables allowed (as far as our setting is concerned), and they propose to change bias as proposed by *heuristics*, instead of guided by the examples received. Closer to our concerns, the approach by Blum et al. (1995) can also be seen as a dynamic strategy for changing bias, though it was introduced for other purposes (making algorithms attribute-efficient). In their approach, the learner starts with a bias containing all concepts on 0 variables (hence, with a tautology and an unsatisfiable formula as the only available hypotheses). When collapse occurs, it moves on to a bias over 1 variable, precisely one which it knows to be relevant, and so on.

Finally, Sebag (1996) proposes an approach which avoids the need for selecting a bias at learning time. Rather, at the time when they are used (for classification), the hypotheses learnt can be interpreted as if they had been learnt with a specific bias. However, this approach is designed for noisy environments, and choosing the bias at classification time is especially directed by the noise observed. This makes her setting richer than ours in one sense, but on the other hand this approach does not make it possible to learn hypotheses in classes which guarantee efficient reasoning as much as possible (third item in Section 9.4.1). It would anyway be an interesting research direction to try to *extend* her approach to meet this requirement.

As a final remark here, note that changes of bias as we propose here have an obvious counterpart for the well-known *support vector machines* (SVMs), for which the space of available kernels indeed defines a space of available biases. Getting inspiration from the rich literature on SVMs would certainly help in the present research project. For a good introduction to SVMs, we refer the reader to Haykin (1999, Chapter 6).

9.4.4 Some Easy Results

We end this chapter by giving a flavour of some simple results which can be obtained for the problem of dynamically changing bias. The first result concerns passive learning. We do not insist on the precise definition of the learning problems for which it is true (think, e.g., of PAC-learning the best approximation of the target concept in the given bias).

Proposition 9.19 *Let* LEARNING *be a learning problem and $\mathcal{C}$ be a class of concepts. Let C_1, C_2 be two co-clones with $C_1 \subseteq C_2$, and assume that there is a passive algorithm for the problem* LEARNING$(\mathcal{C}|C_1)$ *(resp.* LEARNING$(\mathcal{C}|C_2)$*) with sample complexity f_1 (resp. $f_2 \geq f_1$). Then learning with bias C_1, then changing to C_2 requires no more examples than learning with C_2 from the beginning.*

Proof Simply run the algorithm for C_1 until a collapse occurs, having received examples E^+, E^- (say, $m \leq f_1$ in total). Then run the algorithm for C_2 from scratch, but replacing the first m

requests for examples by pop operations on E^+, E^-. Then run this algorithm to completion by requesting new examples as necessary. Since examples are received passively and hence, independently from the bias, the algorithm for B_2 will be as happy with the first algorithm's examples as with any other and hence, not request more than $f_2 - m$ additional examples. □

As far as passive learning is concerned, we can even give a more general approach, for *finite* biases. Consider for instance a shift of bias from 2CNF to 3CNF. As has been well-known for long (Valiant, 1984), the class of kCNF formulas is efficiently PAC-learnable from positive examples only using a consistency approach: store all clauses of size up to k, and each time a positive example e is received, remove all clauses C not satisfied by e. At any time the conjunction of all remaining clauses is the most specific kCNF formula consistent with all examples. Since the VC-dimension of kCNF is less than $\log_2 2^{3^k\binom{n}{k}} = 3^k\binom{n}{k}$ (hence polynomial), it follows from generic arguments (Natarajan, 1991) that this algorithm PAC-learns kCNF formulas.

Clearly, if the learner is given only positive examples, collapse will never occur[2]. But if it is given negative examples as well, collapse will occur exactly when one of them satisfies all remaining clauses. Hence the learner must only check each negative example received so far each time it updates its set of clauses, terminate on collapse if one satisfies it, and otherwise ignore negative examples. Observe that checking negative examples can be made lazily, by making each one of them point to the list of clauses which it falsifies, and detecting collapse when one such list is emptied.

A simple algorithm for changing bias from 2CNF to 3CNF is as follows. First store all clauses up to size 2 and receive examples until collapse (if ever). If collapse occurs, add to the set of clauses all clauses of size exactly 3 which are satisfied by all positive examples received so far, and go on with this new set of clauses. Observe that all superclauses of remaining 2-clauses can be added when changing bias, but all others must be checked against all positive examples received so far, and the data structure for negative examples must be updated. This makes the algorithm nonefficient in models in which an unbounded number of examples may arrive, e.g., in the online learning model.

In the end, everything is as if the learner had tried to learn a 3CNF from the beginning. Even the computational effort is asymptotically the same.

We finally give an easy result about *active* learning. Recall that a concept c si said to be *consistent* with two disjoint sets of examples E^+, E^- if $e \models c$ holds for all $e \in E^+$, and $e \not\models c$ holds for all $e \in E^-$.

Definition 9.20 *Let B be a bias, and assume that the learner has received the sets of positive (resp. negative) examples E^+ (resp. E^-). A membership query on e is said to be* informative *if there are (strictly) less hypotheses consistent with $E^+ \cup \{e\}, E^-$ (resp. with $E^+, E^- \cup \{e\}$) than with E^+, E^- in B.*

In words, a membership query on e is informative if whatever the answer of the oracle, there is at least one hypothesis ruled out.

Example 9.21 *Let B be the class of all monotone formulas, let $E^+ = \{110\}$ and $E^- = \{000\}$. The concepts $c_\wedge = x_1 \wedge x_2$ and $c_\vee = x_1 \vee x_2$ (among others) are consistent with E^+, E^-. Then a membership query on* 010 *is informative, because if* 010 *turns out to be a positive example of the target concept, $c_\wedge$ will not be consistent any more, and if it turns out to be a negative example, $c_\vee$ will not be consistent any more. On the other hand, a membership query on* 111 *is not informative, because it is necessarily a positive example given the bias B (this is because it is componentwise greater than or equal to* 110*, which is itself a positive example).*

Proposition 9.22 *Assume E^+, E^- have been received, and let B_1, B_2 be two biases with $B_1 \subseteq B_2$. If a membership query on e is informative for B_1, then it is informative for B_2.*

[2]Our approach is meaningless in this case, since the concept learnt by Algorithm 5 will always be the tautological one.

Proof By definition of informativeness, there is a concept c^+ in B_1 which is consistent with E^+, E^- but inconsistent with $E^+ \cup \{e\}, E^-$. From $B_1 \subseteq B_2$ we get $c^+ \in B_2$. The dual reasoning can be applied to some concept c^-. Hence the query is informative for B_2. □

This result can be straightforwardly extended to "half-informativeness", that is, to the case when a query will make the version space shrink only if the example is positive, or only when it is negative. The proof also shows that at least as many concepts are ruled out from B_2 as from B_1.

To summarize, the results presented in this section leave hope that there may be algorithms which are indeed able to change bias efficiently, that is, by reusing the examples already obtained, both in the passive and in the active settings. A good case study to start with would be the change from the bias of monotone formulas to that of Horn formulas in the setting of exact identification, since both classes are known to *require* active queries, but on the other hand efficient active algorithms are known for both (Angluin et al., 1992; Bshouty, 1995).

Chapter 10

Summary and More Perspectives

Contents

We now summarize the contributions and research projects presented in this thesis, and give some further, more general perspectives. Section 10.2.1 summarizes the research projects presented in Chapters 5–9 which, in our opinion, constitute interesting subjects for a master's thesis, and similarly for PhD subjects in Section 10.2.2.

10.1 Summary of Contributions

We have presented results about various computational problems in Artificial Intelligence, for agents which use symbolic, propositional representations of their environment.

We have first presented contributions about decision-making with stochastic actions, both when the model is known by the agent (Chapter 5) and when it must be learnt from experience (Chapter 6). The originality of these contributions is to consider action models represented in very compact languages, namely, PPDDL and Probabilistic STRIPS Operators, and in particular in languages which natively handle correlated effects. When the model is known, this avoids the need for converting natural representations of actions to Dynamic Bayesian Networks, a conversion which is not always efficient. When the model is not known (initially), such models emphasize the need for learners which are able to disambiguate the effects hidden in observed transitions of the environment.

We have also presented contributions in preference elicitation (Chapter 7). Our contributions concern conditional preference networks, a very popular language for which no satisfactory elicitation strategy was previously known. Our results give very efficient strategies, which are nearly optimal: they use active queries but we have shown that these are necessary, and the number of queries which they use is close to optimal. In particular, the strategies presented allow an agent to efficiently learn preference networks which depend on a few variables, even if objects are described over thousands of attributes.

Finally, we have discussed how the framework of co-clones and Post's lattice allows agents using propositional knowledge bases to be self-aware of their computational abilities (Chapter 8). We

have however pointed at some limits of co-clones in this respect, and proposed a refinement of this notion. Using constructions now standard in function algebra, we have developed a theory which parallels that of co-clones and polymorphisms. This theory provides tools which allow one to indeed use our frozen co-clones for complexity analyses, and agents to reason on the exact complexity of their algorithms.

We have placed all these studies in a general context by showing to what extent the languages and problems studied are relevant to the design of "real-world" agents (Chapter 4). We have also discussed what other problems, studied in the literature but not in this thesis, are especially relevant.

10.2 Summary of Perspectives

In each chapter of this document, we have raised a number of perspectives and research projects which build on the work presented. We now summarize those which we think most important, distinguishing *master's thesis subjects* from *PhD subjects*.

10.2.1 Subjects for Master's Theses

By a "master's thesis subject", we mean a project which addresses a specific problem, and for which results can *a priori* be obtained after a reasonable amount of work, as in a 6-months work by a student following a master in computer science (the results so obtained may of course justify continuation as a PhD).

Master's Thesis Subject 1: Improvements of RBAB

Study the behaviour of Algorithm RBAB in large-scale experiments. Evaluate the improvements obtained when using other data structures, in particular, affine algebraic decision diagrams. Propose improvements based on precomputing parts of the transition matrix. Propose approximate approaches using the incremental feature of the rules in RBAB.

More details are given in Chapter 5, Page 51.

Master's Thesis Subject 2: Relational MDPs

Propose extensions of RBAB able to compute policies for relational MDPs, in particular as represented in (non grounded) PPDDL.

More details are given in Chapter 5, Page 51.

Master's Thesis Subject 3: Decentralized MDPs

Investigate the possibility of distributing the computation of joint policies for decentralized MDPs, using the backup rules of RBAB. Investigate in particular privacy issues (agents reluctant to disclose their actions) and the possibility of detecting inconsistent joint actions (with potential applications to the design of multiagent systems and to cooperative multiagent systems).

More details are given in Chapter 5, Page 51.

Master's Thesis Subject 4: Control with Exogenous Events

Investigate the use of F-values for control policies which take into account short-term predictable exogenous events. Explore related issues in the framework of multiagent systems, where the decisions of other agents (e.g., to help) can be seen as such events. Draw comparisons with approaches which use "epistemic" variables.

More details are given in Chapter 5, Page 52.

Master's Thesis Subject 5: Learning with Restricted Membership Queries

Study the problem of learning with membership queries restricted to concern the objects in a given database, or satisfying a given set of constraints. Investigate generic concept learning problems and (ordinal) preference elicitation problems. Aim at both generic results and results for specific classes of concepts or preference relations.

More details are given in Chapter 7, Page 86.

Master's Thesis Subject 6: Complexity Classifications with Frozen Co-Clones

Improve tools based on frozen co-clones, for classifying the complexity of problems which need more fine-grained reductions than those provided by (regular) implementations. Start by revisiting the known complexity classification of inference by circumscription with frozen co-clones.

More details are given in Chapter 8, Page 100.

Master's Thesis Subject 7: Easiest NP-Complete Satisfiability Problems

Determine the smallest frozen co-clones which generate co-clones BR and IN. Derive the two easiest *a priori* NP-complete satisfiability problems, and investigate whether any other is indeed (strictly) more difficult. Draw a parallel with the minimal NP-complete problem known in descriptive complexity.

More details are given in Chapter 8, Page 101.

10.2.2 Subjects for PhD Theses

By a "PhD subject", we mean a project which addresses a broad problem, for which deep studies are to be performed without the results being "known in advance", but which in our opinion has a potentially important impact, as could be proposed for a 3-years PhD in computer science.

PhD Subject 1: Policy Revision

Investigate the problem of revising policies for MDPs, with frameless action values as a starting point. Study to what extent it is possible to represent policies in such a way that they can be revised at any horizon. Investigate the tradeoff between the size of representations and the cost of revision (versus computation from scratch). Investigate the use of the proposed schemes in model-based reinforcement learning.

More details are given in Chapter 5, Page 52.

PhD Subject 2: Learning Action Models

Propose approaches for reinforcement learning with propositional Probabilistic STRIPS Operators as a language for representing actions, starting from results about k-closed sets of effects. Extend these approaches to *relational* reinforcement learning, where the problem of ambiguous effects is particularly important. In both settings, aim at theoretical performance guarantees, and otherwise investigate heuristic approaches and nonlearnability results[1].

More details are given in Chapter 6, Page 74.

[1] As a side note, nonlearnability results are of interest for cryptography, for which models of attacks and attackers can also be formalized as learning protocols.

PhD Subject 3: Reinforcement Learning with Ordinal Preferences

Investigate models for learning from experience, when the rewards are given by hidden, ordinal preference relations. Start with ordinal relations which are total orders on states, and with a formalization as a reinforcement learning problem. More generally, investigate means for lifting results known in the setting of Boolean concept learning to reinforcement learning, by investigating exploration strategies which can be deduced from active learning strategies, and guarantees on suboptimal behaviours which can be deduced from results on sample complexity.

More details are given in Chapter 7, Page 87.

PhD Subject 4: Learnability Map in Post's Lattice

Determine a learnability map for co-clones. Along the way, investigate general transformations between problems, giving reductions between problems given a fixed bias or target class, and between biases or target classes for a fixed problem. Pay a particular attention to recently introduced models such as the KWIK framework, and to precise performance criteria such as attribute-efficiency or optimal query complexity. Generalize as much as possible the results to settings other than Post's lattice.

More details are given in Chapter 9, Page 106.

PhD Subject 5: Change of Bias in Post's Lattice

Investigate the problem of dynamically changing bias in Post's lattice. Determine whether there are general results which allow to reuse the examples received so far when considering a new bias. Consider precise classes of formulas (co-clones) as well. Investigate the possibility of dynamically changing bias in frameworks more general than Post's lattice, for instance with arbitrary classes of CNF formulas. Derive a notion of unbiased learning, and investigate its properties.

More details are given in Chapter 9, Page 108.

10.3 More Perspectives

We conclude this thesis with some more perspectives for future work, focusing on topics which have not been directly addressed in this document. This list is of course far from being an exhaustive list of interesting perspectives.

10.3.1 Other Languages

An interesting direction for future work would be to investigate other languages for representing knowledge, beliefs, preferences, and actions. We already mentioned the need for relational languages for decision-making, like PPDDL. As far as computing policies is concerned, relational languages make it possible to compute a policy once and for all for, say, stacking up three blocks among an arbitrary number of blocks, instead of computing a policy for each possible number. As far as reinforcement learning is concerned, trying to induce relational models from experience allows to generalize what is learnt about the actions on, say, Block "A", to any block. Handling relational actions models, in both settings, is a difficult problem, but some solutions have been proposed in the literature which give interesting directions (for instance Džeroski et al. (2001); Pasula et al. (2007); Sanner and Boutilier (2009); Lang and Toussaint (2010)).

Sticking to the propositional setting, another interesting direction is to consider general circuits, instead of formulas, for representing knowledge. When the gates allowed are restricted to be taken from a given language, such formulas correspond to Boolean functions in some *clone* of Post's lattice. It follows that we can derive results similar to complexity classifications known for Γ-formulas. Example problems which have been (recently) classified for circuits with restricted gates

are implication (Beyersdorff et al., 2009) and abduction (Creignou et al., 2010a). Also interesting are the learnability problems for such restricted classes of circuits, since clearly most of our study in Chapter 9 can be done also for these classes.

Concerning restrictions of propositional logic, this thesis focused on fragments defined by *local* properties, that is, by properties which all conjuncts in the formula must satisfy. Other types of restrictions are worth studying, namely restrictions on the whole formula (on the way constraints interact, for instance). A survey of restrictions defined over the graphs of formulas is given by Scarcello et al. (2008).

Another generic direction about representation languages is the setting of *complete* languages, that is, languages able to represent any piece of knowledge. A number of interesting syntactic restrictions (notably, on NNF formulas) have been studied in depth by Darwiche and Marquis (2002), and others have been proposed more recently by Fargier and Marquis (2008, 2009). In particular, Fargier and Marquis (2008) propose to obtain complete fragments by combining tractable (incomplete) fragments, which makes their setting close to ours. The particular case of binary decision diagrams restricted to propositional fragments has also been studied in a series of papers (in particular by Schachte et al. 2010).

Finally, a natural direction of research is to investigate action models whose representation is restricted to some propositional fragments, for instance, to restrict conditions to be Horn clauses for probabilistic STRIPS operators. Nevertheless, a preliminary study has shown that such restrictions do not help much in computing optimal policies (Lesner and Zanuttini, 2010b, in French).

10.3.2 Other Problems

As concerns further computational problems to investigate, we believe that our study about disambiguating effects in observed transitions may settle bases for studying factored *partially observable* Markov Decision Processes (Kaelbling et al., 1998). Indeed, in partially observable MDPs, there is an ambiguity about the current *state*, for resolving which our techniques may be helpful. Partially observable MDPs are of premier importance for autonomous agents, because they formalize these situations in which the whole current state cannot be perceived at control time, which is the case in almost all real-world applications.

Other essential problems are those related to learning with *noise* in the examples received, as is the case in most realistic applications. As we have already pointed out, it would be interesting in particular to extend the approach by Sebag (1996) for delaying the selection of biases while using Post's lattice and, more generally, our approach in Chapter 9.

Finally, we have also mentioned the need for agents to have "social abilities" in most applications. It would be interesting to study the related problems in restricted fragments of logic, as has been done, for instance, for argumentation (Creignou et al., 2010b). Interesting perspectives in game theory, with restrictions on the languages in which agents can express their preferences on the issues of the game, are also given by Bonzon (2007) (see also Bonzon et al. 2009).

10.3.3 Implementing the Agents

To conclude, an interesting perspective for future work is to indeed realize the agents which we described (at a high level) in Chapter 4. Implementing such agents would be the occasion to gather together many results and algorithms developed by researchers in Artificial Intelligence about symbolic representations. As we already mentioned, in our opinion this is a complete research project *per se*, including research about architectures, selection of descriptors, human-machine communication, etc.

Without any doubt, realizing such an application would be fruitful for research in the areas studied in this thesis, by providing a concrete application with which to experiment symbolic and attribute/value approaches to Artificial Intelligence. Without any doubt either, it would point at the many difficulties which arise when tackling real-world applications, and raise many further research questions.

Bibliography

Alekhnovich, Misha, Braverman, Mark, Feldman, Vitaly, Klivans, Adam R., and Pitassi, Toniann (2008). The complexity of properly learning simple concept classes. *Journal of Computer and System Sciences*, 74:16–34.

Amilhastre, Jérôme, Fargier, Hélène, and Marquis, Pierre (2002). Consistency restoration and explanations in dynamic CSPs – application to configuration. *Artificial Intelligence*, 135(1–2):199–234.

Angluin, Dana (1988). Queries and concept learning. *Machine Learning*, 2:319–342.

Angluin, Dana (1990). Negative results for equivalence queries. *Machine Learning*, 5(2):121–150.

Angluin, Dana, Frazier, Michael, and Pitt, Leonard (1992). Learning conjunctions of Horn clauses. *Machine Learning*, 9(2–3):147–164.

Auer, Peter and Long, Philip M. (1999). Structural results about on-line learning models with and without queries. *Machine Learning*, 36:147–181.

Bahar, Iris, Frohm, E., Gaona, C., Hachtel, Gary, Macii, Enrico, Pardo, Abelardo, and Somenzi, Fabio (1997). Algebric decision diagrams and their applications. *Formal Methods in System Design*, 10(2–3):171–206.

Barbanchon, Régis and Grandjean, Étienne (2004). The minimal logically-defined NP-complete problem. In Diekert, Volker and Habib, Michel, editors, *Proc. 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, Lecture Notes in Computer Science, pages 338–349. Springer.

Beyersdorff, Olaf, Meier, Arne, Thomas, Michael, and Vollmer, Heribert (2009). The complexity of propositional implication. *Information Processing Letters*, 109(18):1071–1077.

Biere, Armin, Heule, Marijn, van Maaren, Hans, and Walsh, Toby, editors (2009). *Handbook of Satisfiability*. Number 185 in Frontiers in Artificial Intelligence and Applications. IOS Press.

Blum, Avrim, Hellerstein, Lisa, and Littlestone, Nick (1995). Learning in the presence of finitely of infinitely many irrelevant attributes. *Journal of Computer and System Sciences*, 50:32–40.

Böhler, Elmar, Creignou, Nadia, Reith, Steffen, and Vollmer, Heribert (2004). Playing with Boolean blocks, part II: Constraint satisfaction problems. *SIGACT News*, 35(1):22–35.

Bonzon, Élise (2007). *Modélisation des interactions entre agents rationnels : les jeux booléens*. PhD thesis, Université Paul Sabatier.

Bonzon, Élise, Lagasquie-Schiex, Marie-Christine, Lang, Jérôme, and Zanuttini, Bruno (2009). Compact preference representation and Boolean games. *Journal of Autonomous Agents and Multi-Agent Systems*, 18(1):1–35.

Boussard, Matthieu, Bouzid, Maroua, Mouaddib, Abdel-Illah, Sabbadin, Régis, and Weng, Paul (2010). Non-standard criteria. In Sigaud, Olivier and Buffet, Olivier, editors, *Markov Decision Processes in Artificial Intelligence*, chapter 10, pages 319–359. ISTE/Wiley.

Boutilier, Craig, Brafman, Ronen I., Domshlak, Carmel, Hoos, Holger H., and Poole, David (2004). CP-nets: A tool for representing and reasoning with conditional *ceteris paribus* preference statements. *Journal of Artificial Intelligence Research*, 21:135–191.

Boutilier, Craig, Dean, Thomas, and Hanks, Steve (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

Boutilier, Craig, Regan, Kevin, and Viappiani, Paolo (2010). Simultaneous elicitation of preference features and utility. In Fox, Maria and Poole, David, editors, *Proc. 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 1160–1167. AAAI Press.

Brafman, Ronen I. and Tennenholtz, Moshe (2002). R-max — A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

Brueggemann, Tobias and Kern, Walter (2004). An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science*, 329(1–3):303–313.

Bshouty, Nader H. (1995). Exact learning of Boolean functions via the monotone theory. *Information and Computation*, 123:146–153.

Byskov, Jesper Makholm, Madsen, Bolette Ammitzbøll, and Skjernaa, Bjarke (2005). New algorithms for exact satisfiability. *Theoretical Computer Science*, 332(1–3):515–541.

Chevaleyre, Yann, Koriche, Frédéric, Lang, Jérôme, Mengin, Jérôme, and Zanuttini, Bruno (2010). Learning ordinal preferences on multiattribute domains: The case of CP-nets. In Fürnkranz, Johannes and Hüllermeier, Eyke, editors, *Preference Learning*, pages 273–296. Springer.

Creignou, Nadia, Khanna, Sanjeev, and Sudan, Madhu (2001). *Complexity classifications of Boolean constraint satisfaction problems*. SIAM Monographs on Discrete Mathematics and Applications. SIAM.

Creignou, Nadia, Kolaitis, Phokion G., and Vollmer, Heribert, editors (2008a). *Complexity of Constraints*. Lecture Notes in Computer Science. Springer.

Creignou, Nadia, Kolaitis, Phokion G., and Zanuttini, Bruno (2008b). Structure identification of Boolean relations and plain bases for co-clones. *Journal of Computer and System Sciences*, 74(7):1103–1115.

Creignou, Nadia, Schmidt, Johannes, and Thomas, Michael (2010a). Complexity of propositional abduction for restricted sets of Boolean functions. In Lin, Fangzhen, Sattler, Ulrike, and Truszczynski, Miroslaw, editors, *Proc. 12th International Conference on Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 8–16. AAAI Press.

Creignou, Nadia, Schmidt, Johannes, Thomas, Michael, and Woltran, Stefan (2010b). Sets of boolean connectives that make argumentation easier. In Janhunen, Tomi and Niemelä, Ilkka, editors, *Proc. 12th European Conference on Logics in Artificial Intelligence (JELIA 2010)*, pages 117–129. Springer.

Creignou, Nadia and Zanuttini, Bruno (2006). A complete classification of the complexity of propositional abduction. *SIAM Journal on Computing*, 36(1):207–229.

Dalmau, Victor (1999). A dichotomy theorem for learning quantified Boolean formulas. *Machine Learning*, 35(3):207–224.

Dantsin, Evgeny, Goerdt, Andreas, Hirsch, Edward A., Kannan, Ravi, Kleinberg, Jon, Papadimitriou, Christos, Raghavan, Prabhakar, and Schöning, Uwe (2002). A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83.

Darwiche, Adnan and Marquis, Pierre (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264.

Degris, Thomas, Sigaud, Olivier, and Wuillemin, Pierre-Henri (2006). Learning the structure of factored Markov Decision Processes in reinforcement learning problems. In Cohen, William W. and Moore, Andrew, editors, *Proc. 23rd International Conference on Machine Learning (ICML 2006)*, pages 257–264. ACM.

del Val, Alvaro (1995). An analysis of approximate knowledge compilation. In Mellish, Chris S., editor, *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 830–836. Morgan Kaufmann.

Dimopoulos, Yannis, Michael, Loizos, and Athienitou, Fani (2009). Ceteris Paribus preference elicitation with predictive guarantees. In Boutilier, Craig, editor, *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1890–1895.

Domshlak, Carmel and Brafman, Ronen I. (2002). CP-nets — reasoning and consistency testing. In Fensel, Dieter, Giunchiglia, Fausto, McGuinness, Deborah L., and Williams, Mary-Anne, editors, *Proc. 8th International Conference on Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 121–132. Morgan Kaufmann.

Durand, Arnaud, Hermann, Miki, and Nordh, Gustav (2009). Trichotomy in the complexity of minimal inference. In Pitts, Andrew, editor, *Proc. 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009)*, pages 387–396. IEEE Computer Society.

Džeroski, Sašo, De Raedt, Luc, and Driessens, Kurt (2001). Relational reinforcement learning. *Machine Learning*, 43:7–52.

Eiter, Thomas and Gottlob, Georg (1995). The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42.

Fargier, Hélène and Marquis, Pierre (2008). Extending the knowledge compilation map: Krom, Horn, affine and beyond. In Fox, Dieter and Gomes, Carla P., editors, *Proc. 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 442–447. AAAI Press.

Fargier, Hélène and Marquis, Pierre (2009). Knowledge compilation properties of tress-of-BDDs, revisited. In Boutilier, Craig, editor, *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 772–777.

Feng, Zhengzhu and Hansen, Eric A. (2002). Symbolic heuristic search for factored Markov decision processes. In *Proc. 18th National Conference on Artificial Intelligence (AAAI 2002)*, pages 455–460. AAAI Press.

Fürnkranz, Johannes and Hüllermeier, Eyke, editors (2010). *Preference Learning*. Springer.

Garcia, Frédérick and Rachelson, Emmanuel (2010). Markov decision processes. In Sigaud, Olivier and Buffet, Olivier, editors, *Markov Decision Processes in Artificial Intelligence*, chapter 1, pages 3–38. ISTE/Wiley.

Givan, Robert, Dean, Thomas, and Greig, Matthew (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, pages 163–223.

Gordon, Diana and Perlis, Donald (1995). Explicitly biased generalization. In Ram, Ashwin and Leake, David B., editors, *Goal-Driven Learning*, chapter 13, pages 321–354. MIT Press.

Guestrin, Carlos, Koller, Daphne, Parr, Ronald, and Venkataraman, Shobha (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468.

Haddad, Lucien and Simons, G. E. (2003). On intervals of partial clones of Boolean partial functions. In Stankovic, Radomir S., Waho, Takao, and Miller, Michael, editors, *Proc. 33rd IEEE International Symposium on Multiple-Valued Logic (ISMVL 2003)*, pages 315–322. IEEE Computer Society.

Haykin, Simon (1999). *Neural Networks — A Comprehensive Foundation*. Prentice-Hall.

Hébrard, Jean-Jacques and Zanuttini, Bruno (2003). An efficient algorithm for Horn description. *Information Processing Letters*, 88(4):177–182.

Hermann, Miki and Pichler, Reinhard (2010). Counting complexity of propositional abduction. *Journal of Computer and System Sciences*, 76:634–649.

Hoey, Jesse, St-Aubin, Robert, Hu, Alan J., and Boutilier, Craig (1999). SPUDD: Stochastic planning using decision diagrams. In Laskey, Kathryn B. and Prade, Henri, editors, *Proc. 15th Conference on Uncertainty in Artificial Intelligence (UAI 1999)*, pages 279–288. Morgan Kaufmann.

Jeavons, Peter, Cohen, David, and Gyssens, Marc (1997). Closure properties of constraints. *Journal of the ACM*, 44(4):527–548.

Kaelbling, Leslie Pack, Littman, Michael L., and Cassandra, Anthony R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134.

Kakade, Sham Machandranath (2003). *On the sample complexity of reinforcement learning*. PhD thesis, University College London.

Kavvadias, Dimitris J. and Sideri, Martha (1998). The inverse satisfiability problem. *SIAM Journal on Computing*, 28(1):152–163.

Kearns, Michael and Koller, Daphne (1999). Efficient reinforcement learning in factored MDPs. In Dean, Thomas, editor, *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 740–474. Morgan Kaufmann.

Kearns, Michael, Mansour, Yishay, and Ng, Andrew Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49:193–208.

Kearns, Michael and Singh, Satinder (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2–3):209–232.

Kearns, Michael J. and Schapire, Robert E. (1994). Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Sciences*, 48(3):464–497.

Kearns, Michael J., Schapire, Robert E., and Sellie, Linda M. (1994). Toward efficient agnostic learning. *Machine Learning*, 17:115–141.

Kersting, Kristian, Van Otterlo, Martijn, and De Raedt, Luc (2004). Bellman goes relational. In Brodley, Carla E., editor, *Proc. 21st International Conference on Machine Learning (ICML 2004)*, pages 465–472. ACM.

Khardon, Roni and Roth, Dan (1996). Reasoning with models. *Artificial Intelligence*, 87(1-2):187–213.

Koriche, Frédéric and Zanuttini, Bruno (2009). Learning conditional preference networks with queries. In Boutilier, Craig, editor, *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1930–1935.

Koriche, Frédéric and Zanuttini, Bruno (2010). Learning conditional preference networks. *Artificial Intelligence*, 174:685–703.

Kushmerick, Nicholas, Hanks, Steve, and Weld, Daniel S. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286.

Lang, Jérôme and Mengin, Jérôme (2009). The complexity of learning separable ceteris paribus preferences. In Boutilier, Craig, editor, *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 848–853. IJCAI.

Lang, Tobias and Toussaint, Marc (2010). Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39:1–49.

Lau, Dietlinde (2006). *Function Algebras on Finite Sets — A basic course on many-valued logic and clone theory*. Springer Monographs in Mathematics. Springer.

Lesner, Boris and Zanuttini, Bruno (2010a). Apprentissage par renforcement de pdm factorisés avec effets corrélés. In *Proc. 5es Journées Francophones Planification Décision Apprentissage (JFPDA 2010)*. In French.

Lesner, Boris and Zanuttini, Bruno (2010b). Résolution exacte et approchée de problèmes de décision markoviens formulés en logique propositionnelle. *Revue d'intelligence artificielle*, 24(2):131–158. In French.

Lesner, Boris and Zanuttini, Bruno (2011). Efficient policy construction for MDPs represented in probabilistic PDDL. In Bacchus, Fahiem, Domshlak, Carmel, Edelkamp, Stefan, and Helmert, Malte, editors, *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pages 146–153. AAAI Press.

Li, Lihong, Littman, Michael L., Walsh, Thomas J., and Strehl, Alexander L. (2011). Knows what it knows: a framework for self-aware learning. *Machine Learning*, 82:399–443.

Liberatore, Paolo (1998). *Compilation of Intractable Problems and Its Application to Artificial Intelligence*. PhD thesis, Università degli Studi di Roma "La Sapienza".

Littlestone, Nick (1988). Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318.

Littman, Michael L. (1997). Probabilistic propositional planning: Representations and complexity. In Kuipers, Benjamin J. and Webber, Bonnie, editors, *Proc. 14th National Conference on Artificial Intelligence (AAAI 1997)*, pages 748–754. AAAI Press/MIT Press.

McCarthy, John (1980). Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1–2):27–39.

McCarthy, John (1986). Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116.

Mitchell, Tom M. (1980). The need for biases in learning generalizations. Technical Report CBM-TR 5-110, Rutgers University.

Mitchell, Tom M. (1982). Generalization as search. *Artificial Intelligence*, 18:203–226.

Monien, Burkhard and Speckenmeyer, Ewald (1985). Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295.

Natarajan, Balas K. (1991). *Machine Learning — A theoretical approach*. Morgan Kaufmann.

Nordh, Gustav (2004). A trichotomy in the complexity of propositional circumscription. In Baader, Franz and Voronkov, Andrei, editors, *Proc. 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2004)*, pages 257–269. Springer.

Nordh, Gustav and Zanuttini, Bruno (2005). Propositional abduction is almost always hard. In Kaelbling, Leslie Pack, editor, *Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 534–539.

Nordh, Gustav and Zanuttini, Bruno (2008). What makes propositional abduction tractable. *Artificial Intelligence*, 172:1245–1284.

Nordh, Gustav and Zanuttini, Bruno (2009). Frozen boolean partial co-clones. In Hanyu, Takahiro, editor, *Proc. 39th International Symposium on Multiple-Valued Logic (ISMVL 2009)*, pages 120–125. IEEE.

Pasula, Hanna M., Zettlemoyer, Luke S., and Kaelbling, Leslie Pack (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352.

Post, Emil (1941). The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122.

Puterman, Martin L. (1994). *Markov Decision Processes — Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley.

Rintanen, Jussi (2003). Expressive equivalence of formalism for planning with sensing. In Giunchiglia, Enrico, Muscettola, Nicola, and Nau, Dana S., editors, *Proc. 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 185–194. AAAI Press.

Rodrigues, Christophe, Gérard, Pierre, Rouveirol, Céline, and Soldano, Henry (2010). Incremental learning of relational action rules. In *Proc. 9th International Conference on Machine Learning and Applications (ICMLA 2010)*, pages 451–458. IEEE Computer Society.

Rossi, Francesca, van Beek, Peter, and Walsh, Toby, editors (2006). *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier.

Sachdev, Manish P. (2007). On learning of *ceteris paribus* preference theories. Master's thesis, North Carolina State University.

Safaei, Javad and Ghassem-Sani, Gholamreza (2007). Incremental learning of planning operators in stochastic domains. In van Leeuwen, Jan, Italiano, Giuseppe F., van der Hoek, Wiebe, Meinel, Christoph, Sack, Harald, and Plasil, Frantisek, editors, *Proc. 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007)*, pages 644–655. Springer.

Sanner, Scott and Boutilier, Craig (2009). Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5–6):748–788.

Sanner, Scott and McAllester, David (2005). Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In Leslie Pack Kaelbling, Alessandro Saffiotti, editor, *Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 1384–1390. Professional Book Center.

Scarcello, Francesco, Gottlob, Georg, and Greco, Gianluigi (2008). Uniform constraint satisfaction problems and database theory. In Creignou, Nadia, Kolaitis, Phokion G., and Vollmer, Heribert, editors, *Complexity of Constraints*, Lecture Notes in Computer Science, pages 156–195. Springer.

Schachte, Peter, Søndergaard, Harald, Whiting, Leigh, and Henshall, Kevin (2010). Information loss in knowledge compilation: A comparison of Boolean envelopes. *Artificial Intelligence*, 174(9–10):585–596.

Schaefer, Thomas J. (1978). The complexity of satisfiability problems. In *Proc. 10th Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 216–226. ACM.

Schnoor, Henning and Schnoor, Ilka (2008). Partial polymorphisms and constraint satisfaction problems. In Creignou, Nadia, Kolaitis, Phokion G., and Vollmer, Heribert, editors, *Complexity of Constraints*, Lecture Notes in Computer Science, pages 229–254. Springer.

Sebag, Michèle (1996). Delaying the choice of bias: a disjunctive version space approach. In Saitta, Lorenza, editor, *Proc. 13th International Conference on Machine Learning (ICML 1996)*, pages 444–152. Morgan-Kaufmann.

Selman, Bart and Kautz, Henry (1996). Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224.

Shanahan, Murray P. (1999). The event calculus explained. In Wooldridge, Michael J. and Veloso, Manuela, editors, *Artificial Intelligence Today*, number 1600 in Lecture Notes in Artificial Intelligence, pages 409–430. Springer.

St-Aubin, Robert, Hoey, Jesse, and Boutilier, Craig (2000). APRICODD: Approximate policy construction using decision diagrams. In Leen, Todd K., Dietterich, Thomas G., and Tresp, Volker, editors, *Proc. 13th Annual Conference on Neural Informatino Processing Systems (NIPS 2000)*, pages 1089–1095. MIT Press.

Strehl, Alexander L., Diuk, Carlos, and Littman, Michael L. (2007). Efficient structure learning in factored-state MDPs. In Holte, Robert C. and Howe, Adele, editors, *Proc. 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 645–650. AAAI Press.

Strehl, Alexander L., Li, Lihong, and Littman, Michael L. (2006). Incremental model-based learners with formal learning-time guarantees. In *Proc. 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*. AUAI Press.

Strehl, Alexander L. and Littman, Michael L. (2008). An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74:1309–1331.

Sutton, Richard S. and Barto, Andrew G. (1998). *Reinforcement Learning — An Introduction*. MIT Press.

Utgoff, Paul E. (1986). Shift of bias for inductive concept learning. In Michalski, Ryszard S., Carbonell, Jaime, and Mitchell, Tom, editors, *Machine Learning*, volume II, chapter 5, pages 107–148. Morgan-Kaufmann.

Valiant, Leslie (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

Walsh, Toby (2000). SAT v CSP. In Dechter, Rina, editor, *Proc. 6th International Conference on Principles and Practice of Constraint Programming (CP 2000)*, pages 441–456. Springer-Verlag.

Walsh, Thomas J., Szita, István, Diuk, Carlos, and Littman, Michael L. (2009). Exploring compact reinforcement-learning representations with linear regression. In *Proc. 25th Conference on Uncertainty in Artificial Intelligence (UAI 2009)*.

Wang, Chenggang, Joshi, Saket, and Khardon, Roni (2008). First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31:431–472.

Weissman, Tsachy, Ordentlich, Erik, Seroussi, Gadiel, Verdu, Sergio, and Weinberger, Marcelo J. (2003). Inequalities for the l_1 deviation of the empirical distribution. Technical Report HPL-2003-97, Hewlett-Packard Company.

Weng, Paul (2011). Markov decision processes with ordinal rewards: Reference point-based preferences. In Bacchus, Fahiem and Domshlak, Carmel, editors, *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*. AAAI Press. To appear.

Younes, Høakan L.S. and Littman, Michael L. (2004). PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

Zanuttini, Bruno (2002a). Approximating propositional knowledge with affine formulas. In van Harmelen, Frank, editor, *Proc. 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 287–291. IOS Press.

Zanuttini, Bruno (2002b). Approximation of relations by propositional formulas: complexity and semantics. In Koenig, Sven and Holte, Robert C., editors, *Proc. 5th Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, number 2371 in Lecture Notes in Artificial Intelligence, pages 242–255. Springer.

Zanuttini, Bruno and Hébrard, Jean-Jacques (2002). A unified framework for structure identification. *Information Processing Letters*, 81(6):335–339.