Università degli Studi di Roma "La Sapienza"

Dottorato di Ricerca in Ingegneria Informatica

X Ciclo – 1998

Compilation of Intractable Problems and Its Application to Artificial Intelligence

Paolo Liberatore

Università degli Studi di Roma "La Sapienza"

Dottorato di Ricerca in Ingegneria Informatica

X Ciclo - 1998

Paolo Liberatore

Compilation of Intractable Problems and Its Application to Artificial Intelligence

Thesis Committee

Prof. Marco Schaerf (Advisor)
Prof. Maurizio Lenzerini
Prof. Pierluigi Crescenzi

Author's address:
Paolo Liberatore
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
e-mail: liberato@dis.uniroma1.it
www: http://www.dis.uniroma1.it/∼liberato

Contents

Acknowledgments

I wish to thank all the people who helped me during my Ph.D.. First of all, my deepest debt of gratitude goes to Marco Schaerf, who constantly inspired and encouraged me. It is mainly due to him if I could accomplish and finish this work. I also wish to thank the colleagues with whom I worked in the past years, and in particular my coauthors Marco Schaerf, Francesco Donini, and Marco Cadoli.

I would like to thank all the people of the Dipartimento di Informatica e Sistemistica of the University of Rome "La Sapienza", and in particular the members of the Artificial Intelligence group. I shared with them joys and pains of a working environment which has always been stimulating and which sometimes surprised us with unexpected events.

I am also indebted to Henry Kautz, who gave me his hospitality for five months at the AT&T Bell Labs.

I express my gratitude to the members of the Ph.D. Committee of the Dipartimento di Informatica e Sistemistica, who helped me in the accomplishment of this work. To Georg Gottlob and Berhnard Nebel as external referees, to Maurizio Lenzerini and Pierluigi Crescenzi as internal referees, and to Luigia Carlucci Aiello and Giorgio Ausiello as presidents.

Last but not least, special thanks to my beloved girlfriend Roberta, who encouraged me and gave me strength in difficult moments.

April 21, 1998 Paolo Liberatore

Abstract

Intractable problems are hard to solve. Since the work "has to be done", various solutions have been developed. The two classical ones are the language restriction and the approximation. The first solution is to admit input strings only if they have a certain form. The second one is to find a solution that can be sometimes incorrect. In this dissertation we concentrate on *compilation*.

The basic scenario is the following one: we have an intractable problem, thus a problem for which it is believed no polynomial algorithm exists. However, each instance of this problem is composed of two parts, one (called the *fixed part* is known in advance, while the other (called the *varying part*) only comes at execution time. Our idea is to solve the problem in two steps:

1. Take the fixed part and compile it into a new data structure;
2. Take the new data structure and the varying part, and produce the output.

If the second step can be accomplished in polynomial time, the problem is said to be compilable to polynomial time, that is, after a preprocessing, the cost of solving it is only polynomial.

The dissertation contains a formal definition of compilation and compilability of hard problems. Namely, two hierarchies of classes of problems (similar to the polynomial hierarchy) are defined. One of them is used to prove that problems are compilable, while the other one is mainly used to prove that some problems are *not* compilable.

Applications of this framework includes: compilability of AI problems, problems on graphs, compact representation of AI operators. A comparison with the non-uniform polynomial hierarchy and with the FPT classes are given.

Overview of the Dissertation

In this dissertation we formalize and use the idea of *compiling* hard problems in order to reduce the cost of solving them. The main applications are in the field of Artificial Intelligence (AI), but some examples from other fields are also shown.

The dissertation is organized as follows.

Chapter 1: Introduction The basic concepts of computational complexity are sketched. We also show how the idea of compilation can be applied to reduce the time needed to solve problems.

Chapter 2: Preliminaries This chapter contains the explanation of the basic concepts needed to understand the rest of the thesis. Namely, we revise the theory of NP completeness and the polynomial hierarchy. We also introduce some formalisms of AI: belief revision, circumscription, reasoning about actions, and diagnosis.

Chapter 3: Compilation We formalize the concept of compilation, which was so far only informally introduced. In this chapter we define the classes of compilability, which classify the problems on the basis of how much the compilation can speed-up the process of solving.

Chapter 4: Reductions Here we define a class of reductions that are more general than those used in the theory of NP completeness. These reductions are used to prove negative results. Examples of these results are also shown.

Chapter 5: Compilability Results This chapter contains the classification of the problems of AI introduced in Chapter 2 w.r.t. the possibility of reducing their complexity via a compilation.

Chapter 6: Compact Representation The problem of compact representation is the problem of establishing the amount of space needed to store the result of operators. We prove that compilability classes help us in determining whether this amount of space is polynomial.

Chapter 7: Succinctness The property of succinctness is essentially an extension of that of compact representation: indeed, it classifies formalisms on the basis of how good they are in representing knowledge in little space.

Chapter 8: Related Work In this chapter we compare our ideas with similar work presented in the literature. Namely, we compare the notion of compilability with that of non-uniform classes of complexity, and with the theory of fixed parameter tractability.

Chapter 9: Conclusions We briefly summarize the results obtained in this dissertation.

Chapter 1

Introduction

Efficiency is a key issue in modern software. In this chapter we explain how the efficiency of algorithms and the intrinsic hardness of problems is formalized. This thesis is mainly concerned on the problem of reducing the time of computation via a "preprocessing". The typical application of this idea is in Artificial Intelligence, and namely in the area of applied logics.

1.1 Complexity

Even computer amateurs clearly understand the importance of efficiency in computer applications. Sometimes, however, the idea of what is meant by "efficient" is misunderstood. Once I met an Optical Engineer, and when I said I work on efficiency of algorithms he said: "That's important. I don't understand why nowadays algorithms are so inefficient. When I start my computer, Windows 95 takes two whole minutes to boot."

Some natural interpretations of the word "efficient" are not so natural when formally defined. The first thing to do is to formally define problems and algorithms. The first word, "problem" has a quite clear meaning in English: "a question proposed for solution" (Webster). The idea is that there is an input data, and the goal is to find the solution of the problem. This is almost trivial, and should not need to be discussed any further. However, in computer science, this definition of problem is considered not satisfying.

Before giving the solution of the puzzle "what is a problem?", let us turn our attention to the word "algorithm". As it is well known, the word derives from the name of an Arab mathematics, Abu Abd-Allah ibn Musa al'Khwarizmi, which is believed to be the first to use the figure 0 as a place holder in positional base notation. The main reason we build computers is to make them to do repetitive work, such as numeric calculus, etc. Consider the problem of multiplying two numbers of 4 digit each, on a computer that

does not have multiplication, but only additions and subtractions. Writing an algorithm to solve just *one* multiplication is meaningless: the solution can be found by hand in no more than one minute, while it takes more time to program a computer for solving the same problem.

The usual scenario is: there are a number of questions (e.g. multiplications) to be solved. Solving them by hand requires too much time, or it is too expensive to make humans to do that. Since all these questions are basically the same (e.g. multiply two numbers) it makes sense to program a computer to make all these operations. The sequence of basic operations that solve all the instances of the problems is called an algorithm. This definition clarifies the distinction between a problem and an instance of the problem. For example, the multiplication of 123 and 521 is an *instance* of the problem of multiplication. An algorithm is a method of solving a problem, in the sense that it must solve *all* the instances of the problem. Just to make things clear, for the problem of multiplication:

- multiplying 123 and 521 is an instance of the problem;
- the problem is to multiply a generic pair of numbers;
- an algorithm for multiplication is a sequence of elementary operations that solves all possible instances of the problem.

that is, given a generic pair of numbers, the algorithm must be able to find the product of them.

The question for the efficiency of an algorithm is not "how fast it solves a specific instance of the problem", but "how long it takes for solving the problem" or "how long it takes to solve the instances of the problem". When the involved numbers are small, the solution is clearly easy to find (e.g. multiplying 2 and 4). The problem of efficiency arises when the inputs are large. This holds not only for the problem of multiplication: in almost all problems the time required to solve an instance increases with the size of the instance.

In the best-case scenario, the time required to solve an instance is proportional to the size of the input. If the input is a data structure that requires for example n bytes of memory to be stored, and the time required for finding the solution is $4 \cdot n$, the algorithm is in most cases to be considered good[1]. In the problems analyzed in this work, a length of time $c \cdot n$, where c is a constant, is considered small. Let us compare this with an algorithm that requires time $d \cdot 2^n$, where d is another constant:

Alg1 running time $c \cdot n$

[1]This is not always true: in database problems, where the inputs are usually huge, an algorithm of this kind is instead considered poor. In this work, we neglect these considerations.

Alg2 running time $d \cdot 2^n$

When the number n is small, the time needed to solve the problem using Alg2 may be smaller than that of Alg1. For example, if $c = 100$, $d = 1$ and $n = 4$ we have $2^4 = 16$, which is smaller than $100 \cdot 4 = 400$. However, when n increases, the performance of the second algorithm degrades quickly.

The following table reports the value of the functions $100n$, $10n^2$, and 2^n when the value of n increases. This table is almost unavoidable in a complexity book, so the reader have almost surely seen it in some other text.

n	100n	$10n^2$	2^n
5	500	250	32
10	1000	1000	1024
100	10000	100000	$1.24 \cdot 10^{30}$
1000	100000	10000000	$1 \cdot 10^{301}$

Table 1.1: Polynomial time compared to exponential time.

What is important to note in the table is that when the instance of the problem becomes large (i.e. when n increases), the most efficient algorithms are those requiring time $c \cdot n$. Algorithms with $c \cdot n^2$ are less efficient, but the solution can be always found in a reasonable time, if n is not too large. On the other side, even for small values of n, the algorithm requiring time 2^n are too slow: no one is likely to wait patiently for $1 \cdot 10^{301}$ seconds (which is equivalent to $3.40 \cdot 10^{293}$ years) for the solution of the problem, unless it is the the answer to life, the universe, and everything.

These figures are not related to the increase of computers' speed. It has been noticed that an increase of speed leads only to a constant gain that cannot contrast an exponential function.

The point is that an algorithm with running time $c \cdot n$ or $c \cdot n^2$ can always solve problems even when the input is large, while an algorithm that takes 2^n cannot solve instances over a certain size. In general, algorithms whose running time is polynomial in the size of the input (e.g. $100n^4 + 20n^2$) are always considered to be "efficient" when compared to algorithms that requires exponential time (e.g. 2^n).

Definition 1 *A* polynomial algorithm *is an algorithm whose running time is bounded by a polynomial in the size of the input.*

If x is the input to the algorithm, we denote its size as $||x||$. An algorithm is polynomial if and only if there exists a polynomial p such that the running time of the algorithm on the input x is less or equal than $p(||x||)$.

Definition 2 *A* non-polynomial algorithm *is an algorithm whose running time is not bounded by a polynomial function of the size of the input.*

This is a rough classification of *algorithms*. It is also possible to define the intrinsic hardness of *problems*. Given a problem, there may be several algorithms to solve it. Clearly, we are interested in finding the most efficient algorithm. According to Table 1.1, polynomial algorithms takes a reasonable amount of time, while exponential ones in general does not. Given a problem, the goal is to find a polynomial algorithm to solve it. However, this is not always possible. There are problems for which it has been proved that no polynomial algorithm can solve them. These problems are intrinsically harder than problems for which a polynomial algorithm exists. We classify the problems on the basis of this property:

tractable: a problem is tractable if there exists a polynomial algorithm that solves it

intractable: a problem is intractable if no polynomial algorithm can solve it

1.1.1 Historical Note

The idea that exponential algorithms are poor is not born with computers. The puzzle known as devil's chain is an example of a problem that requires exponential time to be solved. In a small book of mathematical games written back in the XIX century [Tos78], an explanation of the puzzle is given, followed by a mathematical analysis. A picture of the game is in Figure 1.1. The puzzle is made out of a number of circle rings linked to a board. There is another ring, as long as the board, and the goal of the game is to insert the long ring into all the smaller ones.

The solution of the the game is too complex to be explained in details, but an interesting observation is made about it in Tosatti's book [Tos78]. Here is a translation of what the author wrote about the time required to solve the puzzle, and how it is related to the number of rings of the chain:

> The so-called devil's chain is indeed a geometric progression, since every ring, in order to be inserted [in the long one] requires a number of moves that is two times the number needed for the previous ring. As a result, increasing a little the number of rings, the degree of hardness is doubled [for each ring], and eventually the puzzle becomes impossible to solve.
>
> A person who is very good at this game can make a move [inserting or removing a ring] each two seconds, thus 30 moves per minute. As a result, for a chain of 8 rings the 170 moves to solve it require an

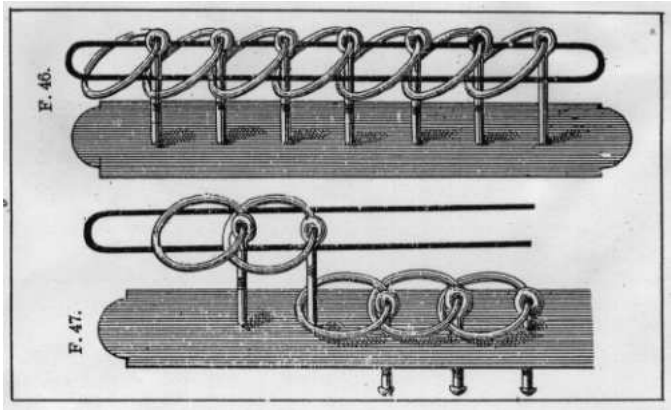


Figure 1.1: The devil's chain.

> half of a quarter of hour [sic]. For a 12-ring chain two hours would not suffice. For a 16-ring chain the work of a day is not enough. For a 25-ring chain a whole year would not suffice. For a 32-ring chain, the last ring will not be inserted even if one worked night and day for a century, as to say 100 year. The 38-ring chain would not be solved even if someone had worked incessantly without wasting a single minute from Adam's creation[2] to nowadays.

1.2 Compilation

Intractable problems are hard to solve. Since the work "has to be done", various solutions have been developed. The two classical ones are the language restriction and the approximation. The first solution is to admit input strings only if they have a certain form. The second one is to find a solution that can be sometimes incorrect. In this work we concentrate on *compilation*. The ideas presented here are new, but the idea of solving hard problems by solving parts of it in advance is as old as mathematics.

The Babylonians used a table of squares of integers in order to make multiplications easier. Namely, they had tables containing, for any integer a up to a given value, the value $a^2/4$. The operation of multiplication is time-consuming: this is still true, and modern algorithms still try to replace multiplications with additions and subtractions whenever possible. Let x and y be two integer to be multiplied. It is possible to speed up the computation using the following

[2] Which, according to the Church's cosmology in the XIX century, took place in 3761 b.C.

formula.

$$x \cdot y = \frac{(x+y)^2}{4} - \frac{(x-y)^2}{4} = \left\lfloor \frac{(x+y)^2}{4} \right\rfloor - \left\lfloor \frac{(x-y)^2}{4} \right\rfloor$$

This operation requires only one addition $x + y$, one subtraction $x - y$, the looking up on the table for the rounding of $\frac{(x+y)^2}{4}$ and $\frac{(x-y)^2}{4}$ and then just one another subtraction.

This is a typical example of compilation: to make a *single* multiplication the whole procedure is meaningless, since it requires the calculation of two squares instead of a single multiplication. However, once determined the table of squares up to a given integer, this table allows the determination of the product of many pairs $\langle x, y \rangle$ in a very efficient way.

Let us come back to modern days. We show a simple problem on graphs, where we can gain some computational advantages by a compilation phase. The problems is: given an undirected graph, determine whether two nodes are connected, that is, whether there exists a path from one node to the other one.

The typical algorithm to solve this problem requires the visit of the whole graph. If the graph contains n nodes and m edges, the total cost is proportional to $n + m$. Consider instead the following algorithm: determine the connected components of the graph, number them, and then write down a table which contains, for each node, the number of the connected component it belongs to. To determine whether two nodes a and b are connected, what is needed is just finding out to which connected components they belong: if it is the same, the two nodes are connected, otherwise they are not.

What is the advantage of this algorithm? Of course, determining the connected components is not easier than searching the graph for a path from a to b. However, if the graph is known in advance, then we can calculate the table once, and then verifying the existence of a path only takes a constant amount of time. This preprocessing (compilation) makes sense if two conditions hold:

1. The graph is known *in advance*, that is, some time before the all data is known and result is needed.

2. It is necessary to make several connectedness checks on the same graph.

If the first condition holds, compilation is convenient since it can be made during a period of time when the full (on-line) computation cannot be made yet. If the second one holds, we gain from compilation since, once determined the table, all subsequent reachability queries can be answered very efficiently.

Other situations where compilation might help come from AI and databases. A modern field in DB is the study of data-warehouses. In a nutshell, the

scenario we are considering is the following one: we have several very large databases where all our data is stored. In order to speed up some queries, a *view* on these bases is stored separately. A view is a relation. The cost of creating and maintaining this relation is quite high, but it allows for very fast answering to a given set of queries. Maintaining the data-warehouse is worthwhile if there are many queries whose cost is reduced by posing them to the data-warehouse rather than to the whole database.

All of the above scenarios considered problems solvable in polynomial time, where compilation helped in reducing the cost of execution. For example, in data-warehousing the input (the database) is usually very large, thus a polynomial algorithm (let's say n^3), does not guarantee that we can answer the query in a reasonable amount of time.

In this paper we are interested in problems that are not polynomial, and we want to investigate whether, after a compilation process, we can solve them in polynomial time.

The basic scenario is the following one: we have an intractable problem, thus a problem for which it is believed no polynomial algorithm exists. However, each instance of this problem is composed of two parts, one (called the *fixed part*) is known in advance, while the other (called the *varying part*) only comes at execution time. Our idea is to solve the problem in two steps:

1. Take the fixed part and compile it into a new data structure;

2. Take the new data structure and the varying part, and produce the output.

If the second step can be accomplished in polynomial time, the problem is said to be compilable to polynomial time, that is, after a preprocessing, the cost of solving it is only polynomial.

Our two-step approach might work well for one of the basic reasoning problems in AI: deciding whether a propositional query is implied from a propositional knowledge base. Formally, the problem can be stated as: given a knowledge base K (a set of propositional clauses), and a query Q (a propositional clause), decide whether $K \models Q$.

It is well known that this problem is intractable, when both inputs are available on-line. In many AI domains, however, a single knowledge base must be queried many times. Thus, it makes sense to compile K, putting it in a form that allows the solution of $K \models Q$ in polynomial time. If this is feasible, the problem of entailment is said to be compilable. Unfortunately, if the query can be *any* clause the problem is not compilable (this result is a slight generalization of a result of Kautz and Selman in [KS92], and is given in [CDS96]).

The same situation arises in DBs, where K is a database and Q is a (relational) query. In fact, the problem is intractable (if both the database and the query are in the input) but the same database is queried many times, thus it may take advantage from a compilation phase.

In both of the considered scenarios, the instance of the problem is divided in two parts: the knowledge or data base, which is fixed (we can assume that it varies slowly over time) and the query, that is the varying part of the input. The same database is queried over with several distinct queries. Thus, it makes sense to compile the database, in order to obtain a faster algorithm to answer queries.

More generally, we consider the following situation: there is a problem that has two inputs, one which is fixed, and one which is varying. The aim of the compilation is to preprocess only the instance of the fixed part, obtaining a *compiled structure*. Given this compiled structure, and an instance of the varying part, solving the problem should be easier (ideally, it should be feasible in polynomial time) than solving the problem directly for the given instances of the fixed and varying part. We remark that the focus is on the efficiency of the on-line algorithm. The algorithm that solves the problem given the compiled structure and the instance of the varying part should be as efficient as possible. The cost of this increase of speed is paid by the compilation phase, which can be very long.

While the general idea of compilation is a very natural one, no formal characterization of problems from this point of view has been done yet. One of the main aims of this work is to give such a formalization, and to show how it can applied to problem from several different fields, mainly from artificial intelligence.

Chapter 2

Preliminaries

In this chapter we give the definitions and notations needed to understand the rest of the work. Namely, we give the formal definition of tractability and intractability. We define the theory of NP completeness, which is a tool for classifying problems w.r.t. the time needed to solve them. We also give the definitions of the AI problems that will be studied as applications of the study of compilability: belief revision, circumscription, reasoning about actions, and diagnosis.

2.1 The Theory of NP Completeness.

The point of Section 1.1 is that an algorithm with polynomial running time can always solve problems even when the input is large, while an algorithm that takes exponential time cannot solve instances over a certain size. Algorithms whose running time is polynomial in the size of the input are considered "efficient" when compared to algorithms that require exponential time.

This is a rough classification of algorithms. About the intrinsic hardness of problems, we noted that, given a problem, there may be several algorithms to solve it: we are interested in finding the most efficient algorithm. Given a problem, the goal is to find a polynomial algorithm to solve it. However, this is not always possible. There are problems for which it has been proved that no polynomial algorithm can solve them: these problems are intrinsically harder than problem for which a polynomial algorithm exists. We classify the problems on the basis of this property. Note that this is a classification of problems, and not a classification of algorithms.

tractable a problem is tractable if there exists a polynomial algorithm that solves it

intractable a problem is intractable if no polynomial algorithm can solve it

Given a problem, the first step is to determine whether it is tractable or not, and if it is discovered to be tractable, to find algorithms whose running time is as low as possible (bounded by a polynomial with the lowest possible degree). Proving that a problem is tractable is easier, since it amounts to find a polynomial algorithm. On the other side, proving that *there is not* a polynomial algorithm is much more difficult.

The classification of tractable/intractable problems has been introduced at the end of the sixty. By 1970 many problems were proved to be tractable (just because someone found polynomial algorithms to solve them), but there were lot of problems for which no one was able to find a polynomial algorithm, but no one was also able to prove that such algorithms do not exist.

This situation ended with the definition of the theory of NP completeness. The idea was the following: we have several problems for which no one has found a polynomial algorithm so far. Let C be the set of this problems. Suppose we can prove a statement like:

> If the problem A is tractable (there exists a polynomial algorithm for it) then all the problems in C are tractable.

If the class C contains several problems, and no one has found a polynomial algorithm for any of them, the fact that all the problems in C are tractable looks unlikely. Thus, if we can prove the statement above for the problem A, this is very good evidence that the problem A is intractable.

Let P be the class of polynomial problems. The time of execution of an algorithm may depend on the model of computation used. It is customary to define P as the set of problems that can be solved in polynomial time on a deterministic Turing machine. This is not restrictive, since it can be proved that this definition does not change if for example we use a RAM or another deterministic computational model.

Things change if we consider non-deterministic machines. Let us first formally define a problem. Essentially, a problem is a function: given an input, find the output. In many cases, the output is simply a bit, that is, 0 or 1. The answer can be interpreted as "yes" or "no", and the problem could be to find if the input has a given property (e.g. if there is path between two nodes in a graph). According to the form of the output, we classify the problems as:

decision problems: output is (or can be represented as) a bit;

finding problems: output is a string.

The inputs and outputs of problems can be represented in many ways. In computational complexity it is customary to represent them as strings. As a

result, a finding problem is a function from strings to strings (that is, a function that takes a string as input and gives a string representing the output), and a decision problem is a function from strings to the set $\{0, 1\}$. It is also useful to consider a decision problem as the set of the input strings for which the output is 1. This may be confusing, but the idea is simple: a problem A is a set of string, such that $x \in A$ means that the output of x should be 1 (and vice versa, $x \notin A$ if the solution is 0).

Definition 3 *A decision problem A is a set of strings.*

Consider for example the problem of existence of a path beween two nodes of a graph. The instance of the problem is composed by two nodes and a graph. The problem is represented as the set of possible inputs that satisfy the given property, i.e. is the set of all the strings that represent a pair of nodes n and m, and a graph G such that there exists a path from n to m in the graph G.

A Turing machine, given an input, executes a sequence of moves. Given the state of the machine and the state of the tape, the next move to do is uniquely determined. For this reason this is called *deterministic* Turing machine. A *non-deterministic* Turing machine is different in that given the internal and tape state, there are more than one possible moves. Since the possible moves are more than one, one may ask which one will be actually done. The quite unintuitive answer is that all the possible moves are executed. Note that the non-deterministic Turing machine is a theoretical model. No actual machine can have, for example, an infinite tape. And no actual machine can do two transitions at the same time. The infinity of tape simulates the (limited) memory of actual computers, while the non-deterministic moves can be seen as an idealization (quite far from reality) of a parallel computation: when there are two possible moves, this can be seen as a processor that "spawns" a child process on another processor. The difference between the model and the practice is huge, but we think that the multi-processor machine is a good model to understand the non-deterministic machine.

A deterministic Turing machine always terminates with "yes" or "no". A non-deterministic Turing machine executes more than one computation on a given input. This can be visually understood by thinking the computation of a deterministic machine as a chain of states: each move is a link of the chain, and it makes the machine to move from one state to the next one of the chain. The diagram of states of a non-deterministic Turing machine is a tree: for each state, there may be more than one next state. As a result, each branch of the tree is a sequence of steps, and it can end with "yes" or "no". The output of the machine on a given input is "yes" if there exists at least a sequence ending with "yes".

The class P is defined as the set of problems for which there exists an algorithm running on a Turing machine that ends after a polynomial number of steps. The class NP (N standing for non-deterministic) is the set of problems for which there exist non-deterministic Turing machines that solve them, and such that each computation (i.e. each branch of the tree) has polynomial length. If we consider the non-deterministic Turing machine as a parallel computer, this is equivalent to say that the parallel machine ends after a polynomial number of steps (the problem with this idea is that non-deterministic Turing machines can always fork a new branch of computation, while parallel machines only have a limited number of processors).

How can this unintuitive class NP help in proving that some problems are intractable? Many problems, for which no polynomial algorithm exist, have been discovered to belong to this class. Let us assume, as we did at the beginning of this section, that we have proved the statement "if the problem A is tractable, then all the problems in NP are tractable." Since there are a lot of problems in NP for which no one has proved their tractability, it is unlikely that A is tractable. As a result, proving this statement is a good hint that the problem is intractable.

This idea can be formalized by using the concept of reduction.

Definition 4 *A polynomial (many-one) reduction from the problem A to the problem B is a function f from strings to strings such that*

$$x \in A \quad \textit{iff} \quad f(x) \in B$$

and such that $f(x)$ can be determined in time bounded by a polynomial in $||x||$.

If there exists a polynomial reduction between A and B we say that A can be polynomially reduced to B (written $A \leq^{\mathrm{P}} B$).

If the function f is polynomial, each instance of A can be solved by reducing it to B. For example, if we want to check whether $x \in A$, we can compute $f(x)$ in polynomial time, and then check whether $f(x) \in B$. As a result, the problem A is easier than B, in the sense that if we want to solve A we can do it by solving B. Thus, if B can be solved in polynomial time, then A can also. For this reason, we write $A \leq^{\mathrm{P}} B$. If this is the case, f is said a (polynomial) reduction from A to B, and A is *polynomially reducible* to B.

The definition of NP hardness is the basis for the proof of intractability of problems.

Definition 5 *If, for each problem $A \in$ NP it holds $A \leq^{\mathrm{P}} B$, then B is called an* NP *hard problem.*

Definition 6 *If B is a member of* NP*, and is* NP *hard, then it is called* NP *complete.*

If B is NP hard, and B can be solved in polynomial time, then all the problems in NP are solvable in polynomial time. As said above, since NP contains lot of problems for which no one have so far proved their polynomiality, this is considered unlikely. As a result, NP complete problems are considered intractable. Nevertheless, the fact that no one has proved that a set of problems are tractable does not mean that they are. For this reason, NP hard problems are called *probably* intractable, as opposite of problems that are *provably* intractable (i.e. problem for which there exists a formal proof of their intractability.)

The first problem that was proven to be NP complete is sat: given a propositional formula, decide if it is satisfiable (the definition of propositional calculus will be given in the sequel.) The paper of Karp [Kar72] proved 21 open problems to be NP complete. Note that if one of these problems were proved to be tractable, the others would also be tractable. When the book of Garey and Johnson [GJ79] was written, about 300 problems were known to be NP complete, and this number has increased since then.

The usual way to prove NP hardness of a problem C is by proving that $B \leq^{\mathrm{P}} C$, where B is a previously proved NP complete problem: indeed, since $\leq^{\mathrm{P}}$ is transitive, this implies the NP hardness of C.

The problem sat is the prototypical NP complete problem. It can be expressed as: is there an assignment such that a propositional formula is satisfied? In first-order logics this corresponds to the validity of the formula $\exists X.F(X)$, that is, given a string representing F, we want to check whether the above formula is valid. This problem can be also used for proving that a problem is in NP. Indeed, if we can polynomially reduce a problem A to sat, then A is in NP (and if we can do the converse, then A is NP complete).

A method for proving that a problem is in NP directly follows from this observation. Suppose that we can express the problem of whether $x \in A$ as a polynomial function f form pairs of strings to boolean, such that $x \in A$ holds if and only if there exists a string y such that $f(x, y)$ is true. It can then be proved that A is in NP: indeed, this is a reduction from A to sat. As a result, a proof of NP membership of A can be given as follows: we provide a function f such that, given x, if we can guess the string y then $x \in A$ can be determined in polynomial time by checking $f(x, y)$. If this is possible, then A is in NP.

2.2 The Polynomial Hierarchy

As said in the previous section, NP complete problems are considered to be harder than polynomial ones, in the sense that it is strongly believed that they

are intractable. However, many problems are NP hard but not NP complete, that is, someone proved them to be NP hard, but no one have ever proved them to *belong* to the class NP.

Such problems are considered even harder than NP problems. In order to classify the hardness of these problems, a hierarchy of classes is defined. The class P is the lowest class in the hierarchy, and all the other classes include it. The class NP is another class of the hierarchy, and it includes P. The class coNP is defined as the problems A such that the complementary problem of A (that is, the problem whose solution on a string x is "yes" if and only if the solution of A on x is "no") is an NP problem. This class is probably not equal to P nor to NP. They are usually represented with a diagram in which an arrow indicates inclusion:

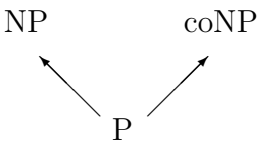


Figure 2.1: Relationships between three classes

The class P is contained in both NP and coNP. It is believed that each of these three classes is different from the other two ones. There are problems that are in NP but not in coNP, and vice versa.

There are problems that are still too hard to be captured by these classes. For this reason the polynomial hierarchy has been introduced. This hierarchy is defined in terms of oracles and (Turing) machines with oracles. An oracle is a device that can solve a specific problem with no extra time. An NP oracle is a device capable to solve an NP problem without spending time on it. An oracle Turing machine is a machine with access to an oracle. Let us define two classes in terms of oracles.

The class Δ_2^p is the class of problems that can be solved in polynomial time by a machine enhanced with an NP oracle. Essentially, the problem must be solved in polynomial time, but the machine can always take advantage of the oracle to solve subproblems at zero cost. Clearly, NP is a subset of Δ_2^p, since each problem in NP can be solved by the oracle-enhanced machine in constant time using the oracle once.

The class Σ_2^p is similar, but is the set of problems that can be solved in polynomial time by a non-deterministic Turing machine powered by an NP oracle. The relationship between the classes introduced so far is illustrated in Figure 2.2.

The class Π_2^p is defined as the set of problems A whose complementary

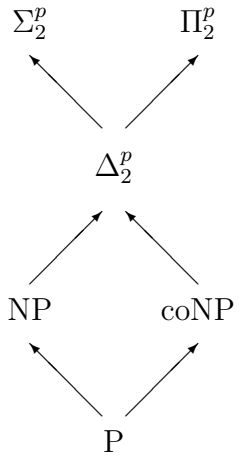


Figure 2.2: Relationships between classes

problem (the problem whose solution on x is "yes" if and only if the solution of A is "no") is in Σ_2^p. If C is a class of problems, the standard notation for the opposite problems is co-C. Thus, Π_2^p is co-Σ_2^p.

The polynomial hierarchy includes the classes named Σ_i^p, Π_i^p, and Δ_i^p, for each positive integer i. The classes Σ_0^p, Π_0^p, and Δ_0^p are P, while Σ_1^p and Π_1^p are NP and coNP, respectively. The relationship between the classes is presented in Figure 2.2. The classes with $i > 2$ are related in a similar manner (Δ_3^p is over both Σ_2^p and Π_2^p, while Σ_3^p is over Δ_3^p and so on). The polynomial hierarchy is defined as the union of all these classes:

$$\mathrm{PH} = \bigcup_{i \geq 0} \Sigma_i^p$$

We end this section by giving the prototypical complete problem for each class. Let F be a propositional formula. The problem $\forall\exists$QBF is defined as: check the validity of a formula in the form $\forall X \exists Y.F(X, Y)$. This is the prototypical Π_2^p complete problem. Similarly, the problem $\exists\forall$QBF is the problem of checking the validity of the formula $\exists X \forall Y.F(X, Y)$, and is the Σ_2^p complete problem.

Given a generic class Π_i^p, its complete problem is to establish whether $\forall X_1 \exists X_2 \ldots X_i.F(X_1, \ldots, X_i)$ is valid. For Σ_i^p the formula must start with an existential quantifier. Note that what makes Π_i^p and Σ_i^p complete a problem is the fact that there are $i - 1$ different non-consecutive quantifiers.

2.3 AI: Belief Revision

During the last years, many formalisms have been proposed in the AI literature to model common-sense reasoning. Particular emphasis has been put in the formal modeling of a distinct feature of common-sense reasoning, that is, its dynamic nature. The AI goal of providing a logic model of human agents' capability of reasoning in the presence of incomplete and changing information has proven to be a very hard one. Nevertheless, many important formalisms have been put forward in the literature.

Given a knowledge base T, representing our knowledge of the "state of affairs" of the world of interest, it is possible that we are lead to trust another piece of information P, possibly inconsistent with the old one T. The aim of revision operators is to incorporate the new formula P into the old one while preserving consistency and, at the same time, avoiding the loss of too much information in this process. This process has been called *belief revision* and the result of revising T with P is denoted as $T * P$.

This "minimal change assumption" was followed by the introduction of a large set of specific revision operators. Among the others, we mention Fagin, Ullman and Vardi [FUV83] and Ginsberg [Gin86], Satoh [Sat88] and Dalal [Dal88]. A general framework for belief revision has been proposed by Alchourron, Gärdenfors and Makinson [AGM85, Gär88]. A close variant of revision is *update*. The general framework for update has been studied by Katsuno and Mendelzon [KM89, KM91] and specific operators have been proposed, among the others, by Winslett [Win90] and Forbus [For89].

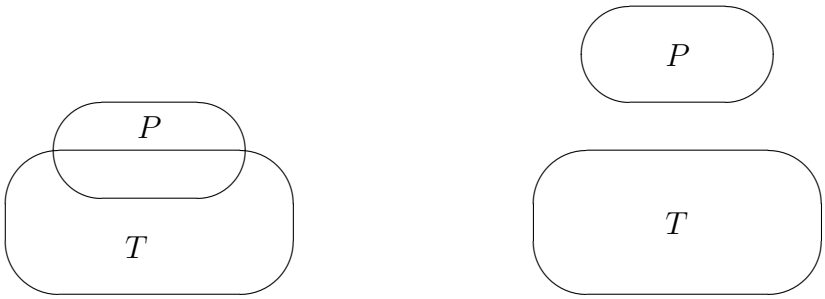
We consider the knowlegde bases T and P to be propositional formulas, over an alphabet of symbol X and with the usual connectives $\vee$, $\wedge$, and $\neg$. The symbol $\rightarrow$ is a shorthand, and $F_1 \rightarrow F_2$ stands for $\neg F_1 \vee F_2$. The symbols $\equiv$ and $\not\equiv$ are also shorthands: $F_1 \equiv F_2$ stands for $(F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$, while $F_1 \not\equiv F_2$ stands for $\neg(F_1 \equiv F_2)$.

An *interpretation* is a truth assignment to the atoms of the alphabet. We denote interpretations as sets of atoms (those assigned true). A *model* M of a formula F is an interpretation that satisfies F (written $M \models F$). A theory is a set of formulae. An interpretation is a model of a theory if it is a model of every formula of the theory. Given a theory T and a formula F we say that T entails F, written $T \models F$, if F is true in every model of T. Given a propositional formula or a theory T, we denote with $Mod(T)$ the set of its models. We say that T is consistent, written $T \not\models \bot$, if $Mod(T)$ is non-empty.

2.3.1 AGM Revision

Consider the revision of a knowledge base T with a formula P, which may be inconsistent with T. The scenario can be better understood considering the

sets of models of T and P. Two propositional formulas T and P are consistent if and only if their sets of models have a non-empty intersection, as shown in Figure 2.3.



a. consistent revision b. inconsistent revision

Figure 2.3: Consistent and inconsistent revisions.

In the case a., when T and P are consistent, the choice is simple: take $T \wedge P$, that is consistent, to be the result of the update. This is equivalent to say that the set of models of the resulting knowledge base is the intersection of those of T and P, or

$$Mod(T * P) = Mod(T) \cap Mod(P)$$

The definition of revision becomes non-trivial when T and P are not consistent together. Alchourrón, Gärdenfors, and Makinson defined some guiding principles for revision. Namely, they defined a set of eight postulates that any revision operator should satisfy. These postulates have been reformulated by Katsuno and Mendelzon as follows:

1. $T * P \models P$
2. If $T \wedge P$ are consistent, then $T * P \equiv T \wedge P$
3. If P is satisfiable, then $T * P$ is satisfiable
4. If $T_1 \equiv T_2$ and $P_1 \equiv P_2$ then $T_1 * P_1 \equiv T_2 * P_2$
5. $(T * P) \wedge Q \models T * (P \wedge Q)$
6. If $(T * P) \wedge Q$ is satisfiable, then $T * (P \wedge Q) \models (T * P) \wedge Q$

Each postulate is motivated by some reasonable assumption. For instance, the fact that $T * P \models P$ is motivated by the fact that, if P has to be inserted into T, this means that the formula P is believed to be true. Thus, P must be true in the knowledge base after the revision.

Katsuno and Mendelzon [KM91] give a very elegant manner to express these postulates in a semantical way. Their formalization is based on faithful orderings. Assume that, for each T, a linear ordering on the interpretations is defined. The intuitive meaning for $\leq_T$ is that $I \leq_T J$, where I and J are interpretations, if the interpretation I is considered *more plausible* than J, if the knowledge base is T. We also assume the following condition

Definition 7 *The ordering $\leq_T$ is faithful if and only if:*

1. *$I \leq_T J$ for each J if and only if $I \in Mod(T)$;*

2. *$T_1 \equiv T_2$ implies that $\leq_{T_1}$ is identical to $\leq_{T_2}$.*

Let us explain the meaning of these orderings with an example. The ordering $<_T$ is defined by $I <_T J$ if and only if $I \leq_T J$ and not $J \leq_T I$. Assume that P has only two models I and J, and it holds $I <_T J$. Each interpretation is a complete description of the world, or, using a frequently used term, is a possible world. If P has two models I and J, then saying that P is true is equivalent to say that we know that only two possible scenarios are consistent with our knowledge of the world, namely the one represented by I, and the one represented by J.

Since the scenario (represented by) I is more plausible than the scenario (represented by) J, the revised knowledge base contains only the model I. This is perfectly coherent with the principle of minimal change: since we must modify our beliefs of the world as little as possible, we choose the most plausible model of P.

The condition of faithfulness of the orderings reflect the fact that the models of T are the most plausible models, if we believe true the formula T. In other words, T is only a two-set classification of the models: the models of $Mod(T)$ are considered plausible while the models of $Mod(\neg T)$ are not. However, not all the models of $Mod(\neg T)$ are in general considered equally (un)plausible. The ordering $\leq_T$ gives a classification of plausibility of the models of $Mod(\neg T)$. The fact that the models of T are more plausible of those of $\neg T$ is reflected by the condition of faithfulness.

The general procedure of revision is very simple: given a knowledge base T and a revision P, the revised base is defined as:

$$Mod(T * P) = \min(Mod(P), \leq_T) \tag{2.1}$$

Indeed, since the possible scenario after the update must be models of P, and $\leq_T$ represents an order of plausibility between the interpretations, the result of the revision is composed by the most plausible models of P, thus the minimal models of P w.r.t. the ordering $\leq_T$.

This definition is correct and complete w.r.t. the revision postulates, in the sense that each revision defined as in (2.1) satisfies the AGM postulates, and each revision satisfying the AGM postulates can be expressed as in (2.1) by giving an appropriate plausibility ordering $\leq_T$ for each base T.

Another way of formulating the AGM revision is the following. A ranking κ is a function from interpretations to non-negative integers. Assume that each formula T is associated to a ranking κ_T, with the following properties:

1. $\kappa_T(I) = 0$ if and only if $I \in Mod(T)$;
2. If $T_1 \equiv T_2$ then $\kappa_{T_1} = \kappa_{T_2}$.

Such a function can represent a plausibility ordering, and $\kappa_T(I)$ represents the degree of plausibility associated to the interpretation I in the knowledge base T. Thus, it must be $\kappa_T(I) \leq \kappa_T(J)$ if and only if $I \leq_T J$, where $\leq$ is the usual ordering between integers. It is possible to prove that each AGM revision can be expressed as

$$\begin{aligned} Mod(T * P) &= \min(Mod(P), \kappa_T) \\ &= \{I \in Mod(P) \mid \nexists J \in Mod(P) \,.\, \kappa_T(J) < \kappa_T(I)\} \end{aligned} \quad (2.2)$$

using a suitable ranking κ_T for each T. The converse also holds: given a ranking κ_T for each formula T that respects the above properties, the revision defined by (2.2) satisfies all the AGM postulates.

This way of formalizing the AGM revision is mainly used in the formalization of iterated revision.

2.3.2 One-step Revision

Several specific revision operators have been defined in the literature. We report here the definitions of the operators used in the sequel.

As said above, the basic principle of belief revision is that of minimal change. The approaches we are going to introduce are some different formalizations of this principle. Let I and J be two interpretations. Each of them represents a possible scenario. If $I = J$, these two scenarios are identical. Thus, if I is a possible situation before the revision and J is a possible situation according to the revising formula P, then there is no need of change in one's mind to achieve the revision: indeed, there is no need of revision, since

the scenario after the revision is identical to the previous one. On the other hand, if I assigns true to the false variables of J (and vice versa), we can say that the change was big, since each single (atomic) believed fact has changed its status from true to false and vice versa.

Using this basic idea, $I = J$ are the minimal possible change while $I \cap J = \emptyset$ is the maximal one, Dalal [Dal88] and Satoh [Sat88] defined two ways of measuring the change required for a revision in terms of the difference between the models of T and P. Let

$$Diff(I, J) = I \backslash J \cup J \backslash I$$

In other words, $Diff(I, J)$ is the set of atoms of the alphabet that are assigned different truth value by I and J. The principle behind Dalal's and Satoh's revisions are: the greater is the set $Diff(I, J)$, the greater amount of change is required. Thus, the result of the revision is the set of models of P that are closest to those of T. Satoh's revision uses the set containment relation $\subseteq$ as a measure of closeness:

Definition 8 *Satoh's revision is defined by*

$$\begin{aligned} Mod(T *_S P) \quad = \quad & \{\ J \in Mod(P) \mid \exists I \in Mod(T) \textit{ such that} \\ & \nexists I' \in Mod(T),\ J' \in Mod(P)\ .\ Diff(I', J') \subset Diff(I, J)\ \} \end{aligned}$$

Satoh's revision can be reformulated as follows: let

$$\begin{aligned} K_{T,P} \quad = \quad & \{Diff(I, J) \mid I \in Mod(T), J \in Mod(P) \text{ and} \\ & \nexists I' \in Mod(T), J' \in Mod(P) \text{ such that } Diff(I', J') \subset Diff(I, J)\} \end{aligned}$$

then $Mod(T *_S P)$ can be rewritten as

$$Mod(T *_S P) = \{J \in Mod(P) \mid \exists I \in Mod(T) \text{ such that } Diff(I, J) \in K_{T,P}\}$$

On the converse, Dalal's revision uses the *number* of atoms in $Diff(I, J)$ to formalize the distance between the interpretations I and J.

Definition 9 *Dalal's revision is defined by*

$$\begin{aligned} Mod(T *_D P) \quad = \quad & \{\ J \in Mod(P) \mid \exists I \in Mod(T) \textit{ such that} \\ & \nexists I' \in Mod(T), J' \in Mod(P).\ |Diff(I', J')| \leq |Diff(I, J)|\} \end{aligned}$$

Dalal's revision can be reformulated as follows. Let $k_{T,P}$ be the minimum value of $Diff(I, J)$ for $I \in Mod(T)$ and $J \in Mod(P)$. It holds

$$Mod(T *_D P) = \{J \in Mod(P) \mid \exists I \in Mod(T) \text{ such that } |Diff(I, J)| = k_{T,P}\}$$

It has been proved that Dalal's revision satisfies all the AGM postulates, while Satoh's does not. Winslett's revision is similar to Satoh's, except that any model of T is "separately" revised.

Definition 10 *Winslett's revision is defined by*

$$\begin{aligned} Mod(T *_W P) &= \{J \in Mod(P) \mid \exists I \in Mod(T) \text{ such that} \\ &\quad \nexists J' \in Mod(P) \,.\, Diff(I, J') \subset Diff(I, J)\} \end{aligned}$$

One can prove that

$$Mod(T *_W P) = \bigcup_{I \in Mod(T)} Mod(Form(I) *_S P)$$

Forbus' definition uses a similar criterion with the cardinality based distance.

Definition 11 *Forbus' revision is defined by*

$$\begin{aligned} Mod(T *_F P) &= \{J \in Mod(P) \mid \exists I \in Mod(T) \text{ such that} \\ &\quad \nexists J' \in Mod(P) \,.\, |Diff(I, J')| \subset |Diff(I, J)|\} \end{aligned}$$

Forbus' revision relates to Dalal's in the following manner.

$$Mod(T *_F P) = \bigcup_{I \in Mod(T)} Mod(Form(I) *_D P)$$

Weber's definition uses the set $K_{T,P}$ defined above.

Definition 12 *Weber's revision is defined by*

$$Mod(T *_{Web} P) = \{J \in Mod(P) \mid \exists I \in Mod(T) \text{ such that } I \backslash \Omega = J \backslash \Omega\}$$

where $\Omega = \cup K_{T,P}$.

The revision operators defined by Ginsberg and Nebel, as well as the WIDTIO revision, are defined over a pair (T, P) where T is a theory (a set of propositional formulas) rather than a single formula. In the following definitions, and in any other place in which those revision operators appear, we implicitly assume that T is a set of formulas: this is called a theory. As a result, T will be used for representing both a formula (when we talk of Dalal's etc. revisions) and a theory (when we show results about Ginsberg's or Nebel's revisions).

Let $W(T, P)$ the set of the largest subsets of T that are consistent with P, that is,

$$W(T, P) = \max(\{T' \subseteq T \mid T' \cup \{P\} \text{ is consistent } \}, \subseteq)$$

Definition 13 *Ginberg's revision is defined as:*

$$T *_G P = P \wedge \bigvee W(T, P)$$

Nebel's revision extends Ginsberg's revision by adding priority to formulas in T. Since Ginsberg's revision is a specific case of Nebel's one, all the negative results (incompilability, etc.) that hold for the former hold also for the latter.

The WIDTIO revision has a more simplified definition.

Definition 14 *The WIDTIO revision is defined as*

$$T *_{Wit} P = P \wedge \bigwedge \cap W(T, P)$$

2.3.3 Iterated Revision

The AGM postulates and their Katsuno-Mendelzon semantical characterization (2.1) are the main results about the one-step revision. They express how to revise a base T with respect to a single revision P, but they are not intended to formalize the iteration of this process. In other words, they express what $T * P_1$ should be, but they do not say how to revise the resulting base w.r.t. a subsequent revision P_2. One may define the result of the second revision simply as $(T * P_1) * P_2$, but this might lead to counter-intuitive results. The problem is that κ_T represents some meta-information on how to behave when a revision occurs, that is, something that may not depend only on the formula T. On the other hand, according to AGM postulates, κ_{T*P_1} depends only on $T * P_1$ (which is a formula with no extra information associated), and it is not related on the plausibility ordering before the revision κ_T. Intuitively, the plausibility after one revision should instead be related to the previous ordering of plausibility.

We give the definition of some revision operators based on the principle that the plausibility ordering after the revision should be obtained from the ordering before the revision. This allows the definition of some intuitive ways of doing sequences of revisions.

We need a consistent way to denote the orderings of plausibility. In the AGM framework, each possible formula T is associated to a ranking κ_T. We maintain this convention. However, κ_T is used only to define the *first* revision, that is, $Mod(T * P_1) = \min(Mod(P_1), \kappa_T)$. This formula $T * P_1$ represent the beliefs about the state of the world after the first revision. In case we need to do a second revision, we need not only a set of beliefs, but also a new ranking. To denote this ranking we use the notation $\kappa_{[T;P_1]}$. Note that this may be different (and indeed it will be, in almost all cases) from κ_{T*P}. If a new revision P_2 occurs, the result of this new revision is determined by this new ordering $\kappa_{[T;P_1]}$. As a result, the new belief base is

$$\min(Mod(P_2), \kappa_{[T;P_1]})$$

A careful reader may note that this is not a definition, since we have not written a name for this formula. The most intuitive way to denote the result

of the second revision is $T * P_1 * P_2$ or, still better, $(T * P_1) * P_2$. However, this is formally incorrect. Indeed, $T * P_1$ is a formula, and its associated ranking is κ_{T*P_1} which can be different from $\kappa_{[T;P_1]}$. Indeed, $(T * P_1) * P_2$ is already defined, and is different from the result of the second revision.

In general, if we revise a base T with a sequence of revisions $[P_1, \ldots, P_m]$, we denote the formula representing the beliefs after the sequence of revisions by

$$T * [P_1, \ldots, P_m]$$

and the associated ranking by

$$\kappa_{[T;P_1,\ldots,P_m]}$$

A graphical representation of the process of iterated revision is reported in Figure 2.4.

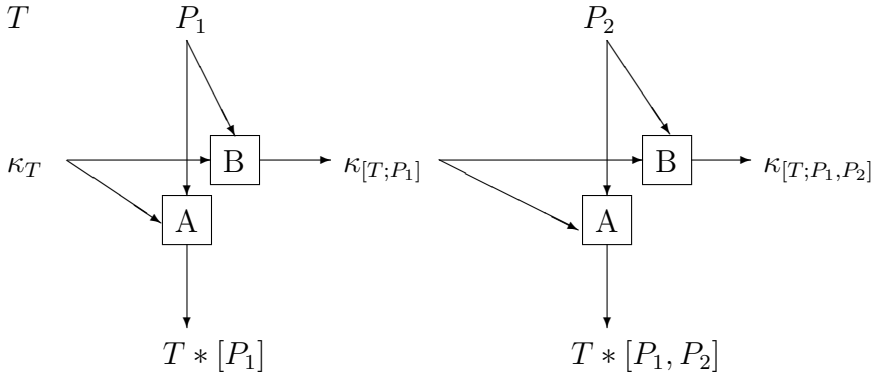


Figure 2.4: The process of iterated revision.

The process goes as follows: from P_1 and κ_T we derive $T * [P_1]$. This is represented by the block A, that represents the rule $Mod(T * [P_1]) = \min(Mod(P_1), \kappa_T)$. From P_1 and κ_T we also obtain the new plausibility ordering $\kappa_{[T;P_1]}$ (block B). When another revision P_2 occurs, we are able to calculate $T * [P_1, P_2]$ from $\kappa_{[T;P_1]}$ and P_2 using the same procedure A. The only point of this procedure not yet defined is the block B, that is, how to determine $\kappa_{[T;P_1]}$ from κ_T and P_1. The iterated revisions proposed in the literature differ mainly for this point. Note that T is not used in the process of revision: indeed, it is only used to determine κ_T, and it does not directly influence the process of revision.

In general, the block A represents the computation

$$Mod(T * [P_1, \ldots, P_m]) = \min(Mod(P_m), \kappa_{[T;P_1,\ldots,P_{m-1}]})$$

Let us now define the block B. The first definition has been given by Spohn. We are given a ranking $\kappa_{[T;P_1,\ldots,P_{m-1}]}$ and a formula P_m, and we wish to have a new ranking $\kappa_{[T;P_1,\ldots,P_m]}$. Since the number of interpretations is finite, we can represent the ranking as a partition of the models, as in Figure 2.5. The models in the lower classes are those considered minimal

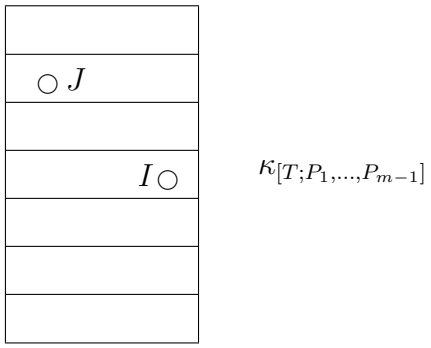


Figure 2.5: A representation of an ordering.

w.r.t. the ordering, thus the most plausible ones. For example in the figure $\kappa_{[T;P_1,\ldots,P_{m-1}]}(I) < \kappa_{[T;P_1,\ldots,P_{m-1}]}J$.

Definition 15 (Prioritized revision) *The plausibility ordering after a revision P_m is defined as*

$$\kappa_{[T;P_1,\ldots,P_m]}(I) = \begin{cases} \kappa_{[T;P_1,\ldots,P_{m-1}]}(I) & \text{if } I \in Mod(P_m) \\ \kappa_{[T;P_1,\ldots,P_{m-1}]}(I) + r & \text{otherwise} \end{cases}$$

where r is the maximal value of $\kappa_{[T;P_1,\ldots,P_{m-1}]}$ over all *the interpretations.*

This definition is much simpler than it looks. Let us consider an example. Figure 2.6 represents the revision of the ordering w.r.t. the formula P_m.

The result of Spohn's revision is that all the models of P_m are moved under all the other models. On the figure, the box representing the models of P_m is taken as a whole and moved under the other models, as shown in Figure 2.7.

The next definition is due to Boutilier [Bou93].

Definition 16 (Natural revision) *The plausibility ordering after a revision P_m is defined as*

$$\kappa_{[T;P_1,\ldots,P_m]}(I) = \begin{cases} 0 & \text{if } I \in \min(Mod(P_m), \kappa_{[T;P_1,\ldots,P_{m-1}]}) \\ \kappa_{[T;P_1,\ldots,P_{m-1}]}(I) + 1 & \text{otherwise} \end{cases}$$

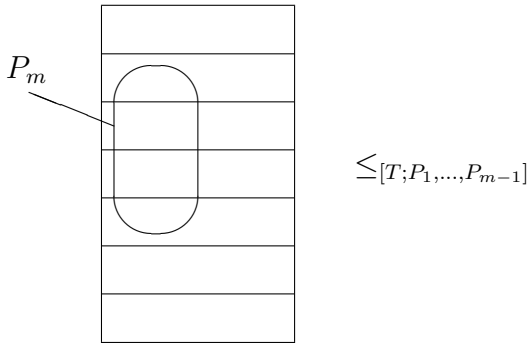


Figure 2.6: Ordering and revision

A graphical representation of how the ordering is transformed is given in Figure 2.8. The models of $\min(Mod(P_m), \kappa_{[T;P_1,\ldots,P_{m-1}]})$ are moved below all the other models. The other models of P_m are not moved, in contrast of what happens in Spohn's revision.

There are other definitions, besides Spohn's and Boutilier's ones. We do not report these definition here; they are given in the papers where they are introduced: Williams' adjustment and conditionalization are defined in [Wil94], Lehman's iterated revision is defined in [Leh95]. The computational complexity of these revisions is studied in [Lib97a]. In this dissertation we report some results about the compilability of iterated revisions.

2.4 Circumscription

Circumscription has been introduced to formalize an important aspect of common-sense reasoning: the closed world assumption. In every-day reasoning, humans often make assumption not logically justified from their knowledge. However, such conclusions are intuitively correct. For example, if we see a gray building in front of our office window every day, it is natural to assume that the building will be there tomorrow, and that it will be still gray. However, there is no rule that says that if a thing has a color, then the color remains the same forever.

Closed world assumption is a formalization of common-sense reasoning. Under this assumption, facts that are not known are assumed to be false. In the example of the color, if we do not know anything about the fact "the building has been painted", we assume this fact to be false. In this case, the formalization is simple: if T is our knowledge base and c is the atom representing the change of color, our common-sense conclusion is that $T \wedge \neg c$

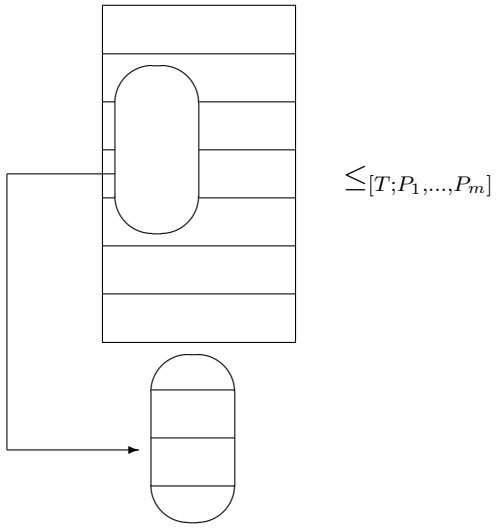


Figure 2.7: Result of Spohn's revision

is true (we can use this enhanced knowledge base to draw some conclusions that are not derivable from T alone.)

The formalization is more complex when there are two or more atoms (facts) that are false "by default". Let us consider for example the knowledge base $T = x_1 \vee x_2$. The fact x_1 is not derivable from T, thus we could conclude that it is false. The same for x_2, thus both x_1 and x_2 should be false. However, $T \wedge \neg x_1 \wedge \neg x_2$ is inconsistent. We cannot make false both variables at the same time without getting a contradiction.

There are many formalisms that solve this problem. Circumscription is one of the best known. What is done in circumscription is to try to make false as many variables as possible without obtaining an inconsistent knowledge base. This leads to many possible choices. In the example above, we can make false the variable x_1, obtaining $T \wedge \neg x_1$. If we make this choice, we have $T \wedge \neg x_1 \models x_2$: since x_2 is implied by the knowledge base, we cannot assume that it is false. The other choice is to start by assuming x_2 false. In this case, $T \wedge \neg x_2 \models x_1$ and thus we cannot assume x_1 to be false. As a result, this "maximization of false atoms" leads to two possible knowledge bases:

$$\begin{array}{rcl} & +\neg x_1 \;\Rightarrow\; T_1 \;=\; T \wedge x_1 \;=\; \neg x_1 \wedge x_2 \\ T \;\Rightarrow & & \\ & +\neg x_2 \;\Rightarrow\; T_2 \;=\; T \wedge x_2 \;=\; x_1 \wedge \neg x_2 \end{array}$$

T_1 and T_2 are two possible knowledge bases. Since there is no way to choose between them, we assume that any of these possibilities may be the

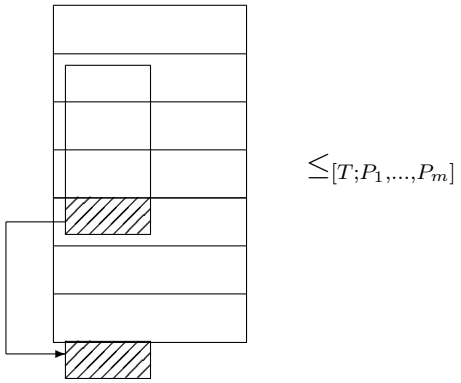


Figure 2.8: Result of Boutilier's revision

real set of facts. Thus, a fact is implied by T under circumscription if and only if both of these bases imply the fact.

Let us now define circumscription formally. Let us consider the set of models of T. The base T is consistent with x_i if and only if there exists a model of T that has x_i false. As a result, when we add a maximal quantity of false atoms we obtain a model with the maximal quantity of false atoms. Any model M is represented as the set of true atoms, thus what we obtain is a minimal model of T. Each of these minimal models is a way to maximize the number of false atoms. Since we want to draw a conclusion if and only if it is true in all these "false-maximized" knowledge bases, what we obtain is that a conclusion is true if and only if its sets of models contains all the minimal models of T. This is equivalent to say that the knowledge base with the circumscription assumption is equivalent to the formula that contains exactly the minimal models of the original knowledge base.

In many cases, not all the variables can be assumed false when unknown. Let X be the set of such variables. In such cases, the comparison between models must be made only for the variables in X, that is, only the variables in X must be minimized.

Definition 17 *The circumscription of a propositional formula T is a formula whose set of models is*

$$\begin{aligned} Mod(CIRC(T;X,Y,Z)) \quad = \quad & \{M \in Mod(T) \mid \nexists M' \in Mod(T) \ . \\ & M' \cap Y = M \cap Y \text{ and } M' \cap X \subset M \cap X\} \end{aligned}$$

The circumscription has been defined by McCarthy in [McC80] for first order theories. The definition presented here is the propositional case version. In [EG93], the problem of deciding whether $CIRC(T;X,Y,Z) \models Q$ is proved

to be Π_2^p complete, while deciding $M \models CIRC(T; X, Y, Z)$ is proved coNP complete in [Cad92].

2.5 Resoning about Actions

One of the main topics in knowledge representation is the description of properties of actions and their effects [Bak91, LS95b, MH69, Rei91]. This problem is also related to belief revision. One of the most successful formalisms proposed in the last few years is the language $\mathcal{A}$, introduced by Gelfond and Lifschitz [GL93]. $\mathcal{A}$ is a simple "high-level" language, composed of two kinds of statements: statements about the effects of actions, and statements about states.

As a "hello world" example of this language, we give the formalization of the well-known Yale Shooting Problem [HM87]:

$$\begin{aligned}
&\textsf{initially } \neg Loaded\\
&\textsf{initially } Alive\\
&Load \textsf{ causes } Loaded\\
&Shoot \textsf{ causes } \neg Alive \textsf{ if } Loaded\\
&Shoot \textsf{ causes } \neg Loaded
\end{aligned}$$

The first two statements mean that initially Fred is alive, and the gun is not loaded. The three other statements describe how the actions of loading and shooting the gun change the state of the world: for example Fred dies if the loaded gun is shoot at him.

A detailed description of the language $\mathcal{A}$ is given in [GL93], where more examples are also provided. For the sake of completeness, we recall here the syntax and the semantics of $\mathcal{A}$. More details, including the translation of $\mathcal{A}$ into logic programs, can be found in the above paper.

2.5.1 Syntax of $\mathcal{A}$

There are two nonempty sets of symbols, called fluent names and action names. A fluent expression is a fluent name possibly preceded by the negation symbol $\neg$. If F is a fluent expression and $A_1, \ldots, A_m$ are action names, then

$$F \textsf{ after } A_1; \ldots; A_m$$

is a value proposition. If $m = 0$, the above statement is written

$$\textsf{initially } F$$

A value proposition specifies the truth value of a fluent in the initial state, or after a sequence of actions. An effect proposition is an expression of the form

$$A \text{ causes } F \text{ if } P_1, \ldots, P_m$$

where A is an action name and $F, P_1, \ldots, P_m$ are fluent expressions. The meaning is of course that the action A, when performed, makes the fluent expression F true, if the fluents $P_1, \ldots, P_1$ are true. The fluents $P_1, \ldots, P_1$ are called preconditions, while F is the effect or postcondition of the proposition. If $m = 0$ the world "if" can be dropped, obtaining the shorter version

$$A \text{ causes } F$$

A domain description is a set of value and effect propositions.

2.5.2 Semantics of $\mathcal{A}$

The semantics of $\mathcal{A}$ is defined in terms of *states* and *models*. A state is a set of fluent names. A fluent expression F is true in the state σ if $F \in \sigma$, false otherwise. A fluent expression $\neg F$ is true in σ if and only if F is false in σ. A transition function Φ is a function from the set of pairs (A, σ), where A is an action and σ a state, to the set of states. With $\Phi(A, \sigma)$ we want to represent the state obtained performing the action A in the state σ.

Let $V_D^+(A, \sigma)$ be the set of the fluent names F (i.e. positive fluent expressions) such that there exists an effect proposition A causes F if $P_1, \ldots, P_m$ in the domain description D and $P_1, \ldots, P_m$ are true in σ. Intuitively, $V_D^+(A, \sigma)$ represents the set of fluents whose value must became true when the action A is performed in the state σ.

In a similar manner, $V_D^-(A, \sigma)$ is the set of fluents whose value must became false, and thus is defined as the set of fluent names F such that there exists an effect proposition A causes $\neg F$ if $P_1, \ldots, P_m$ in the domain description D and $P_1, \ldots, P_m$ are true in σ.

The transition function associated to a domain description D is the (partial) function Ψ_D defined as

$$\Psi_D(A, \sigma) = \begin{cases} (\sigma \cup V_D^+(A, \sigma)) \backslash V_D^-(A, \sigma) & \text{if } V_D^+(A, \sigma) \cap V_D^-(A, \sigma) = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

An interpreted structure is a pair (σ_0, Φ), where σ_0 is a state (the initial state of the system) and Φ is a transition function. Given an interpreted structure $M = (\sigma_0, \Phi)$ and a sequence of actions $A_1; \ldots; A_m$, we denote the state after the sequence as

$$\Phi(A_1; \ldots; A_m, \sigma_0) = \Phi(A_m, \Phi(A_{m-1}, \ldots, \Phi(A_1, \sigma_0) \ldots))$$

A value proposition F after $A_1; \ldots; A_m$ is true in the structure M if F is true in the state $\Phi(A_1; \ldots; A_m, \sigma_0)$, false otherwise.

An interpreted structure $M = (\sigma_0, \Phi)$ is a model of the domain D if and only if the transition function Φ is equal to Ψ_D, and is total (i.e. defined for each pair (A, σ)) and each value proposition in the domain is true in M. Different models of the same domain description differ only by their initial state.

A domain description is consistent if and only if there is a model of it. A domain description D entail a value proposition V (written $D \models V$) if and only if V is true in any model of D. The complexity of the deciding the consistency of a domain description is NP complete, while entailment is coNP complete. These results are proved in [Lib97c].

2.6 Diagnosis

Diagnosis – finding the causes of observable effects – is one of the main applications of artificial intelligence. The word "diagnosis" is quite self-explaining, but there are several formalizations in the literature. All of them share the basic idea: given a description of a complex system, and a set of facts about it, find what are the "causes" of these facts. The applications of diagnosis are countless: the system may be a physical machine, the manifestations the observable abnormal behavior of the machine, and the aim of the diagnostic process to find the faults. In medicine there are a set of symptoms, and the aim is the find the causes of them.

Here we concentrate on the definition of diagnosis given by Peng and Reggia in [PR86]. They express a problem of diagnosis as follows: given a set of faults F, a set of effects M, a function $e : F \to \mathcal{P}(M)$, a set of observable effects $M' \subseteq M$, and an integer k, decide if there exists a set $F' \subseteq F$ of size at most k, such that $\bigcup_{f \in F'} e(f) \supseteq M'$.

Short explanation: the behavior of the system is basically expressed by the function e. This function does not describe the normal behavior of the system. Rather, it expressed the effect of the faults, that is, when a set of F' of faults occurs in the system, we know that the result will be a set of manifestations $\bigcup_{f \in F'} e(f)$. Since the aim of diagnosis is to find the causes of a given set of manifestations M', we define a (minimal) diagnosis as a minimal set of faults that explain that set of manifestations. Formally, this is expressed as finding a set of faults $F' \subseteq F$ such that the effects of these faults explain all the observable manifestations M', that is $\bigcup_{f \in F'} e(f) \supseteq M'$.

This is a finding problem, while we are more interested in decision problems. Thus, we consider the variant of the problem in which the answer is yes/no. The most interesting decision problem of a diagnostic problem is to

decide the number of faults that can explain the manifestations, and thus we consider the problem of deciding whether k faults are enough. This problem is NP complete, as proved in [ATBJ87].

Chapter 3

Compilation

In this chapter we introduce the formal definition of compilability and preprocessing. The idea of preprocessing is that a given problem may be solved efficiently when part of the input can be preprocessed in advance, and we do not require this preprocessing to be very efficient. We define a hierarchy of classes of problems (similar to the polynomial hierarchy) that takes into account the possibility of decreasing the complexity via a preprocessing. We also show how these definitions can be applied to the problem of belief revision.

3.1 Formal Definition of Compilability

The definition of preprocessing (or compilation) can be better explained graphically. Let us start with the usual schema of a generic algorithm. An algorithm can be viewed as a "box" that takes as input a string, and gives as output a bit (in the case of algorithms for decision problems) or another string (in the case of finding problems). Consider the decision problem A. As already said in the introduction, we consider each decision problem as a language (set) of strings. Thus the (formal) problem A is informally described as the problem of deciding whether a string x belongs to A or not. Thus, the generic algorithm for solving A can be seen as follows.

Figure 3.1: A generic algorithm for solving A

The algorithm takes the string x as input, do some processing on it, and end up with the answer (whether x belongs to A or not). In the "classical"

complexity analysis of algorithms the only things we are interested about the box are:

1. how much time it takes to give the answer, and
2. how much space (i.e. memory) it uses.

The lower is the time/space used, the better is the algorithm. This is a natural measure of goodness of an algorithm, since we want the solution in a reasonably short amount of time, and without requiring more memory than reasonable.

However, for the sake of formalizing the efficiency when preprocessing is allowed, the old definition of efficiency does not have a sufficient degree of "granularity". The problem is that the overall time of the algorithm is not a good measure when there is a part of the input that can be preprocessed in some way. Let us consider the example of diagnosis. It is a very intuitive one, and it has many practical applications.

Consider, for example, a system of diagnosis of an airplane. There are a number of possible faults (for example: engine fault, low fuel, etc.), each represented as an element of a set of faults F. Each fault is known to cause a set of manifestations. This is represented by the function e. We denote $e(F')$, where F' is a set of faults, the set of observable manifestations that occur when the faults F' happen at the same time.

The instance of the problem is (essentially) composed by two parts: M', which is the set of manifestations that are occurring, and e, which is the behavior of the system. The behavior of the system is known when the airplane is built, while the set of manifestations is (of course) not known before it occurs. The instance of the problem, the pair $\langle e, (M', k)\rangle$, is composed by two elements with different status: the first part e is known, and we can spend more time on it, while we want a very fast answer (deciding if there exists an explanation of size k) in response of a given set of manifestations M'.

Let us compare two algorithms for this problem:

Alg1: takes time $0.00001 \cdot 2^{||e||+||M'||}$

Alg2: preprocess e in time $1000 \cdot 2^{||e||}$, and then to explain a set of manifestations M' it takes $||M'||^2$.

The first algorithm is described only in terms of the total amount of time required to solve an instance: from this point of view it is "black box", as in Figure 3.1. The description of the way the second algorithm works is more detailed: first e is analyzed alone, then the actual diagnosis is performed.

The first algorithm may seem more efficient in many cases, but let us consider the practical application of it: since the behavior of the airplane is known in advance, there is no problem in doing a very long processing

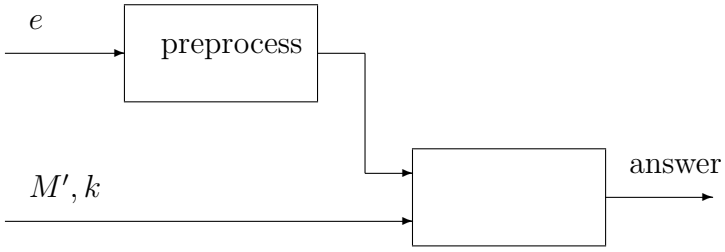


Figure 3.2: Preprocessing of part of the input

on e alone. For example, one week on a supercomputer may be very well affordable. On the converse, when a set of manifestations occur, the time scale is completely different: two seconds of delay may be too much.

The overall time required is not a good measure in such a scenario. We allow a preprocessing of part of the input, and we are not interested in the time spent in this phase. We only want to minimize the time needed for the second phase.

We consider the preprocessing from the point of view of the structural complexity: we consider tractable a problem for which there is a polynomial algorithm that solves it, and intractable a problem for which there are none. Moreover, we assume intractable an NP hard problem. In this framework, the diagnosis problem can be seen as follows: the problem is NP complete, thus intractable. However, it may be possible that, preprocessing only the behavior of the system, the computation needed on M' is only polynomial. In the schema of Figure 3.2, we are not interested in the time needed for the preprocess. We only require the "empty box" to take only a polynomial time to give the answer, given M', k, and the result of the preprocess. Thus, Alg1 in the example above is a poor algorithm since it is exponential in $||M'||$, while Alg2 is good, since the time required after the compilation of e is polynomial.

In order to formalize this idea, we give the following definitions. The problems we are interested in have two inputs: the first is the part on which we allow a preprocessing (in the case of diagnosis: the function e), and the second is the rest of the input (in the case of diagnosis: the set of the current manifestations M' and the number k). All the data (inputs and outputs) are represented as strings: let Σ be the alphabet of the strings. A problem, as said above, is represented by a set of strings, that is, $A \subseteq \Sigma^*$. Since we want a distinction between the two parts of the input, we consider a problems as a set of pairs of strings, that is $A \subseteq \Sigma^* \times \Sigma^*$. In the case of diagnosis, a pair $\langle e, (M', k)\rangle$ (where e is the first element of the pair and (M', k) is the second one) belongs to the set if and only if there is an explanation of the set of manifestations M' of size less than k. Given an instance $\langle x, y\rangle$ of a problem

of pairs A, we say that x (the part on which a preprocessing is allowed) is the *fixed* part, while y is the *varying* part.

Definition 18 *A (decision) problem of pairs is a set $A \subseteq \Sigma^* \times \Sigma^*$. An instance is an element $\langle x, y \rangle$ of $\Sigma^* \times \Sigma^*$. The first element of the pair x is said the fixed part, while the second element y is the varying part.*

In order to formalize the possibility of reducing the complexity via a preprocessing, we introduce the following class.

Definition 19 *The class of problems $\langle \text{EXPTIME}, \text{P} \rangle$ is the set of problems that can be solved in polynomial time, if an exponential-time preprocessing on the fixed part of the input is allowed.*

In other words, it must be possible to preprocess x in exponential time in such a way it makes polynomial the solving of the problem knowing y and the result of the preprocessing.

Formally, the class $\langle \text{EXPTIME}, \text{P} \rangle$ can be expressed as follows. A generic problem of pairs A belongs to $\langle \text{EXPTIME}, \text{P} \rangle$ if and only if there exist an exponential function f (a function that can be calculated in exponential time) and a polynomial language S such that $\langle x, y \rangle \in A$ if and only if $\langle f(x), y \rangle \in S$. The function f formalizes the preprocessing, and $f(x)$ is the result of this phase. The language S is the on-line computation: given $f(x)$ and y it must be possible to decide whether $\langle x, y \rangle \in A$ in polynomial time.

In the case of diagnosis, as in many other NP complete problem, this compilation is trivially possible. However, as explained in the sequel, this is not possible for many other problems from artificial intelligence.

Theorem 1 *The problem of diagnosis is in $\langle \text{EXPTIME}, \text{P} \rangle$.*

Proof. In order to prove the claim, we must show that there is an exponential function f and a polynomial language S such that, given a problem of diagnosis $\langle e, (M', k) \rangle$, we can prove that there exists a diagnosis of size less or equal than k if and only if $\langle f(e), (M', k) \rangle$ is a member of the language S. The function f of Figure 3.3 provided a table of $2^{|M|}$ elements, built according to the following algorithm. Let n be the total number of all possible faults. where `F'++` transform `F'` into its next lexicographic set of faults. The table T obtained from e using f contains, for each set of manifestations, the size of the minimal explanation for it. Using $f(e)$ and M', we can find this size with a simple look up in the table. Doing that, and comparing the result with k is polynomial, thus it can be formalized as a polynomial language S. □

The proof is very simple. However, finding that a problem is in the class $\langle \text{EXPTIME}, \text{P} \rangle$ is not a very useful result. The result of the preprocessing of

```
function f( e );
set_of_faults F';
array T[2^M] of integer;
begin
F':=empty_set;
while F' <> [true,...,true] do
  if |F'| < T[e(F')]
  then
    T[e(F')]:=|F'|;
  F'++;
done
return T;
end;
```

Figure 3.3: The function f

the fixed part may be of exponential size. While it may be affordable to spend exponential time preprocessing part of the input, the result of this phase should not be too large. In the example of the airplane diagnostic system, spending two weeks compiling the function e is affordable, while storing a huge quantity of data on the computer of a single plane is not. Time and space, from this point of view, have completely different status: time is not a problem, since the preprocessing is made off-line; on the converse, having to store a too large result of this phase may be a problem. In order to make this point clear, just consider how much memory a program can write if each operation it makes is to write a value on a different memory location. In order to overcome this drawback, we need some new definitions. In the next section we restrict the definition by replacing EXPTIME with smaller classes.

A final note about this class $\langle \text{EXPTIME}, \text{P} \rangle$. We defined a problem to be in this class if and only if the complexity after preprocessing is polynomial. However, we should specify if it must be polynomial in the size of the compiled data $\langle f(x), y \rangle$, or polynomial in the size of the original input $\langle x, y \rangle$. This distinction makes sense since $f(x)$ may be exponential w.r.t. x. Our assumption from now on will be that the post-processing algorithm is polynomial in the *size of the original input*. The proof of $\langle \text{EXPTIME}, \text{P} \rangle$ membership of diagnosis respect this constraint. However, since there are cases in which we want a polynomial algorithm working on an exponential input, we should specify the kind of machine used. This consideration is neglected in the next sections,

where we assume that the size of $f(x)$ is polynomial in x.

3.2 Classes of Compilability

The class $\langle \text{EXPTIME}, \text{P} \rangle$ has been introduced in the previous section to formally express the problems that are compilable to P if an exponential preprocessing is allowed. In this section we extend that definition. There are two directions in which the definition can be extended:

- Allow more/less computational power in the preprocessing phase.
- Allow more computational power in the on-line phase.

The first extension is justified by the fact that "exponential preprocessing" is not the only kind of preprocessing we are interested in: there are cases in which the preprocessing is easier, and other cases in which is harder.

The second extension is justified by the practice: we will show in the sequel that many problems cannot be solved in polynomial time even when a preprocessing is allowed. In such cases we are interested in how much we can reduce the complexity (even if we do not obtain something polynomial) via a preprocessing.

Consider for example a Π_2^p problem. It is possible that this problem is not in $\langle \text{EXPTIME}, \text{P} \rangle$: this means that the problem cannot be solved in polynomial time, even with the preprocessing. However, it may be that the complexity of the on-line processing is only NP. In this case, we denote the corresponding class with $\langle \text{EXPTIME}, \text{NP} \rangle$. This definition can be extended replacing NP with any other class of the polynomial hierarchy.

We also use classes different from EXPTIME to denote the computational power of the preprocessing. We define the class of problems of pairs $\langle \text{FC}, \text{C} \rangle$, where FC is a class of search problems, and C a class of decision problems. Intuitively, FC means that we are allowed to use any function in FC in the preprocessing. For example, in $\langle \text{EXPTIME}, \text{P} \rangle$, we were allowed to use an exponential function for the sake of preprocessing the fixed part of the input. In general, in the class $\langle \text{FC}, \text{C} \rangle$ we can use any function in FC.

The class C is used to mean that the on-line computation must be limited to the complexity of C, that is, the problem after the preprocessing must be a problem in C. The formal definition follows.

Definition 20 *A problem of pairs of strings A belongs to the class $\langle \text{FC}, \text{C} \rangle$ if and only if there exist a function f in* FC *and a language of pairs S in* C*, such that*

$$\langle x, y \rangle \in A \quad \textit{iff} \quad \langle f(x), y \rangle \in S$$

Indeed, $f(x)$ represents the result of the preprocessing phase. This phase is limited by the fact that f must be a function in FC. After this phase, we can use $f(x)$ and y to solve the original problem (which was: is it true that $\langle x, y\rangle \in A$?). The fact that the problem S must be in C is a bound on the complexity of the problem after the preprocessing.

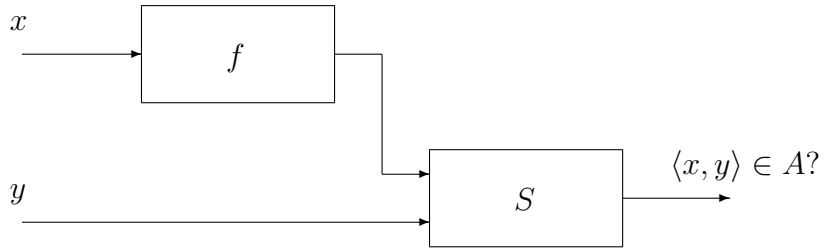


Figure 3.4: Generic class with compilation.

Figure 3.4 gives a visual representation of the class. Note that it is not a representation of an algorithm to decide whether $\langle x, y\rangle \in A$. Indeed, f is a function and S a language. The figure does not include the algorithms to calculate $f(x)$, neither the way to decide whether $\langle f(x), y\rangle$ is in S or not. It merely indicates what f and S are. If $f \in$ FC and $S \in$ C then the problem A is in $\langle \text{FC}, \text{C}\rangle$.

3.3 Compilability of Belief Revision

In this section we show how some belief revision operators can be compiled. Namely, we show that Dalal's revision can be compiled to coNP using a $\Delta_2^p[\log n]$ preprocessing, while the operators introduced for the iterated revision scenario can be compiled to coNP using a Δ_2^p preprocessing. Note that the complexity of Dalal's revision (without compilation) is $\Delta_2^p[\log n]$, while the complexity of the operators introduced for the iterated revision is Δ_2^p.

Let us consider Dalal's revision. We prove the following theorem.

Theorem 2 *The problem of deciding whether $T *_D P \models Q$ is in the class $\langle \Delta_2^p[\log n], \text{coNP}\rangle$, where the pair $\langle T, P\rangle$ is the fixed part and Q is the varying part of the problem.*

Proof. Dalal's revision can be reformulated as

$$Mod(T *_D P) = \{J \in Mod(P) \mid \exists I \in Mod(T) \,.\, Diff(I, J) = k_{T,P}\}$$

As a result, once known $k_{T,P}$, the statement $T *_D P \models Q$ is only coNP. Indeed, the opposite of the statement is equivalent to guess that there exists a pair

of models $I \in Mod(T)$ and $J \in Mod(P)$ such that $Diff(I, J) = k_{T,P}$ and $J \notin Mod(Q)$.

It is left to prove that $k_{T,P}$ can be determined with a polynomial number of calls to an NP oracle. This can be done as in [EG92]. We report the algorithm here for the sake of completeness. The algorithm starts with a value of k equal to $n/2$, where n is the number of variables involved. We check whether there exists a pair $I \in Mod(T)$ and $J \in Mod(P)$ such that $Diff(I, J) = k$. This is an NP problem. If such a pair exists, assigns $n/4$ to k, otherwise assign $3 \cdot n/4$ to k. This is a binary search, and ends after a number of steps that is logarithmic in the initial value of k, thus is $\log n/2$. The final value of k is the minimal value of the distance between models of T and models of P, that is, is equal to $k_{T,P}$. Note that $k_{T,P}$ is determined only in function of T and P, thus it can be found in the preprocessing phase. □

The proofs of compilability for the iterated revision operators are longer. Two revision operators have been presented here to treat the iterated case, namely, Spohn's revision and Boutilier's revision. Other ones can be found in the literature. The complexity of iterated revision has been analyzed in [Lib97b]. Table 3.1 reports the complexity results.

	inference	model checking	computation
Natural Revision	Δ_2^p complete	Δ_2^p complete	NP equivalent
Adjustment	Δ_2^p complete	Δ_2^p complete	NP equivalent
Condizionalization	Δ_2^p complete	Δ_2^p complete	NP equivalent
Prioritized Revision	Δ_2^p complete	coNP complete	NP equivalent
Ranked (general)	$\Delta_2^p[\log n]$-hard in Δ_2^p	D^p-hard in Δ_2^p	NP equivalent
Ranked (upper bound)	Δ_2^p complete	Δ_2^p complete	NP equivalent

Table 3.1: Complexity of iterated revision

The table contains three columns. For each revision it reports the complexity of the problem of query answering (deciding whether $T*[P_1,\ldots,P_m] \models Q$), the problem of model checking (deciding whether $M \models T*[P_1,\ldots,P_m]$), and finally, the problem of actually finding a formula that is equivalent to $T*[P_1,\ldots,P_m]$. These results hold if we assume that the initial ranking κ_T is polynomial, that is, $\kappa_T(I)$ can be determined in polynomial time for each I.

Let us concentrate on the problem of query answering. From the table follows that the complexity is, in almost all cases, Δ_2^p complete. We can prove that this complexity can be reduced via a preprocessing. First of all, we must specify which are the fixed and the varying part of the problem. The input is composed by a set of $m+2$ propositional formulas, namely T, P_1 to P_m, and Q. The result of the revision $T*[P_1,\ldots,P_m]$ represents our knowledge about the state of the world after the sequence of revisions. The way to extract information from this base is to query it, that is, given a fact Q we want to know if it is true or false, to check whether

$$T*[P_1,\ldots,P_m] \models Q \ ? \tag{3.1}$$

In most cases, a knowledge base must be queried many times, with many different Q's. If this is the case, it makes sense to compile the formulas $T, P_1, \ldots, P_m$ in order to obtain a representation that allows to solve the question (3.1) in less time.

We can prove that this is actually possible. Indeed, our results state that the complexity of query answering, when $T, P_1, \ldots, P_m$ are the fixed part, is $\langle \Delta_2^p, \text{coNP} \rangle$. This means that a Δ_2^p preprocessing of the knowledge base and the revisions reduces the complexity of the query answering problem to coNP.

The complexity without preprocessing was Δ_2^p. The preprocessing is Δ_2^p, too. However, given an initial knowledge base and a sequence of revision, the preprocessing must be done only once, and then we are able to solve the query answering problem w.r.t. many queries.

We give the full proof of compilability for Boutilier's natural revision. We use the following notation: k_i denotes the minimal value of $\kappa_{[T;P_1,\ldots,P_{i-1}]}(I)$ for $I \in Mod(P_i)$. This is the value of the ranking of the models that have their ranking changed by the revision. Note that is a value and not a function.

Lemma 1 *Given $\kappa_T, k_1, \ldots, k_i, P_1, \ldots, P_i$, finding $\kappa_{[T;P_1,\ldots,P_i]}(I)$ is polynomial.*

Proof. The definition of $\kappa_{[T;P_1,\ldots,P_i]}(I)$ can be rewritten as follows

$$\kappa_{[T;P_1,\ldots,P_i]}(I) = \begin{cases} 0 & \text{if } I \in Mod(P_i) \text{ and } \kappa_{[T;P_1,\ldots,P_{i-1}]}(I) = k_i \\ \kappa_{[T;P_1,\ldots,P_{i-1}]}(I) & \text{otherwise} \end{cases}$$

Since we already know k_i for each i, we can use this formula recursively to determine the value of the final ranking. At each step we must check only if $I \in Mod(P_i)$, which is polynomial, and then recursively calculate $\kappa_{[T;P_1,\ldots,P_{i-1}]}(I)$. Since at most i recursive calls are required, the whole procedure is polynomial. □

The previous lemma show how easy is to find the ranking when these magic numbers k_i are known. The next lemma shows that the complexity of finding the numbers k_i knowing the ranking is harder.

Lemma 2 *Given $\kappa_{[T;P_1,\ldots,P_i]}$ and P_{i+1}, finding k_{i+1} is in Δ_2^p.*

Proof. This is done by binary search. Since $\kappa_{[T;P_1,\ldots,P_i]}$ is a polynomial function, the length of the result is at most polynomially long. Let us for example assume that $||\kappa_{[T;P_1,\ldots,P_i]}|| \leq n^c$ where n is the total size of the input and c is a constant. With n^c bits we can write numbers with value inside the interval $[0,\ldots,2^{n^c}-1]$. Thus, the maximal possible ranking of an interpretation is $2^{n^c}-1$. In order to determine k_i we operate as follows. We start with an temporary value $k = (2^{n^c})/2$. Then we check whether there exists a model $I \in Mod(P_i)$ such that $\kappa_{[T;P_1,\ldots,P_i]}(I) \leq k$. This step is in NP, since it is a check of existence of an interpretation that satisfies two polynomial conditions. If such a model exists, we modify $k \leftarrow k/2$, otherwise we assign $k \leftarrow k3/4$.

This is essentially a binary search, which ends at most after $\log_2(2^{n^c}) = n^c$. Thus, this procedure requires a polynomial number of calls to an NP oracle, thus the problem is in Δ_2^p. □

Lemma 3 *Given $\kappa_T, k_1, \ldots, k_i, P_1, \ldots, P_{i+1}$, finding κ_{i+1} is in Δ_2^p.*

Proof. This is proved by induction. The value of k_1 is Δ_2^p by the previous lemma. Given k_1 and P_1, the function $\kappa_{[T;P_1]}$ is polynomial, thus k_2 can be determined with another polynomial number of calls to an NP oracle, and so on. □

These three lemmas allow to prove the compilability of the natural revision.

Theorem 3 *The problem of query answering for the natural revision is in $\langle \Delta_2^p, \text{coNP} \rangle$, where $T, P_1, \ldots, P_m$ are the fixed part and Q is the varying part.*

Proof. Given $T, P_1, \ldots, P_m$, determining k_m is Δ_2^p. Our compilation function is indeed

$$f(T, P_1, \ldots, P_m) = (k_1, \ldots, k_m, P_m)$$

The problem of deciding whether $T * [P_1, \ldots, P_m] \models Q$ is equivalent to decide whether

$$\min(Mod(P_m), \kappa_{[T;P_1,\ldots,P_{m-1}]}) \subseteq Mod(Q)$$

By definition, k_m has the property that

$$\min(Mod(P_m), \kappa_{[T;P_1,\ldots,P_{m-1}]}) = \{I \in Mod(P_m) \mid \kappa_{[T;P_1,\ldots,P_{m-1}]}(I) = k_m\}$$

Thus, $T * [P_1, \ldots, P_m] \not\models Q$ if and only if there exists a model of $Mod(P_m)$ that satisfies the properties $\kappa_{[T;P_1,\ldots,P_{m-1}]}(I) = k_m$ and $I \notin Mod(Q)$. These properties can all be determined in polynomial time, once known $k_1, \ldots, k_m$, thus the converse of the query answering is in NP. Since the compilation is Δ_2^p, the whole problem is $\langle \Delta_2^p, \text{coNP} \rangle$. □

3.4 Relationships between Classes

In this section we show some results about the relationship between classes of compilability, and between classes of compilability and usual classes of problems. First of all, the trivial results:

Theorem 4 *If* $\text{FC} \subseteq \text{FC}'$ *and* $\text{C} \subseteq \text{C}'$ *then* $\langle \text{FC}, \text{C} \rangle \subseteq \langle \text{FC}', \text{C}' \rangle$. *If any of the two above containment is strict the latter containment is strict.*

The following theorem is fundamental for the proofs of non-compilability of the next sections.

Theorem 5 *If* $\langle \text{FC}, \text{C} \rangle \subseteq \langle \text{FC}', \text{C}' \rangle$ *then* $\text{C} \subseteq \text{C}'$.

Proof. Let A be an arbitrary problem in C. The problem $*A$, defined as $*A = \{\langle x, y \rangle \mid y \in A\}$ is in $\langle \text{FC}, \text{C} \rangle$, and by hypothesis is in $\langle \text{FC}', \text{C}' \rangle$. Let ϵ be the 0-lenght string. It holds

$$\begin{array}{lll} y \in A & \text{iff} & \langle \epsilon, y \rangle \in *A \\ & \text{iff} & \langle f(\epsilon), y \rangle \in S \end{array}$$

where f is in FC' and S is in C'. Since ϵ is a constant string, $f(\epsilon)$ is also a constant string. Thus the problem of deciding whether $y \in A$ can be polynomially reduced to a single check of membership in a C' language. Thus the problem A is in C'. Since A is a generic C problem, the claim follows. □

The previous section showed that a problem such query answering in iterated belief revision can be compiled to coNP. The original complexity was Δ_2^p, and the compilability class is $\langle \Delta_2^p, \text{coNP} \rangle$. Note, however, that the problem can also be solved using an identity function as compilation function, and then solving on-line the problem as usual.

In general, any problem in C is also a problem in $\langle P, C \rangle$. Indeed, just use $f(x) = x$, and S equal to the original problem. The compilation function is linear, while the on-line processing is as hard as the whole problem.

The aim of compilation is to reduce the on-line complexity, thus to decrease the second element of the pair $\langle P, C\rangle$, even if the first class becomes higher in the polynomial hierarchy. In the case of Natural Revision, the problem belongs both to $\langle P, \Delta_2^p\rangle$ (using the identity function as compilation) and to $\langle \Delta_2^p, \mathrm{coNP}\rangle$. In the cases in which a compilation is acceptable (e.g. when the knowledge base must be queried many times) the second characterization is more significant, since it expresses the actual possibility of gaining from compilation.

In short: a problem may belong to more compilability classes, for example $\langle \mathrm{FC}_1, \mathrm{C}_1\rangle$ and $\langle \mathrm{FC}_2, \mathrm{C}_2\rangle$. Each class characterizes the problem from different points of view. If $\mathrm{FC}_1 \subseteq \mathrm{FC}_2$ then the compilation given by the first characterization is easier than that given by the second one. On the converse, if $C_2 \subseteq C_1$ then the on-line processing of the first case is harder than that of the second one. The classes give only a theoretical characterization. The actual choice of algorithms will be made on the base of the time allowed for the preprocessing and for the on-line solution.

Chapter 4

Reductions

So far, the definition of classes of compilability has been given. Using these classes, we were able to classify some problems from the point of view of the hardness to solve them when a preprocessing is allowed. What is missing is a tool to prove that a given problem does not belong to a class. This is shown in this chapter, where we provide the right kind of reductions and a new definition of hardness of problems. These definitions lead to the introduction of the class of function FPSIZE and the comp hierarchy, that will be heavily used in the next chapters. All the proofs that comes from the use of the concept of reduction are conditional, that is, the non-membership to classes holds only if the polynomial hierarchy does not collapse.

4.1 Reductions: Prerequisites

In the last chapter we defined the classes used to classify the computational properties of a problem when a preprocessing is allowed. Using this formalism, we are able to say that the problem of diagnosis is in the class $\langle \text{EXPTIME}, \text{P} \rangle$, and the problem of query answering for the iterated belief revision operators is $\langle \Delta_2^p, \text{coNP} \rangle$. These results state that a problem is in a given class. However, the problems may be still easier to solve, and these results give no hint about it. For example, it may be that the problem of query answering for iterated belief revision is $\langle \Delta_2^p, \text{P} \rangle$. We need a tool for proving that a problem cannot belong to a given class. As for the polynomial hierarchy, we employ a specific form of reduction.

We cannot use the standard polynomial many-one reductions, since they do not distinguish between the fixed and the varying part of a problem. Consider the following example.

Example 1 *Let A and B be the languages of pairs:*

$$\begin{aligned} A &= \{\langle x, y\rangle \mid x \text{ and } y \text{ are two satisfiable formulas}\} \\ B &= \{\langle x, y\rangle \mid x \text{ is a satisfiable formula and } y \text{ is any string}\} \end{aligned}$$

The problem A is $\langle \mathrm{NP}, \mathrm{NP}\rangle$ while B is simpler, since once known the satisfiability of x, which can done off-line, the problem is polynomial: B is $\langle \mathrm{NP}, \mathrm{P}\rangle$. The problem B is thus simpler than A, since it can be solved easily once the fixed part is preprocessed. However, using the polynomial many-one reductions, we are able to prove that $A \leq^{\mathrm{P}} B$. Indeed, just consider the mapping

$$f(\langle x, y\rangle) = \langle x \wedge y[X/Y], y\rangle$$

where $y[X/Y]$ denotes the new formula obtained replacing the variables X with new variables Y appearing nowhere else. The formulas x and y are both satisfiable if and only if $x \wedge y[X/Y]$ is satisfiable. Thus, $\langle x, y\rangle \in A$ if and only if $f(\langle x, y\rangle) \in B$. Since f can be computed in polynomial time, this is a polynomial many-one reduction from A to B.

This example shows that a problem we wish to classify as "difficult" can be polynomially reduced to an "easy" problem. This is a fault of the polynomial many-one reductions, and namely of the fact that they treat in the same manner the fixed and the varying part of the instances. In the above example the problem was that the varying part of the problem A was encoded in the fixed part of the problem B. This is as to say that using this reduction we are compiling the *varying* part of A, while in our schema only the fixed part is known by the preprocessing function.

This problem can overcome by using a new kind of reductions. In this section we just outline the basic properties that any reduction must have, in order to be suitable to define hardness and completeness of problems. The first properties are shared by the polynomial many-one reductions, and can be found also in basic textbooks of computational complexity, such as [GJ79]. In the following properties we consider a generic class of reduction. If a problem A is reducible to B we write $A \leq B$.

Definition 21 (Compatibility) *A class of reductions is compatible with the class of problems* C *if and only if $A \leq B$ and $B \in \mathrm{C}$ imply $A \in \mathrm{C}$.*

This definition formalizes the basic property that a comparison between problems must have: if the problem A can be reduced to B, then A is at most as difficult as B.

Definition 22 (Transitivity) *If $A \leq B$ and $B \leq C$ then $A \leq C$.*

Definition 23 (Hard problem existence) *There exists an hard problem for the class* C *w.r.t. a class of reductions if and only if there exists a problem* B *such that* $A \leq B$ *for each* $A \in \mathrm{C}$.

These properties are necessary to define hard problems for the classes. The classical polynomial many-one reductions are not compatible with the classes $\langle \mathrm{FC}, \mathrm{C} \rangle$, as shown in the example above, where $A \leq^{\mathrm{P}} B$ and $B \in \langle \mathrm{NP}, \mathrm{P} \rangle$, but A was not in that class. Formally, we have the following theorem.

Theorem 6 *The polynomial many-one reductions are not compatible with the classes* $\langle \mathrm{FC}, \mathrm{C} \rangle$ *where* FC *is a class greater than* C.

Proof. We prove that the problem A of the example above is not in $\langle \mathrm{NP}, \mathrm{P} \rangle$. Assume that A is in that class. Then, we can solve the problem of satisfiability of a propositional formula as follows: given a formula y, it is satisfiable if and only if $\langle \mathsf{true}, y \rangle \in A$. Since A is in $\langle \mathrm{NP}, \mathrm{P} \rangle$, there exists an NP function f and a polynomial language S such that $\langle x, y \rangle \in A$ if and only if $\langle f(x), y \rangle \in S$. As a result, y is satisfiable if and only if $\langle f(\mathsf{true}), y \rangle \in S$. Since $f(\mathsf{true})$ is a constant string, we have polynomially reduced the problem of satisfiability to S, which is polynomial. As a result, the problem of satisfiability is polynomial, which implies the collapse of the polynomial hierarchy. A similar proof can be used for any compilability class $\langle \mathrm{FC}, \mathrm{C} \rangle$. □

The polynomial many-one reductions suffers from yet another drawback. The main aim of proving hard a problem w.r.t. a class is to prove that it does not belong to any other class strictly contained in it (the same holds for the polynomial hierarchy). Using the polynomial many-one reduction there are (lot of) problems which are not complete in $\langle \mathrm{FC}, \mathrm{C} \rangle$, but that can be proved not to belong to any class $\langle \mathrm{FC}, \mathrm{C}' \rangle$ with $\mathrm{C}' \subset \mathrm{C}$.

Consider for example the problem

$$D = \{ \langle x, y \rangle \mid x \text{ is any string and } y \text{ is a satisfiable formula } \}$$

This problem belongs to $\langle \mathrm{coNP}, \mathrm{NP} \rangle$, but cannot be compiled further than NP, that is, it does not belong to classes $\langle \mathrm{coNP}, \mathrm{C} \rangle$ with $\mathrm{C} \subset \mathrm{NP}$: this is proved as we did in the previous theorem for A. However, this problem is not hard w.r.t. polynomial many-one reductions. Indeed, assume that the problem is complete: the problem E, defined as

$$E = \{ \langle x, y \rangle \mid x \text{ is a tautology and } y \text{ is any string } \}$$

is also in $\langle \mathrm{coNP}, \mathrm{NP} \rangle$, thus can be reduced to D. This implies that the coNP complete problem E can be polynomially reduced to the NP problem D, implying $\mathrm{coNP} \subseteq \mathrm{NP}$, and the collapse of the polynomial hierarchy.

In the next section we show a kind of reductions that has all the properties above with respect to the classes $\langle \mathrm{FC}, \mathrm{C} \rangle$.

4.2 Compilability Reductions and Hardness

The Example 1 of the last section showed that the usual many-one reductions are not suitable for formalizing the degree of hardness of a problem in the compilability classes. The problem is that the varying part of A is translated in part of the fixed part of B. This is equivalent to preprocessing the varying part of A, which is forbidden by definition.

The most easy way of translating an instance $\langle x, y\rangle$ of A into an instance of B is to map x into an x' and y into y' in such a way

$$\langle x, y\rangle \in A \quad \text{iff} \quad \langle x', y'\rangle \in B$$

However, this kind reduction is too restrictive. Using this definition, it is very difficult to show the hardness of a problem. On the contrary, we allow the varying part of the instance of the second problem to depend on the fixed part of the first one. This is not a problem, since we assume the on-line computation to be easier than the preprocessing. The definition is the following.

Definition 24 *An* FC *reduction from the problem A to the problem B is a triple $\langle f_1, f_2, g\rangle$, where f_1 and f_2 are unary functions in* FC *and g is a binary polynomial function, such that*

$$\langle x, y\rangle \in A \quad \textit{iff} \quad \langle f_1(x), g(f_2(x), y)\rangle \in B$$

If there exists a reduction from A to B we say that A is FC *reducible to B, written $A \leq^{\text{FC}} B$.*

This definition looks complex to understand. Figure 4.1 shows a representation of the generic reduction. The function f_1 transform the fixed part of x

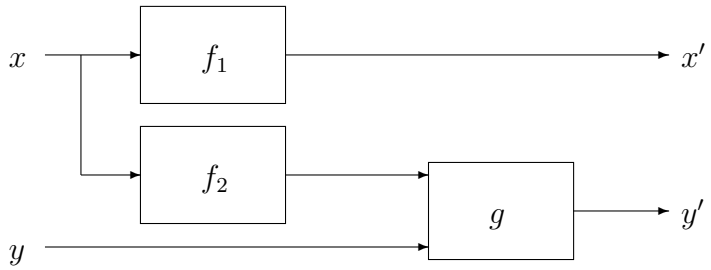


Figure 4.1: The generic FC reduction.

into the fixed part x' of the other problem. Since these are fixed parts of the problems, we are allowed to use all the computational power of the preprocessing, thus the only constraint is that $f_1 \in$ FC. The function g determines

the varying part y' in function of the varying part y. These are the parts of the input that are processed on-line, thus the function g has the same constraints of the many-one reductions, namely it must be polynomial. Finally, the function f_2 expresses the dependence of y' on x. Intuitively, the fixed part x determines not only the x', but also a way to translate y into y'. Indeed, once given x, the string y' can be found from y in time polynomial in the total size of the input.

Now we can show that the FC reductions have the right computational properties to characterize the hardness of problems w.r.t. the classes $\langle \mathrm{FC}, \mathrm{C} \rangle$. We assume that C is a class of the polynomial hierarchy. We also need some assumptions on the class FC, namely:

1. FC is limited in size: if $f \in \mathrm{FC}$ then $f(x)$ has size polynomial in the size of x

2. the composition and concatenation of FC functions are also FC functions: if f and h are in FC, then $f(h(x))$ and $f(x) \cdot h(x)$ are also in FC.

These assumptions are true for the classes Δ_i^p of the polynomial hierarchy, and all the classes of functions we use in the sequel (but is not compatible, for example, with the class EXPTIME).

Property 1 *The* FC *reductions are compatible with the class* $\langle \mathrm{FC}, \mathrm{C} \rangle$.

Proof. Let us assume that $A \leq^{\mathrm{FC}} B$ and $B \in \langle \mathrm{FC}, \mathrm{C} \rangle$: we prove that $A \in \langle \mathrm{FC}, \mathrm{C} \rangle$. Since $A \leq^{\mathrm{FC}} B$, there exist a triple $\langle f_1, f_2, g \rangle$, where f_1 and f_2 are in FC, and g is a polynomial function, such that

$$\langle x, y \rangle \in A \quad \text{iff} \quad \langle f_1(x), g(f_2(x), y) \rangle \in B$$

Since $B \in \langle \mathrm{FC}, \mathrm{C} \rangle$, there exists an FC function f and a C language S such that

$$\langle x, y \rangle \in B \quad \text{iff} \quad \langle f(x), y \rangle \in S$$

Combining these two properties, we have

$$\begin{array}{rll} \langle x, y \rangle \in A & \text{iff} & \langle f_1(x), g(f_2(x), y) \rangle \in B \\ & \text{iff} & \langle f(f_1(x)), g(f_2(x), y) \rangle \in S \end{array}$$

Let us consider

$$\begin{array}{rcl} f'(x) & = & \langle f(f_1(x)), f_2(x) \rangle \\ S' & = & \{ \langle \langle a, b \rangle, c \rangle \mid \langle a, g(b, c) \rangle \in S \} \end{array}$$

(nothing forbid us to define f' as a function whose result is a pair of strings rather than a single string, and S' to be defined over a pair whose first element is in turn another pair).

The function f' is in FC since by hypothesis the composition of FC functions is another FC function. Nevertheless, the language S' is C, since it involves the determination of a P function (g), and a check of membership in S, which is C. □

The FC reductions are also transitive, as the following property proves.

Property 2 *The class of* FC *reductions is transitive, for each* FC*.*

Proof. Let us assume $A \leq^{\mathrm{FC}} B$ and $B \leq^{\mathrm{FC}} C$. We prove that $A \leq^{\mathrm{FC}} C$. By hypothesis, there exists two FC reductions $\langle f_1, f_2, g\rangle$ and $\langle f_1', f_2', g'\rangle$ such that

$$\begin{array}{lll} \langle x, y\rangle \in A & \text{iff} & \langle f_1(x), g(f_2(x), y)\rangle \in B \\ \langle a, b\rangle \in B & \text{iff} & \langle f_1'(a), g'(f_2'(x), y) \in C \end{array}$$

Let us consider the reduction $\langle f_1'', f_2'', g''\rangle$ defined as

$$\begin{array}{rcl} f_1''(x) & = & f_1'(f_1(x)) \\ f_2''(x) & = & \langle f_2(x), f_2'(f_1(x))\rangle \\ g''(\langle a, b\rangle, y) & = & g'(b, g(a, y)) \end{array}$$

This is a FC reduction, since f_1'' and f_2'' are FC (composition and concatenation of FC functions are FC functions), and g'' is polynomial since g and g' are polynomial. Furthermore, we have

$$\begin{array}{lll} \langle x, y\rangle \in A & \text{iff} & \langle f_1(x), g(f_2(x), y)\rangle \in B \\ & \text{iff} & \langle f_1'(f_1(x)), g'(f_2'(f_1(x)), g(f_2(x), y)\rangle \in C \\ & \text{iff} & \langle f_1''(x), g''(f_2''(x), y)\rangle \in C \end{array}$$

As a result, $A \leq^{\mathrm{FC}} B$. □

Finally, the proof of existence of hard problem w.r.t. this kind of reductions. There is a natural assumption here: the considered classes C must have hard problems themselves. Since $\langle \mathrm{FC}, \mathrm{C}\rangle$ is in some sense an "extension" of C (it is C plus a preprocessing), it is natural to assume that C has an hard problem.

Lemma 4 *If B is a hard problem for the class* C *with respect to polynomial many-one reductions, then the following problem $*B$*

$$*B = \{\langle x, y\rangle \mid y \in B\}$$

is hard for the class $\langle \mathrm{FC}, \mathrm{C}\rangle$ with respect to FC *reductions.*

Proof. Let B be the generic hard problem for the class C w.r.t. polynomial many-one reductions. We prove that $*B$ is hard in $\langle \mathrm{FC}, \mathrm{C} \rangle$. Let A be a generic $\langle \mathrm{FC}, \mathrm{C} \rangle$ problem. By definition

$$\langle x, y \rangle \in A \quad \text{iff} \quad \langle f(x), y \rangle \in S$$

where f is FC and S is C. Since S is in C and B is C hard, there exists a polynomial function g such that

$$\langle x, y \rangle \in S \quad \text{iff} \quad g(x, y) \in B$$

Thus,

$$\begin{array}{lll} \langle x, y \rangle \in A & \text{iff} & \langle f(x), y \rangle \in S \\ & \text{iff} & g(f(x), y) \in B \\ & \text{iff} & \langle a, g(f(x), y) \rangle \in *B \quad \text{for any string } a \end{array}$$

As a result, $\langle f_1, f, g \rangle$ is FC reduction from A to B, where f_1 is any FC function. This proof works for any language A in $\langle \mathrm{FC}, \mathrm{C} \rangle$, thus $*B$ is hard in the class. □

Each pair of strings whose second element is a member of B is a member of $*B$. There is no constraint on the first element of the pair: it can be any string. These problem are chosen because they are the "easiest" hard problems in $\langle \mathrm{FC}, \mathrm{C} \rangle$. Their usefulness is shown in the sequel, and namely when we prove hard many non-trivial problems by reducing a $*B$ to them. From the above lemma it is trivial to prove the existence of hard problems for the class $\langle \mathrm{FC}, \mathrm{C} \rangle$.

Property 3 *The classes* $\langle \mathrm{FC}, \mathrm{C} \rangle$ *have hard problems w.r.t. the class of* FC *reductions, provided that* C *has hard problem w.r.t. polynomial reductions.*

Proof. By Lemma 4, the problem $*B$ is hard in $\langle \mathrm{FC}, \mathrm{C} \rangle$, for any C hard problem B. □

Given these properties, we can define the concept of hardness and completeness for the classes $\langle \mathrm{FC}, \mathrm{C} \rangle$. We start with the definition of hardness.

Definition 25 *The problem* A *is* $\langle \mathrm{FC}, \mathrm{C} \rangle$ *hard if, for any language of pairs* $B \in \langle \mathrm{FC}, \mathrm{C} \rangle$*, it holds* $B \leq^{\mathrm{FC}} A$*.*

Definition 26 *The problem* A *is* $\langle \mathrm{FC}, \mathrm{C} \rangle$ *complete if it is* $\langle \mathrm{FC}, \mathrm{C} \rangle$ *hard and is also a member of the class* $\langle \mathrm{FC}, \mathrm{C} \rangle$*.*

Since the FC reductions are transitive, it follows that, in order to prove that a problem A is $\langle \mathrm{FC}, \mathrm{C} \rangle$ hard, it suffices to prove that B can be reduced to A, where B is a previously proved hard problem.

The definition of hardness and completeness are useful for proving the non-membership of a problem to a class, and namely, to identify the smallest class to which a problem can be compiled to. In order to prove such results, we must prove that the $\langle \mathrm{FC}, \mathrm{C} \rangle$ hardness of a problem implies its non-membership to a class $\langle \mathrm{FC}, \mathrm{C}' \rangle$ with C' strictly contained in C.

From Theorem 5, which says that $\langle \mathrm{FC}, \mathrm{C} \rangle \subseteq \langle \mathrm{FC}, \mathrm{C}' \rangle$ implies $\mathrm{C} \subseteq \mathrm{C}'$, we derive the following corollary, which is fundamental in proving the non-membership of problems to classes.

Corollary 1 *If A is $\langle \mathrm{FC}, \mathrm{C} \rangle$ hard and A is in $\langle \mathrm{FC}, \mathrm{C}' \rangle$, then $\mathrm{C} \subseteq \mathrm{C}'$.*

Proof. We prove that $\langle \mathrm{FC}, \mathrm{C} \rangle \subseteq \langle \mathrm{FC}, \mathrm{C}' \rangle$, which implies $\mathrm{C} \subseteq \mathrm{C}'$ by the previous theorem. Let B be a generic $\langle \mathrm{FC}, \mathrm{C} \rangle$ problem. Since A is hard, we have $B \leq^{\mathrm{FC}} A$ by definition. Since A is in $\langle \mathrm{FC}, \mathrm{C}' \rangle$ we have $B \in \langle \mathrm{FC}, \mathrm{C}' \rangle$ by compatibility of the relation $\leq^{\mathrm{FC}}$. □

Given a problem A, what we do is to try to prove that it belongs to a class $\langle \mathrm{FC}, \mathrm{C} \rangle$, with C as small as possible. Once we proved the membership to such a class, we prove the hardness w.r.t. that class. If we manage to do that, we have proved that the problem do not belong to a class $\langle \mathrm{FC}, \mathrm{C}' \rangle$ with C' strictly contained in C. This way, we prove that the problem A can be compiled to C using a FC preprocessing, and that such a preprocessing cannot reduce the complexity any more.

4.3 FPSIZE and the comp Hierarchy

In the last section we showed how the FC reductions can be used to define the hardness for classes $\langle \mathrm{FC}, \mathrm{C} \rangle$ that uses a preprocessing in FC. This allows a classification of problems between two classes with the same power FC in the preprocessing.

In this section we begin considering classes and reductions with different preprocessing power. Let $\mathrm{FC} \subseteq \mathrm{FC}'$. First of all, $\leq^{\mathrm{FC}'}$ is more general than $\leq^{\mathrm{FC}}$.

Property 4 *Any* FC *reduction is also a* FC' *reduction, provided that* $\mathrm{FC} \subseteq \mathrm{FC}'$.

Proof. An FC reduction is a triple $\langle f_1, f_2, g \rangle$ with $f_1, f_2 \in \mathrm{FC}$. Since $\mathrm{FC} \subseteq \mathrm{FC}'$ the triple is also a FC' reduction. □

As a result, if $A \leq^{\mathrm{FC}} B$, then $A \leq^{\mathrm{FC}'} B$. The following theorem shows an effect of the last property on the hardness of problems.

Theorem 7 *If A is $\langle \mathrm{FC}, \mathrm{C}\rangle$ hard (w.r.t.* FC *reductions) then A is also $\langle \mathrm{FC}', \mathrm{C}\rangle$ hard (w.r.t.* FC′ *reductions), provided that* $\mathrm{FC} \subseteq \mathrm{FC}'$.

Proof. Since A is $\langle \mathrm{FC}', \mathrm{C}\rangle$ hard, it follows that $*B \leq^{\mathrm{FC}'} A$, where B is any C hard problem. Since any FC′ reduction is also a FC reduction, by previous lemma it follows $*B \leq^{\mathrm{FC}} A$. Since $*B$ is $\langle \mathrm{FC}, \mathrm{C}\rangle$ hard, it follows that A is $\langle \mathrm{FC}, \mathrm{C}\rangle$ hard too. □

Note that the converse of this theorem does not hold. In words, this theorem says that the FC′ reductions are *more general* than the FC reductions. This means that if a problem A can be reduced to B using a FC reduction, the same holds for FC′ reductions. On the converse, there may be a reduction of the second kind but not of the first kind.

The idea we use is the following. We have a $\langle \mathrm{FC}, \mathrm{C}\rangle$ problem A. We want to prove that it does not belong to any class $\langle \mathrm{FC}, \mathrm{C}'\rangle$ with $\mathrm{C}' \subset \mathrm{C}$. This is the main aim of proving hardness. We can prove such a claim with just by proving that

$$A \text{ is } \langle \mathrm{FC}', \mathrm{C}\rangle \text{ hard}$$

Let us assume we have proved this property. Thus, A is not in $\langle \mathrm{FC}', \mathrm{C}'\rangle$. Since $\mathrm{FC} \subseteq \mathrm{FC}'$, it follows that $\langle \mathrm{FC}, \mathrm{C}'\rangle \subseteq \langle \mathrm{FC}', \mathrm{C}'\rangle$. As a result, A is not in $\langle \mathrm{FC}, \mathrm{C}'\rangle$. We summarize this result in the following property.

Property 5 *If A is $\langle \mathrm{FC}', \mathrm{C}\rangle$ hard, then it does* not *belong to $\langle \mathrm{FC}, \mathrm{C}'\rangle$, provided that*

1. $\mathrm{C}' \subset \mathrm{C}$
2. *C' is closed under polynomial reductions*
3. $\mathrm{FC} \subseteq \mathrm{FC}'$

The logical steps in the analysis of a problem w.r.t. its properties of compilability are the following:

1. Decide the computation power FC of the preprocessing phase.
2. Prove that the problem is $\langle \mathrm{FC}, \mathrm{C}\rangle$, with C as small as possible (this is proved by giving an actual algorithm).
3. Prove that the problem is not in $\langle \mathrm{FC}, \mathrm{C}'\rangle$ with C' contained in C. This can be done in two ways
   (a) Prove that the problem is $\langle \mathrm{FC}, \mathrm{C}\rangle$ hard (preferred).
   (b) Prove that the problem is $\langle \mathrm{FC}', \mathrm{C}\rangle$ hard, for a class FC′ that contains FC.

About the point 3b, we note that the larger is the class FC', the easier is in general to prove the hardness. Indeed, any reduction that uses FC is also a reduction with FC', but not vice versa. It is useful to consider the largest possible class FC. Such a class must obey three properties showed in the last section, and reported here for convenience. Namely, given two function $f, h \in \mathrm{FC}$

1. $f(x)$ has size polynomial in the size of x
2. $f(h(x))$ is in FC
3. $f(x) \cdot h(x)$ is in FC

Consider the following class.

Definition 27 *The class* FPSIZE *is the set of all the functions (even the non-recursive ones) f such that $f(x)$ has size polynomial in the size of x.*

(or, more formally, for each $f \in \mathrm{FPSIZE}$ there exists a polynomial p such that $||f(x)|| \leq p(||x||)$). This class contains functions that obey the first property above. The second and third properties also hold for the class FPSIZE. Indeed the size of $f(h(x))$ is a polynomial of a polynomial, thus is itself a polynomial. The size of $f(x) \cdot h(x)$ is a sum of polynomials, thus is also a polynomial.

The class FPSIZE is the largest class of functions that satisfies the three properties. By the principle "the larger the class, the easier to prove the hardness" this is the best possible choice, when we arrive to the point 3b of the procedure above. This class contains undecidable functions, but this is not a problem, since we use this class only to prove negative (i.e. non-membership) results. What we generally give will be an FPSIZE reduction that proves the $\langle \mathrm{FPSIZE}, \mathrm{C} \rangle$ hardness of a problem. We proved that this implies that the problem is not in $\langle \mathrm{FC}, \mathrm{C}' \rangle$, for any C' contained in C and any FC contained in FPSIZE. We introduce shorthands for these concepts.

Definition 28 $\rightsquigarrow$C *is a shorthand for* $\langle \mathrm{FPSIZE}, \mathrm{C} \rangle$.

Definition 29 *A comp reduction is an* FPSIZE *reduction, and $\leq_{comp}$ is a shorthand for $\leq^{\mathrm{FPSIZE}}$*

We give a short summary of this very technical section. We proved that non-membership to a class $\langle \mathrm{FC}, \mathrm{C}' \rangle$ can be done by proving the $\langle \mathrm{FC}', \mathrm{C} \rangle$ hardness, where FC' contains FC and C contains C'. We also showed that this task is easier when the class FC' is large. For this reason we introduce the largest possible class that respects our principles, the class FPSIZE.

Finally, we introduce the classes $\rightsquigarrow$C and the corresponding comp reductions as shorthands for the classes and reduction with a FPSIZE functions of preprocessing.

4.4 Limitation of the comp Reductions

The class $\rightsquigarrow$C contains all the problems that are compilable to C after a preprocessing that produced a limited (polynomial-size) data structure. This is theoretical class, as it includes problems for which the compilation is a non-recursive function. The main aim of the class is indeed to prove the non-compilability of a problem: this is done by showing that the problem is hard for a class $\rightsquigarrow$C, and this implies the non-membership to $\rightsquigarrow$C$'$ for any C$'$ $\subseteq$ C, which in turn implies the non-membership to $\langle$FC, C$'\rangle$.

In this section we show that, although these classes are sometimes useful for the aim they were defined, sometimes they are not. Let us consider Dalal's revision, which is in $\langle\Delta_2^p[\log n], \text{coNP}\rangle$. The best thing the compilation could do is to decrease the after-preprocessing complexity to P. However, this is impossible, as the next theorem shows.

Theorem 8 *Dalal's revision is* compcoNP *complete.*

Proof. Membership follows from the fact that the problem is in the class $\langle\Delta_2^p[\log n], \text{coNP}\rangle$, and since $\Delta_2^p[\log n] \subseteq$ FPSIZE it is in compcoNP.

Hardness is proved by reduction from the compcoNP hard problem $*\mathsf{unsat}$. Since unsat (the problem of deciding the unsatisfiability of a propositional formula) is coNP complete, then $*\mathsf{unsat}$ is compcoNP complete. Let $\langle x, y\rangle$ be the generic instance of $*\mathsf{unsat}$. Let f_1, f_2, g be defined as follows.

$$\begin{aligned} f_1(x) &= (\mathsf{true}, \mathsf{true}) \\ f_2(x) &= \epsilon \\ g(a, b) &= \neg b \end{aligned}$$

The problem $T *_D P \models Q$, when T and P are the fixed part of the instance, is reformulated as a problem of pairs:

$$\mathsf{dalal} = \{\langle (T, P), Q\rangle \mid T *_D P \models Q\}$$

We prove that $\langle f_1, f_2, g\rangle$, which is a comp reduction, is a reduction from $*\mathsf{unsat}$ to dalal.

$$\begin{array}{lll} \langle x, y\rangle \in *\mathsf{unsat} & \text{iff} & y \text{ is not satisfiable} \\ & \text{iff} & \neg y \text{ is valid} \\ & \text{iff} & \mathsf{true} \models \neg y \\ & \text{iff} & \mathsf{true} *_D \mathsf{true} \models \neg y \\ & \text{iff} & \langle (\mathsf{true}, \mathsf{true}), \neg y\rangle \in \mathsf{dalal} \\ & \text{iff} & \langle f_1(x), g(f_2(x), y)\rangle \in \mathsf{dalal} \end{array}$$

As a result, dalal is compcoNP complete. □

There are other problems for which such a proof is impossible to give. Consider the problem of diagnosis. Formally

$$\mathsf{diagn} = \{\langle e, (M', k)\rangle \mid \text{there exists a diagnosis for } M' \text{ of size smaller than or equal to } k, \text{ for the system } e\}$$

The problem is NP complete. We proved that it is in $\langle \text{EXPTIME}, \text{P}\rangle$, but we also noted that an exponential compilation is acceptable only if it produces a limited output. In the case of diagnosis, however, this is not possible. First of all, the problem is of course in compNP. Suppose we tried to prove that is in $\rightsquigarrow$P, but did not succeed at it. What we would like to prove is that the problem is not compilable, and this could be proved by showing that is compNP hard. The following theorems shows that it is impossible.

Theorem 9 *If e is a constant function, then* diagn *is a polynomial problem.*

Proof. We assumed that e is represented as a set of effect, for each fault. If e is a constant function, this means that the set of effects that can be triggered by faults is constant-size. Thus, we can write down a table with a row for each set of effects, with the minimum set of faults that generates it. This table is again constant-size (and can be written in constant time), and it allows the solution of the problem of diagnosis in polynomial time. □

Actually, assuming that e is a constant function is not very relevant, since it is equivalent to assume that the relevant set of faults and effects are also limited in size. However, this result allows the proof of the next impossibility theorem.

Theorem 10 *If* diagn *is* compNP *hard then* P=NP*.*

Proof. Let us assume that diagn is compNP hard: thus there exists a comp reduction $\langle f_1, f_2, g\rangle$ from $*$sat to diagn:

$$\langle x, y\rangle \in *\mathsf{sat} \quad \text{iff} \quad \langle f_1(x), g(f_2(x), y)\rangle \in \mathsf{diagn}$$

We prove that there exists a polynomial algorithm for sat. Let y be a generic propositional formula.

$$\begin{array}{lll} y \in \mathsf{sat} & \text{iff} & \langle \epsilon, y\rangle \in *\mathsf{sat} \\ & \text{iff} & \langle f_1(\epsilon), g(f_2(\epsilon), y)\rangle \in \mathsf{diagn} \end{array}$$

Since ϵ is a constant string, $f_1(\epsilon)$ (the function e of the problem of diagnosis) is also a constant. Thus, the problem of diagnosis can be solved in polynomial time. Since $f_2(\epsilon)$ is also a constant, the string $\langle f_1(\epsilon), g(f_2(\epsilon), y)\rangle$ can be

obtained from y in polynomial time. The satisfiability of y can be solved by a polynomial translation to a polynomial problem. Thus, sat is polynomial, and hence P=NP. □

We always assume that the polynomial hierarchy is proper – all its classes are different. Following this assumption, the problem of diagnosis is not compNP hard. In the next section we show how it can however be proved that it cannot be compiled to P, that is $\mathsf{diagn} \notin \rightsquigarrow$P.

4.5 Non-uniform Classes and Reductions

The classes of compilability differs to the usual classes of the polynomial hierarchy for they have a preprocessing phase which can be in general more hard than the rest of the computation. In this preprocessing phase part only part of the input is known. If $\langle x, y\rangle$ is a generic instance, the fixed part x is preprocessed. The assumption behind is that this part of the input is known in advance, or that many instances to be solved share the same fixed part.

In this section we propose an extension of this idea. Let us assume that not only x, but also the size of the varying part y is known in advance (or it is shared by many instance of the problem to be solved). In such cases, it is natural to preprocess both x and the size of the varying part $||y||$. This is shown in Figure 4.2. The box denoted as $||.||$ represents the function that gives

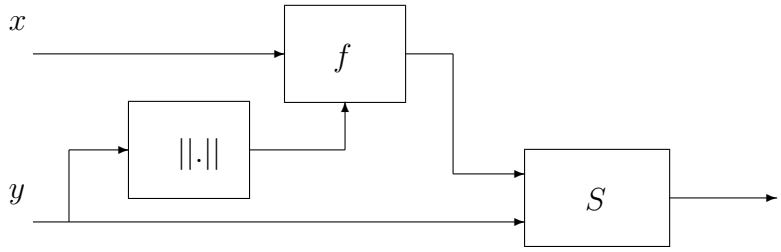


Figure 4.2: Preprocessing with size of the varying part.

the size of a string. The figure is identical of that of the generic compilation, with an exception: the function that represents the preprocessing f this time uses both x and $||y||$. The definition of the generic class $\|\!\rightsquigarrow$C follows.

Definition 30 *A problem of pairs of strings A belongs to $\|\!\rightsquigarrow$C if and only there exists a polysize function f and a* C *language S such that*

$$\langle x, y\rangle \in A \quad \textit{iff} \quad \langle f(x, ||y||), y\rangle \in S$$

The only difference with $\rightsquigarrow$C is that f has x and $||y||$ as inputs. The meaning of "polysize" this time treats differently x and $||y||$. Indeed, when we want to obtain a polynomial-size data structure as the result of compilation, we admit structures that are polynomial in the total size of the input $||x|| + ||y||$. However, the function f takes as inputs the whole string x, and the size $||y||$ which is itself a string whose size is polynomial in the size of y.

If we adopt the usual definition of polysize function, the f is limited by the fact that $f(x, ||y||)$ must have size polynomial in $||x|| + \log ||y||$. There are two ways to overcome this drawback: the first is to assume that the string $||y||$ used in the definition above is represented in unary notation (so that $||y||$, seen as a string, has the same size of y). The other way is to modify the definition of polysize functions, in such a way it treats differently the first and the second argument. We do not pay attention to the difference any more.

In order to prove non-membership to this class, what is left is the concept of reduction and hardness. The definition of the nucomp reductions is straightforward.

Definition 31 *A nucomp reduction from a problem A to a problem B is a triple $\langle f_1, f_2, g\rangle$ such that f_1 and f_2 are polysize and g is polynomial, and*

$$\langle x, y\rangle \in A \quad \textit{iff} \quad \langle f_1(x, ||y||), g(f_2(x, ||y||), y)\rangle \in B$$

If there exists a nucomp reduction from A to B, we say that A is nucomp reducible to B, written $A \leq_{nucomp} B$.

The only difference with the comp reductions are the fact that the polysize functions f_1 and f_2 have also $||y||$ as a second argument. A graphical representation of a nucomp reduction is given in Figure 4.3.

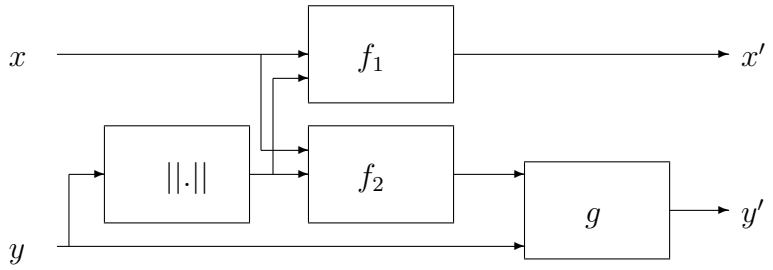


Figure 4.3: The nucomp reduction.

We prove that the nucomp reductions are indeed compatible with the classes of the non-uniform hierarchy, that are transitive, and that there exist hard problems.

Theorem 11 *If $A \leq_{nucomp} B$ and $B \in \|\!\rightarrowtail$C then $A \in \|\!\rightarrowtail$C.*

Proof. By hypothesis we have

$$\begin{array}{lll} \langle x,y\rangle \in A & \text{iff} & \langle f_1(x,||y||), g(f_2(x,||y||),y)\rangle \in B \\ \langle x,y\rangle \in B & \text{iff} & \langle f(x,||y||),y\rangle \in S \end{array}$$

Since g is a polynomial function, and f_2 is a polysize function, there exists a polynomial p such that $||g(f_2(x,||y||),y)|| \leq p(||x||+||y||)$. We assume, without loss of generality, that the size of $g(\ldots)$ is exactly equal to $p(||x||+||y||)$: if this is not true we can always add blanks to the result of g and modify S accordingly.

Let us define

$$\begin{array}{rcl} f'(x,k) & = & (f(f_1(x,k),p(||x||+k)),f_2(x,k)) \\ S' & = & \{\langle (a,b),c\rangle \mid \langle a,g(b,c)\rangle \in S\} \end{array}$$

The function f' is polysize, while S' is in the same class of S. We have

$$\begin{array}{lll} \langle x,y\rangle \in A & \text{iff} & \langle f_1(x,||y||), g(f_2(x,||y||),y)\rangle \in B \\ & \text{iff} & \langle f(f_1(x,||y||),||g(f_2(x,||y||),y)||), g(f_2(x,||y||),y)\rangle \in S \\ & \text{iff} & \langle f(f_1(x,||y||),p(||x||+||y||)), g(f_2(x,||y||),y)\rangle \in S \\ & \text{iff} & \langle f'(x,||y||),x\rangle \in S' \end{array}$$

Since f' is polysize and S' is in C, we proved that A is indeed in $\|\!\rightarrowtail$C. □

The next step is to prove the transitivity of the relation $\leq_{nucomp}$.

Theorem 12 *If $A \leq_{nucomp} B$ and $B \leq_{nucomp} C$, then $A \leq_{nucomp} C$.*

Proof. By hypothesis

$$\begin{array}{lll} \langle x,y\rangle \in A & \text{iff} & \langle f_1(x,||y||), g(f_2(x,||y||),y)\rangle \in B \\ \langle x,y\rangle \in B & \text{iff} & \langle f'_1(x,||y||), g'(f'_2(x,||y||),y)\rangle \in C \end{array}$$

Let us define the reduction $\langle f_1, f_2, g\rangle$ as

$$\begin{array}{rcl} f''_1(x,k) & = & f'_1(f_1(x,k),p(||x||+k)) \\ f''_2(x,k) & = & (f'_2(f_2(x,k),p(||x||+k)),f_2(x,||y||)) \\ g''((a,b),c) & = & g'(a,g(b,c)) \end{array}$$

This reduction reduces A to C. Indeed:

$$\begin{array}{lll}
\langle x,y\rangle \in A & \text{iff} & \langle f_1(x,||y||), g(f_2(x,||y||),y)\rangle \in B \\
& \text{iff} & \langle f_1'(f_1(x,||y||), ||g(f_2(x,||y||),y)||)), g'(f_2'(f_1(x,||y||), \\
& & ||g(f_2(x,||y||),y)||, g(f_2(x,||y||),y))\rangle \in C \\
& \text{iff} & \langle f_1'(f_1(x,||y||), p(||x||+||y||)), \\
& & g'(f_2'(f_1(x,||y||), p(||x||+||y||), g(f_2(x,||y||),y))\rangle \in C \\
& \text{iff} & \langle f_1''(x,||y||), g(f_2''(x,||y||),y)\rangle \in C
\end{array}$$

This proves that $A \leq_{nucomp} C$. □

The final step is the proof of hard problems for each of the class of the non-uniform hierarchy. The next theorem shows that the $*B$ problems (that are hard for the regular comp hierarchy) are hard for the non-uniform comp hierarchy. We remark that this is possible because increasing the power of the preprocessing phase, we increase also the power of the reduction.

Theorem 13 *If B is* C *hard then $*B$ is $\|\!\rightarrowtail$C hard.*

Proof. Let A be a generic $\|\!\rightarrowtail$C problem. By definition there exists a polysize function f and a C language S such that

$$\langle x,y\rangle \in A \quad \text{iff} \quad \langle f(x,||y||),y\rangle \in S$$

Since B is C hard, and S is in C, there exists a polynomial function h such that

$$\langle f(x,||y||),y\rangle \in S \quad \text{iff} \quad h(\langle f(x,||y||),y\rangle) \in B$$

Let us define the reduction

$$\begin{array}{rcl}
f_1(x,k) & = & \epsilon \\
f_2(x,k) & = & f(x,k) \\
g(a,y) & = & h(\langle ,a,y\rangle)
\end{array}$$

Indeed, we have $\langle x,y\rangle \in A$ if and only if $\langle \epsilon, h(\langle f(x,||y||),y\rangle)\rangle \in B$. □

About the relationships between uniform and non-uniform compilability classes, we can prove the following properties.

Property 6 *For each* C *it holds* $\rightsquigarrow$C $\subseteq$ $\|\!\rightarrowtail$C.

Property 7 *If $A \leq_{comp} B$ then $A \leq_{nucomp} B$.*

Using the nucomp reductions it is possible to prove that the problem of diagnosis is not in $\|\!\rightarrowtail$P.

Theorem 14 *The problem of diagnosis is* $\|\!\rightarrowtail$NP *complete.*

We prove this lemma first.

Lemma 5 *There exists a polynomial reduction from* 3sat *to* diagn *such that the set* M' *and the number* k *of the instance of diagnosis do not depend on the instance of the problem of satisfiability, but only on its size.*

Proof. Let $\Pi = \{\gamma_1, \ldots, \gamma_m\}$ be a set of clauses of three variables over the alphabet X. Let n be the size of Π. The number of variables of Π is less or equal than n. Let Π_n be the set of *all* the clauses of three variables over an alphabet on n variables.

We build the instance of the problem of diagnosis as following. The set of faults is the set of literals:

$$F = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$$

The set of manifestations is the set of the clauses of Π_n, that is $M = \Pi_n$. The function e is defined as

$$e(f) = \{\gamma \mid \gamma \in \Pi_n \text{ and } f \in \gamma\}$$

(that is, is the set of clauses that are satisfied by an interpretation that satisfies f). The set of current manifestation to be explained is

$$M' = \Pi \cup \{x_i \vee \neg x_i\}$$

and finally, $k = n$. Note that the only part of the instance of diagn that depends on Π and not only on its size is M'.

We prove that Π is satisfiable if and only if there exists a diagnosis for the system with size less or equal than $k = n$. First of all, since all the effects $x_i \vee \neg x_i$ must be explained, an explanation must contain at least a literal between x_i and $\neg x_i$. Moreover, since the explanation must be of size less or equal than n, it must contain exactly n literals.

Let us assume Π satisfiable. Let I be a model of Π. The set of faults $F' = I \cup \{\neg x_i \mid x_i \notin I\}$ has size exactly n. Moreover, since each clause in Π is satisfied by I, each $m \in M'$ is in $e(F')$. Thus, the problem of diagnosis has a diagnosis of size n.

Let us assume F' to be an explanation of size n for the problem of diagnosis. Let us define $I = \{x_i \mid x_i \in F'\}$ (that is, I is made only of the positive literals of F'). The interpretation I is a model of Π, since each clause of Π is explained by some literals of F'. □

Using this lemma it is easy to prove the claim.

Proof of Theorem 14. Since it is possible to give a reduction from 3sat to diagn in which only M' depends on the instance of Π, and the rest depends only on the size of Π, this is a nucomp reduction from $*$3sat to diagn. Figure 4.4 explains this point. The dashed lines and box stand for features of the nucomp

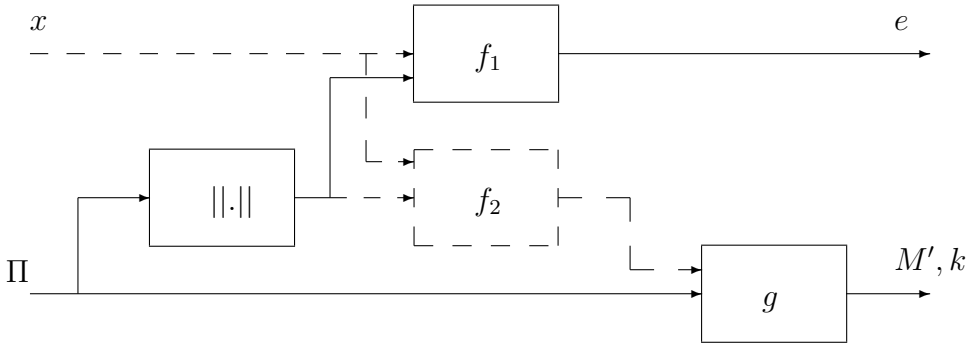


Figure 4.4: The reduction from 3sat to diagn.

reductions that are not used in this specific one. Namely, the function f_2, and the input string x are not used. The function f_1 determines the function e, while the function g determines M' and k. □

In order to prove the nucomp hardness of a problem, we often prove a statement like Lemma 5, proving that there exist reductions from sat or QBF such that the fixed part of the problem do not depends on the formula of the sat or QBF problem, but only on its size. This is a form of non-uniform reduction, and we do not usually state this fact explicitly.

4.6 Monotonic Polynomial Reductions

In this section we give some sufficient conditions for proving the $\|\!\rightharpoonup$NP hardness of problems. These condition are based on the concept of monotonic polynomial reductions, that are special cases of polynomial reduction from languages of string to languages of pairs.

A proof of NP hardness of a language B is usually a polynomial reduction from a (previously) proved NP hard problem A to B. Suppose, for example, that we have proved the NP hardness of a language of pairs B by means of a polynomial reduction from the problem 3sat. We state the following definition.

Definition 32 *A polynomial reduction $\langle r, h\rangle$ from* 3sat *to a language of pairs B is said to be monotonic if, for any two sets of clauses Π_1 and Π_2 over the*

same alphabet with $\Pi_1 \subseteq \Pi_2$*, it holds*

$$\langle r(\Pi_1), h(\Pi_1)\rangle \in B \quad \Leftrightarrow \quad \langle r(\Pi_2), h(\Pi_1)\rangle \in B \tag{4.1}$$

It can be proved that, given a monotonic polynomial reduction, one can build a proof of $\|\!\rightarrowtail$NP hardness for the problem B. Formally,

Theorem 15 *If there exists a monotonic polynomial reduction from* 3sat *to a problem of pairs* B*, then* B *is* $\|\!\rightarrowtail$NP *hard.*

Proof. Suppose there exists a monotonic polynomial reduction $\langle r, h\rangle$ from 3sat to B. We prove that there exists a nu-comp reduction f_1, f_2, g from the problem $*$3sat (that is compNP hard: see [CDLS96b]) to B.

Let $||\Pi||$ denote the size of Π (that is, the size of the string used to represent it), and $Var(\Pi)$ be the number of atoms in it. Furthermore Π_m is the set of all the clauses of three literals over a set of variables $\{x_1, \ldots, x_m\}$.

Let f_1, f_2, g be as follows

$$\begin{aligned} f_1(s, m) &= r(\Pi_m) \\ f_2(s, m) &= \epsilon \\ g(a, \Pi) &= h(\Pi \cup \{x_{Var(\Pi)+1} \vee \neg x_{Var(\Pi)+1}, \ldots, x_{||\Pi||} \vee \neg x_{||\Pi||}\}) \end{aligned}$$

Let $\Pi' = \Pi \cup \{x_{Var(\Pi)+1} \vee \neg x_{Var(\Pi)+1}, \ldots, x_{||\Pi||} \vee \neg x_{||\Pi||}\}$. Using the fact that $\langle r, h\rangle$ is a monotonic polynomial reduction we have that

$$\begin{aligned} \langle x, \Pi\rangle \in *\textsf{3sat} \quad & \text{iff} \quad \Pi \text{ is satisfiable} \\ & \text{iff} \quad \Pi' \text{ is satisfiable} \\ & \text{iff} \quad \langle r(\Pi'), h(\Pi')\rangle \in B \\ & \text{iff} \quad \langle r(\Pi_{Var(\Pi')}), h(\Pi')\rangle \in B \\ & \text{iff} \quad \langle r(\Pi_{||\Pi||}), h(\Pi')\rangle \in B \end{aligned}$$

But $r(\Pi_{||\Pi||}) = f_1(x, ||\Pi||)$ and $h(\Pi') = g(a, \Pi)$, thus $\langle x, \Pi\rangle \in *$3sat if and only if $\langle f_1(x, ||\Pi||), g(f_2(x, ||\Pi||), \Pi)\rangle \in B$. □

As a result, in order to prove that a problem B is not compilable, it suffices to find a polynomial reduction from 3sat to B, and then to prove that this reduction has the property of monotonicity (4.1). This is useful, because often there already exists a proof of hardness of the problem B that uses a reduction that turns out to be monotonic.

The following theorem, and its proof, shows how this result can be applied to the example of diagnosis. We need to specify which is the fixed and varying part. In this case, the set of possible effects, faults, and the functions e are the fixed part (since it is determined by a static analysis of the system to be diagnosed), while the set of the current effect (that is, the abnormal behavior to be analyzed) is the varying part.

Theorem 16 *The problem* diagn *is* $\|\!\rightsquigarrow$NP *hard (thus is not in* $\rightsquigarrow$P*).*

Proof. The problem diagn has been proved to be NP hard via the following reduction from 3sat, given in [ATBJ87]. Without loss of generality, assume that the given set of clauses Π_1 contains all the clauses of the form $x_i \vee \neg x_i$ for each x_i in the given alphabet. Now, the set of faults F is equal to the set of literals of the alphabet, and the set of effects M is equal to the set of clauses. The function e is defined as

$$e(f) = \{\ m \mid \text{the clause } m \text{ contains the literal } f\ \}$$

Finally, the set of current effects M' is equal to M, and the number k is equal to the number of atoms in the considered alphabet.

Now, consider two sets of clauses Π_1 and Π_2 over the same alphabet, with $\Pi_1 \subseteq \Pi_2$. The instance $\langle f(\Pi_1), g(\Pi_1)\rangle$ is built as in the previous construction. The instance $\langle f(\Pi_2), g(\Pi_1)\rangle$ has new effects (the clauses in $\Pi_2 \backslash \Pi_1$), and the function e is modified accordingly. However, these new effects need not to be explained (since the set of the current effects is the same of the previous case), thus a diagnosis of size k exists for the second instance if and only if exists for the first one. □

Note an interesting feature of this kind of proofs: we do not need to prove that the given reduction works, that is, we do not need to prove that a set of clauses is satisfiable if and only if the corresponding diagnosis problem has a solution of size less or equal than k. This can be proved elsewhere (in this case, in [ATBJ87]). All we have to do is to prove that this reduction is monotonic.

Of course, not all the NP hardness proofs use a reduction from 3sat. We now state the definitions of monotonicity for other kinds of reductions.

Definition 33 *A reduction* $\langle r, h\rangle$ *from* NC *(Node cover) or* clique *to a problem of pairs* B *is said to be monotonic if, for any two sets of edges* $E_1 \subseteq E_2$ *over the same set of nodes* N*, it holds*

$$\langle r((N, E_1), k), h((N, E_1), k)\rangle \in B \quad \Leftrightarrow \quad \langle r((N, E_2), k), h((N, E_1), k)\rangle \in B$$

The monontonic reductions from NC and clique have the same property of the mononotonic reductions from 3sat. Namely, they prove the $\|\!\rightsquigarrow$NP hardness of the problem.

Theorem 17 *If there exists a monotonic polynomial reduction from in* NC *or* clique *to a problem of pairs* B*, then* B *is* $\|\!\rightsquigarrow$NP *hard (and thus it does not belong to* $\rightsquigarrow$P*, unless the polynomial hierarchy collapses).*

Proof. We prove only the theorem for NC. The proof for clique is similar. First of all we prove that the problems kNC and lclique are $\|\not\hookrightarrow$NP hard.

$$\begin{aligned} \text{kNC} &= \{\langle k, G\rangle \mid G \text{ has a node cover of size } k\} \\ \text{lclique} &= \{\langle l, G\rangle \mid G \text{ has a clique of size } l\} \end{aligned}$$

There is no actual reason for compiling these problem. We prove their hardness just as an intermediate step for proving the theorem. The hardness of NC has been proved in [Kar72] by means of a reduction from clique, and clique is proved to be hard by means of a reduction from 3sat. The latter reduction goes as follows: given a set of clauses Π, the corresponding instance of clique has a structure $\langle|\Pi|, G\rangle$, where G is obtained from Π, and we do not care how. This is not a nucomp reduction, since the fixed part l of lclique is $|\Pi|$ and thus is not function of $||\Pi||$ only. However, it holds

$$\begin{aligned} \Pi \in \mathsf{3sat} \quad & \text{iff} \quad \Pi \cup \{x_{|\Pi|} \vee \neg x_{|\Pi|}, \ldots, x_{||\Pi||} \vee \neg x_{||\Pi||}\} \in \mathsf{3sat} \\ & \text{iff} \quad \langle||\Pi||, G'\rangle \in l\mathsf{clique} \end{aligned}$$

where G' is the graph corresponding to the set of clauses $\Pi \cup \{\ldots\}$. Since the fixed part l of the instance of lclique depends only on $||\Pi||$, this is a nucomp reduction.

Consider the reduction from clique to NC. In that reduction, given an instance $\langle l, (N, E)\rangle$ of lclique, the corresponding instance of kNC has $k = |N| - l$. This is not a nucomp reduction, since the fixed part of the instance of NC depends on $|N|$ and not only on l. However, it holds

$$\begin{aligned} \langle l, (N, E)\rangle \in \mathsf{lclique} \quad & \text{iff} \quad \langle l, (N \cup \{n_{|N|+1}, \ldots, n_{||(N,E)||}\}, E)\rangle \in \mathsf{lclique} \\ & \text{iff} \quad \langle|N \cup \{n_{|N|+1}, \ldots, n_{||(N,E)||}\}| - l, G'\rangle \in \text{kNC} \\ & \text{iff} \quad \langle||(N, E)|| - l, G'\rangle \in \text{kNC} \end{aligned}$$

where G' is the graph corresponding to $(N \cup \{\ldots\}, E)$ using Karp's reduction. This is a nucomp reduction from lclique to kNC.

Now we can prove that if there exists a monotonic polynomial reduction $\langle r, h\rangle$ from NC to a problem of pairs B, then B is $\|\not\hookrightarrow$NP hard. We prove this giving a nucomp reduction from the problem kNC above to B. Let

$$\begin{aligned} f_1(k, m) &= r(K_m, k) \\ f_2(k, m) &= k \\ g(k, (N, E)) &= h((N \cup \{n_{|N|+1}, \ldots, n_{||(N,E)||}\}, E), k) \end{aligned}$$

where K_m is the complete graph with m nodes. To prove that this is indeed a reduction from kNC to B, we use the hypothesis that $\langle r, h\rangle$ is a polynomial

monotonic reduction from NC to B. Let $N' = N \cup \{n_{|N|+1}, \ldots, n_{||(N,E)||}\}$. We have

$$\begin{array}{lll} \langle k, (N,E)\rangle \in \text{kNC} & \text{iff} & \langle k, (N',E)\rangle \in \text{kNC} \\ & \text{iff} & \langle r(k,(N',E)), h(k,(N',E))\rangle \in B \\ & \text{iff} & \langle r(k, K_{||(N,E)||}), h(k,(N',E))\rangle \in B \end{array}$$

Since

$$\begin{array}{rcl} r(k, K_{||(N,E)||}) & = & f_1(k, ||(N,E)||) \\ h(k,(N',E)) & = & g(f_2(k,||(N,E)||),(N,E)) \end{array}$$

the claim follows. □

The above theorem is useful for reductions whose starting point is a problem on graphs. For problems of sets, we give the definition of monotonic reduction using the problem exact cover (ec).

Definition 34 *A reduction $\langle r, h\rangle$ from the problem exact cover (ec) or three-exact cover (x3c) to a problem of pairs B is said to be monotonic if, for any two subsets S_1, S_2 of W, with $S_1 \subseteq S_2$, it holds*

$$\langle r(S_1), h(S_1)\rangle \in B \quad \Leftrightarrow \quad \langle r(S_2), h(S_1)\rangle \in B$$

For these monotonic reductions is possible to prove the same statement about the $\|\!\rightarrowtail$NP hardness of the problem.

Theorem 18 *If there exists a monotonic polynomial reduction from ec or x3c to a problem of pairs B, then B is $\|\!\rightarrowtail$NP hard (and thus it does not belong to $\rightarrowtail$P, unless the polynomial hierarchy collapses).*

4.7 Examples of Incompilable Problems

In this section we apply the techniques described in the last sections to prove that two well-known problems cannot be made tractable by preprocessing part of the input. Namely, in this section we introduce two NP complete problems for which a preprocessing of part of the input makes sense, and prove that they are $\|\!\rightarrowtail$NP hard.

4.7.1 Vertex Cover

The problem of vertex cover is the problem of finding a minimal set of nodes of a graphs that "covers" all the edges of a graph. The decision problem

associated is the following: given a graph (N, E) and a number k, decide whether there exists a subset $N' \subseteq N$ such that $|N| \leq k$ and such each edge have at least an extreme node in N'.

In general one may ask a cover of only a subset E' of the edges of the graph. The fixed part of the input, in this case, is the graph, while the varying part is the set of edges to be covered and the number k of the required cover. Since deciding this question is NP complete for the subcase in which $E' = E$, this problem is NP complete. About the compilability of this problem, we state the following theorem.

Theorem 19 *The problem of covering a subset of edges is* $\|\!\rightsquigarrow$NP *complete, if the graph is the fixed part of the input.*

Proof. Vertex cover is proved NP hard by reduction from 3sat. The reduction is reported for example in [GJ79], and is monotonic. We do not report the full proof here. We just prove that it is monotonic.

To each set of clauses Π_1 corresponds a graph G_1 and a number k. The set of edges to be covered is equal to the set of all the edges of G_1. Thus we have $r(\Pi_1) = G_1$ and $h(\Pi_1) = (E, k)$. The value of G_1 and k are irrelevant. The graph $G_2 = r(\Pi_2)$ is the graph corresponding to another set of clauses Π_2 that contains Π_1 but has the same variables. This graph is identical to G_1, but has new nodes and new edges. Clearly, each new node can only be linked to new edges.

We prove that G_2 has a cover of E of size k if and only if G_1 has a cover of size k. If G_1 has a cover, since the new edges of G_2 are not to be covered, this set is also a cover of G_2. On the converse, assume that G_2 has a set of nodes of cardinality k that covers all the edges in E. Assume that this set has a node not in G_1. In this case, all the edges linked to this node are not to be covered. As a result, this node can be removed from the cover. Thus, we can eliminate all the nodes not in G_1 from this cover. Eventually, we obtain a cover of E that contains only nodes of G_1. This is also a cover of G_1. □

4.7.2 Dominating Set

This is the problem of finding a subset of nodes of a graph in such a way that any other node is linked to one of these. The corresponding decision problem is: given a graph and a number, decide if there is a dominating set of cardinality less or equal than the number. A dominating set is a subset of nodes such that all the nodes of the graph are either in this set, or linked to a node in this set.

If we want to "dominate" only a subset of nodes, we obtain a problem in which there is a fixed part (the graph) and a varying part (the subset plus the number). This problem is not compilable.

Theorem 20 *The problem of deciding whether a graph G has a set of nodes of cardinality less or equal than k that covers a subset of nodes N' is $\|\rightharpoonup$NP hard, if the fixed part of the problem is the graph.*

Proof. The proof of NP hardness of the problem with all the nodes to be dominated is monotonic. The proof is harder than that of vertex cover. Indeed, the graph corresponding to Π_2 has new nodes and new edges, but the new nodes are linked to nodes of G_1.

The graphs G_1 and G_2 have a triple of nodes for each variable in Π_1 and Π_2 (the set of variables of Π_1 and Π_2 is the same). These nodes are linked to each other. Moreover, one of these nodes are linked only with the other two (and to no other node of the graph). Since the number k is equal to the number of atoms, this implies that a dominating set of size less or equal than k must contains exactly a node in each triple, and no other node. This holds both holds for the instance $\langle G_1, (N, k)\rangle$ and the instance $\langle G_2, (N, k)\rangle$. As a result, the dominating sets have only nodes that are shared by these graphs. Since the nodes in G_2 that are not in G_1 are not to be dominated, a dominating set of G_2 is also a dominating set of G_1. □

Chapter 5

Compilability Results

The last sections gave a precise framework for the study of the complexity of problems from the point of view of their compilability. In this chapter we apply to various problems the techniques introduced. First, we analyze the problem of compilability in belief revision. Then, circumscription and reasoning about actions. We conclude the chapter with a summary of results from graph theory and databases.

5.1 Compilability of Belief Revision

In this section we present the computational analysis of belief revision from the point of view of compilation. We start with the complexity of model checking, that is, given two formulas T and P, and a model M, decide whether $M \models T * P$. Afterwards, we study the query answering problem (decide whether $T * P \models Q$). We show that some results of hardness of query answering can be obtained from the results of model checking.

The formulas T and P are the fixed part of the input. Written as set of pairs of strings, the problems of model checking (MC) and query answering (QA) can be reformulated as:

MC (model checking) $= \{\langle (T, P), M\rangle \mid M \models T * P\}$

QA (query answering) $= \{\langle (T, P), Q\rangle \mid T * P \models Q\}$

In order to simplify the notations, we use some shorthands. Namely, if X and Z are sets of variables with the same cardinality, $X \equiv Z$ is a shorthand for $\bigwedge_{x_i \in X} x_i \equiv z_i$. In a similar manner, $X \not\equiv Z$ stands for $\bigwedge_{x_i \in X} x_i \not\equiv z_i$, and $X \vee Z$ stands for $\bigwedge_{x_i \in X} x_i \vee z_i$, $\neg X$ is $\{\neg x_i \mid x_i \in X\}$, etc.

5.1.1 Model Checking

The results of model checking of belief revision are summarized in Table 5.1. The fixed part of the problem is composed by T and P, while M is the varying part. The table has four columns. The first two contain the results when T

	General Case		Horn Case	
	Unbounded	Bounded	Unbounded	Bounded
Ginsberg	$\|\!\not\rightarrow$coNP complete	$\|\!\not\rightarrow$coNP complete	$\rightsquigarrow$P	$\rightsquigarrow$P
Winslett/ Borgida	$\|\!\not\rightarrow\Sigma_2^p$ complete	$\rightsquigarrow$P	$\|\!\not\rightarrow$NP complete	$\rightsquigarrow$P
Satoh	$\|\!\not\rightarrow\Sigma_2^p$ complete	$\rightsquigarrow$P	$\|\!\not\rightarrow$NP complete	$\rightsquigarrow$P
Forbus	$\|\!\not\rightarrow\Sigma_2^p$ complete	$\rightsquigarrow$P	$\|\!\not\rightarrow$coNP hard	$\rightsquigarrow$P
Dalal	$\|\!\not\rightarrow$NP complete	$\rightsquigarrow$P	$\|\!\not\rightarrow$NP complete	$\rightsquigarrow$P
Weber	$\|\!\not\rightarrow$NP complete	$\rightsquigarrow$P	$\rightsquigarrow$P	$\rightsquigarrow$P
WIDTIO	$\rightsquigarrow$P	$\rightsquigarrow$P	$\rightsquigarrow$P	$\rightsquigarrow$P

Table 5.1: Compilability of model checking for belief revision operators.

and P are generic formulas, while the other two report the compilability of the operators when the involved formulas T, P are Horn. Another distinction is made on the size of the revision P. Namely, in many cases the revising formula P is much smaller than the current knowledge base T (bounded case). In such cases the complexity often decreases. The same happens for the compilability results.

We need a preliminary lemma.

Lemma 6 *If there exist three polynomial functions r, s, and t, such that*

$$\Pi \textit{ is satisfiable} \quad \textit{iff} \quad r(\Pi) \models s(X) * t(X)$$

for each set of clauses of three literals Π such that $Var(\Pi) \subseteq X$, then the model checking of $*$ is $\|\!\not\rightarrow$NP hard.

Proof. We use a reduction from the problem $*\mathsf{3sat}$. Let Π be a set of clauses of three literals. Let the alphabet of Π be $X' = \{x_1, \ldots, x_n\}$. Define $X = \{x_1, \ldots, x_n, x_{n+1}, \ldots, x_m\}$, where m is determined by $||X|| = ||\Pi||$. Since $Var(\Pi) \subseteq X$ the hypothesis of the lemma holds, thus there exist three polynomial functions s, r, and t such that

$$\Pi \text{ is satisfiable} \quad \text{iff} \quad r(\Pi) \models s(X) * t(X)$$

Now, define

$$\begin{array}{rcl} f_1(x,l) & = & (s(\{x_1,\ldots,x_l\}), r(\{x_1,\ldots,x_l\})) \\ f_2(x,l) & = & \epsilon \\ g(z,y) & = & r(y) \end{array}$$

As a result, we have

$$\Pi \text{ is satisfiable} \quad \text{iff} \quad \langle f_1(x, ||\Pi||), g(f_2(x, ||\Pi||), \Pi)\rangle \in MC$$

for any string x, thus

$$\langle x, \Pi\rangle \in *\mathsf{3sat} \quad \text{iff} \quad \langle f_1(x, ||\Pi||), g(f_2(x, ||\Pi||), \Pi)\rangle \in MC$$

which implies that $\langle f_1, f_2, g\rangle$ is a nucomp reduction from $*\mathsf{3sat}$ to MC. □

A similar result can be proved for any other class of the polynomial hierarchy. If we are able to prove that the validity of a QBF formula $\exists\forall\ldots.\Pi$ is equivalent to $r(\Pi) \models s(X) * h(X)$, then the model checking problem for $*$ is $\|\!\sim\!\!\rightarrow\Sigma_i^p$ hard. The same holds for $\|\!\sim\!\!\rightarrow\Pi_i^p$.

We start the proofs of compilability with a useful result about Ginsberg's revision.

Lemma 7 *If P is consistent, $\{f_1, \ldots, f_m\} * P$ is equivalent to $\{P \wedge f_1, \ldots, P \wedge f_m\} * \mathsf{true}$.*

This property is useful since it proves that the complexity of Ginsberg's revision is the same if the revising formula P is bounded or not. Indeed, the unbounded revision can always be reduced to the bounded, thus the complexity do not decrease assuming that P has a fixed size.

Let us now prove the results of Table 5.1. The complexity of model checking is studied in [LS96]. While for most of the formalisms the level of compilability is directly related to the complexity of model checking, for others the two results are not correlated.

Theorem 21 *Ginsberg's revision model checking is $\|\!\sim\!\!\rightarrow$*coNP *complete.*

Proof. We show the thesis by providing a nucomp reduction from the problem of model checking of circumscription, which is proved $\|\!\sim\!\!\rightarrow$coNP complete in Theorem 48. Namely, we consider the following subproblem: given a propositional formula K and an interpretation I, decide if I is a minimal model of K, that is, if $I \models \mathit{CIRC}(K; X, \emptyset, \emptyset)$. The fixed part of the problem is K while the varying part is I. Given an instance $\langle K, I\rangle$ of this problem, we construct T, P and M as follows: $T = \{\neg x | x \in X\}$, $P = K$ and $M = I$. Notice that this a comp reduction and, hence, a nucomp reduction. By [LS96, Theorem 25] we have that I is a minimal model of K if and only if $M \models T *_G P$. As a consequence, Ginsberg's belief revision formalism is $\|\!\sim\!\!\rightarrow$coNP hard. Since model checking is in coNP we have that this formalism is $\|\!\sim\!\!\rightarrow$coNP complete. □

Theorem 22 *Model checking in Winslett's and Borgida's update is* $\|\!\sim\!\!\rightarrow\Sigma_2^p$ *complete.*

Proof. We will prove the $\|\!\sim\!\!\rightarrow\Sigma_2^p$-hardness of model checking for Winslett's operator only. The proof for Borgida's is similar. Let Π be a set of clauses of three literals over the alphabet $X \cup Y$. Let $\Pi_{X\cup Y}$ be the set of all the clauses of three literals over the alphabet $X \cup Y$. We prove that $\exists X \forall Y . \neg\Pi$ if and only if $M \models T *_W P$, where

$$\begin{aligned} T &= (X = Z) \wedge Y \wedge w \wedge r \\ P &= (\neg X \wedge \neg Y \wedge Z \wedge r \wedge \neg w) \vee \Big[(X = Z) \wedge ((\bigwedge_{\gamma_i \in \Pi_{X\cup Y}} c_i \rightarrow \gamma_i\ ) = w = r)\Big] \\ M &= \{c_i \mid \gamma_i \in \Pi\} \cup Z \cup \{r\} \end{aligned}$$

where C is a set of new variables, one to one with the clauses of $\Pi_{X\cup Y}$, and Z is a set of new variables, one to one with X. Given a set of clauses Π and a truth evaluation X_1 over the X, there is exactly one model of T, namely

$$I = \{c_i \mid \gamma_i \in \Pi\} \cup X_1 \cup \{z_i \mid x_i \in X_1\} \cup Y \cup \{w, r\}$$

By definition, $\mathit{Mod}(T *_W P) = \cup_{I \in \mathit{Mod}(T)} \mathit{Form}(I) *_W P$. We prove that

1. $M \not\models \mathit{Form}(J) *_W P$, if $J \cap (C \cup X) \neq I \cap (C \cup X)$.
2. $M \models \mathit{Form}(I) *_W P$, if and only if for each truth evaluation Y_1 over the y_i's the interpretation $X_1 \cup Y_1$ is not a model of Π.

The first proposition is easy to prove. Indeed, consider the model N defined as

$$N = (J \cap (C \cup X)) \cup [M \cap (Y \cup Z \cup \{w, r\})]$$

This is a model of P. Consider the symmetric difference between J and M:

$$Diff(J, M) = [Diff(J \cap (C \cup X), M \cap (C \cup X))] \cup Diff(J, N)$$

Since $J \cap (C \cup X)$ is different by hypothesis from $M \cap (C \cup X)$, we have that N is strictly closer to J than M. As a result, $M \not\models Form(J) *_W P$.

Let us now prove the second point, that is, $M \models Form(I) *_W P$ if and only if it is not possible to extend X_1 to form a model of Π.

Suppose Π unsatisfiable. Consider the set $Form(I) *_W P$. Since Π is false for any Y_1, the formula P has two kinds of models (apart from M): the models with $C = C_1$, $X = X_1$ and both w and r false, and the models with a different evaluation of C or X. None of these models can prevent M from being a model of the revision $Form(I) *_W P$. The models with a different C or X are not less distant from I than M; Indeed, let J be that model: if I has both x_i and z_i, but J has none of these literals, then J is not closer of I than M, since the last has z_i but not x_i. The same argument holds if I has none of x_i and z_i but J has both. For the differences on C the fact that J is not closer to I than M is trivial, since I and M agree on C, while I and J do not. The models with the same value of X but a negative value of w and r are more distant from I than M, since M and I agree on w and r while the other models disagree.

Suppose Π satisfiable. We can prove that M is not a model of $Form(I) *_W P$. Consider that for any X_1 we have in P the model $M' = \{c_i \mid \gamma_i \in \Pi\} \cup X_1 \cup Z_1 \cup Y_1 \cup \{w, r\}$ (with $Z_1 = \{z_i \mid x_i \in X_1\}$) where Y_1 is the value of Y that satisfies Π. This model is closer to I than M, thus M is not in the revised base.

As a result, we have that M can be a model only of $Form(I) *_W P$, where $I \cap C = \{c_i \mid \gamma_i \in \Pi\}$. Moreover, M is a model of this partial revision if and only if for any truth evaluation Y_1, it holds that $(I \cap X) \cup Y_1$ is not a model of Π. As a result, M is a model of $T *_W P$ if and only if there exists a truth evaluation X_1 over X such that for any Y_1, the interpretation $X_1 \cup Y_1$ is not a model of Π.

The same proof can be used to prove the complexity of Borgida's operator. □

Theorem 23 *Model checking for Satoh's revision is $\|\!\rightarrow\Sigma_2^p$-complete.*

Proof. We reduce Winslett's revision to Satoh's using a comp reduction. Let T and P be two formulas, and M be a model over the alphabet X. We prove

that $M \models T *_W P$ if and only if $M' \models T' *_S P'$, where

$$\begin{aligned} T' &= T \wedge (X = Y) \wedge (Y \neq Z) \\ P' &= \neg Y \wedge \neg Z \wedge P \\ M' &= M \end{aligned}$$

where Y and Y are set of new variables one-to-one with X. Note that M' is not the same as M, since it is considered over an extended alphabet.

Let us consider the formal definitions of $M \models T *_W P$ and $M' \models T' *_S P'$.

$$\begin{array}{rll} M \models T *_W P & \text{iff} & \exists I \in Mod(T)\ \nexists N \in Mod(P)\ .\ Diff(I,N) \subset Diff(I,M) \\ M' \models T' *_S P' & \text{iff} & \exists I' \in Mod(T')\ \nexists J' \in Mod(T'), N' \in Mod(P')\ . \\ & & Diff(J',N') \subset Diff(I',M') \end{array}$$

Let us consider the second definition. Assume $y_i \notin Diff(J',N')$ and $y_i \in Diff(I',M')$. Since $y_i \notin N'$ this implies $y_i \notin J'$, thus $z_i \in J'$. As a result, $z_i \in Diff(J',N')$, which implies $z_i \in Diff(I',M')$, thus $z_i \in I'$ and also $y_i \notin I'$ and thus $y_i \notin Diff(I',M')$ contradicting the hypothesis. As a result, $Diff(J',N') \subset Diff(I',M')$ implies $J' \cap (Y \cup Z) = I \cap (Y \cap Z)$.

Since the value of each x_i in I' is equal to the value of y_i, and the same for J', we have $I' = J'$. As a result

$$Diff(J',N') \subset Diff(I',M') \quad \text{iff} \quad Diff(I',N') \subset Diff(I',M')$$

The definition of $M' \models T' *_S P'$ can thus be simplified

$$M' \models T' *_S P' \text{ iff } \exists I' \in Mod(T')\ \nexists N' \in Mod(P')\ .\ Diff(J',N') \subset Diff(I',M')$$

This is very close to the definition of Winslett's update. The only point left to prove is the mapping between I and I' and N and N'. This is clearly true, since the models of P and P' are the same, while the models of T' are extensions of the models of T. Formally we have

$$\begin{array}{rll} I' \in Mod(T') & \text{iff} & I \cap X \in Mod(T) \text{ and } x_i \in I' \text{ iff } y_i \in I' \text{ iff } z_i \notin I' \\ N' \in Mod(P') & \text{iff} & N \cap (Y \cup Z) \text{ and } N \cap X \in Mod(P) \end{array}$$

This means that our condition $M' \models T' *_S P'$ can be rewritten further

$$\begin{array}{rll} M' \models T' *_S P' & \text{iff} & \exists I' \in Mod(T') \\ & & \nexists N' \in Mod(P')\ .\ Diff(J',N') \subset Diff(I',M') \\ & \text{iff} & \exists I \in Mod(T)\ \nexists N \in Mod(P)\ . \\ & & Diff((I \cup \{y_i \mid x_i \in I\} \cup \{z_i \mid x_i \notin I\}), N) \subset \\ & & Diff((I \cup \{y_i \mid x_i \in I\} \cup \{z_i \mid x_i \notin I\}), M) \\ & \text{iff} & \exists I \in Mod(T)\ \nexists N' \in Mod(P)\ .\ Diff(I,N) \subset Diff(I,M) \\ & \text{iff} & M \models T *_W P \end{array}$$

This proves that model checking under Winslett's semantics can be reduced to model checking under Satoh's. Now, since T' and P' depends only on T and P, this is indeed a comp reduction. Since Winslett's revision is $\|\!\sim\Sigma_2^p$ complete, this is a proof of $\|\!\sim\Sigma_2^p$ completeness of Satoh's revision. □

Theorem 24 *Forbus' update model checking is* $\|\!\sim\Sigma_2^p$ *complete.*

Proof. Membership follows from the complexity result of [LS96]. In order to prove hardness, we use a reduction from $*\exists\forall$QBF. Let Π be a set of clauses, each composed by three literals, with variables $X \cup Y$. We assume, without loss of generality, that $|X| = |Y| = n$. We prove that $\exists X \forall Y.\neg\Pi$ if and only if $M \models T *_F P$, where

$$\begin{array}{rcl} T & = & (X = Z) \wedge \neg Y \wedge \neg W \wedge \neg r \\ P & = & (X \wedge \neg Z \wedge \neg Y \wedge \neg W \wedge r) \vee [(X = Z) \wedge (Y \neq W) \wedge \neg r \wedge \\ & & \bigwedge_{\gamma_i \in \Pi_{X \cup Y}} c_i \rightarrow \gamma_i \\ M & = & \{c_i \mid \gamma_i \in \Pi\} \cup X \cup \{r\} \end{array}$$

Notice that T and P do not depend on Π, but only on its alphabet. Let us assume that there exists a truth evaluation X_1 of the x_i's such that Π is false for each model over $X \cup Y$ extending X_1. Consider the following model of T:

$$I = X_1 \cup Z_1 \cup \{c_i \mid \gamma_i \in \Pi\}$$

where $Z_1 = \{z_i \mid x_i \in X_1\}$. We prove that $M \models \mathit{Form}(I) *_F P$. The distance between I and M is

$$|\mathit{Diff}(I, M)| = |X \backslash X_1| + |Z_1| + 1 = n + 1$$

Now, consider a model N of $(X = Z) \wedge (Y \neq W) \wedge \ldots$. Since Π is false for each model extending X_1, a model of the latter formula must have a different truth evaluation for X or a different evaluation for C. Thus, N must be a model of the kind:

$$N_1 = X_2 \cup Z_2 \cup Y_2 \cup W_2 \cup \{c_i \mid \gamma_i \in \Pi\}$$

where $W_2 = \{w_i \mid y_i \notin Y_2\}$, provided that Π is true in the model $X_2 \cup Y_2$. Or a model of the kind:

$$N_2 = X_1 \cup Z_1 \cup Y_1 \cup W_1 \cup C_2$$

where Y_1 is any valuation of the set Y and $W_1 = \{w_i \mid y_i \notin Y_1\}$, provided that $X_1 \cup Y_1 \cup C_2 \models \Pi_{X \cup Y}$. The difference between I and N_1 is

$$\begin{aligned} |Diff(I, N_1)| &= |Diff(X_1, X_2)| + |Diff(Z_1, Z_2)| + |Y_2| + |W_2| \\ &= |Diff(X_1, X_2)| + |Diff(Z_1, Z_2)| + n \\ &> n + 1 \end{aligned}$$

This is because X_1 and X_2 differ for at least a literal, and the same holds for Z_1 and Z_2. The difference between I and N_2 is

$$\begin{aligned} |Diff(I, N_2)| &= |Y_1| + |W_1| + |Diff(C_2, \{c_i \mid \gamma_i \in \Pi\})| \\ &= n + |Diff(C_2, \{c_i \mid \gamma_i \in \Pi\})| \\ &\leq n + 1 \end{aligned}$$

The update P has two kind of models, namely M and the models of $(X = Z) \wedge (Y \neq W) \wedge \ldots$. Since the models of the latter (N_1 and N_2) are not closer to I than M, it follows that M is a model of $Form(I) *_F P$, thus it is a model of $T *_F P$.

Now, let us assume that there is no truth evaluation X_1 such that all the models extending X_1 over $X \cup Y$ falsifies Π. Thus, for each X_1 there exists a Y_1 such that $X_1 \cup Y_1 \cup \{c_i \mid \gamma_i \in \Pi\} \models \bigwedge_{\gamma_i \in \Pi_{X \cup Y}} c_i \rightarrow \gamma_i$. Consider the *generic* model I of T:

$$I = X_1 \cup Z_1 \cup C_1$$

where $Z_1 = \{z_i \mid x_i \in X_1\}$. We prove that M is not a model of $Form(I) *_F P$ for any $I \in Mod(T)$, thus proving that M is not a model of $T *_F P$. The distance between I and M is

$$\begin{aligned} |Diff(I, M)| &= |X \backslash X_1| + |Z_1| + |Diff(C_1, \{c_i \mid \gamma_i \in \Pi\})| + 1 \\ &= n + 1 + |Diff(C_1, \{c_i \mid \gamma_i \in \Pi\})| \end{aligned}$$

Since for each X_1 there is a model $X_1 \cup Y_1 \cup \{c_i \mid \gamma_i \in \Pi\} \models \Pi_{X \cup Y}$, the formula P contains the following model

$$N = X_1 \cup Z_1 \cup Y_1 \cup W_1 \cup C_1$$

where $W_1 = \{w_i \mid y_i \notin Y_1\}$. Consider the distance between I and N

$$|Diff(I, N)| = |Y_1| + |W_1| = n$$

As a result, M is not a model of $Form(I) *_F P$, and this holds for any $I \in Mod(T)$. As a result, M is not a model of $T *_F P$. □

Theorem 25 *Model checking for Dalal's and Weber's revisions is* $\|\!\rightarrow$NP *complete.*

Proof. Membership: Given T and P, we off-line compute the minimal distance between models of T and P, the number $k_{T,P}$ (for Dalal's operator) and Ω (for Weber's operator). Now, given a model M, this is a model of $T * P$ if and only if there exists another model N such that $N \models T$ and $|Diff(N, M)| = k_{T,P}$ (for Dalal's operator) or $N \models T$ and $Diff(N, M) \subseteq \Omega$ for Weber's operator. Notice that both checks can be completed with an NP-machine. As a consequence, the compilability level of model checking in $\|\!\rightarrow$NP.

Hardness: We show a nucomp reduction from $*$3sat. Given a propositional formula Π on an alphabet X of n letters and a set Y of n new letters in one-to-one correspondence with letters of X, we define:

$$\begin{aligned} T &= X \not\equiv Y \wedge \bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i \\ P &= \neg X \wedge \neg Y \\ M &= \{c_i \mid \gamma_i \in \Pi\} \end{aligned}$$

We show that Π is satisfiable iff $M \models T *_D P$ iff $M \models T *_{Web} P$.

Notice that in every model of T exactly n atoms from $X \cup Y$ are true, while in every model of $T *_D P$ all atoms from $X \cup Y$ are false. Hence, recalling the definition of Dalal's revision, $k_{T,P} \geq n$. Moreover, $k_{T,P} = n$ since X is a model of T and $\emptyset$ is a model of P, and the cardinality of their difference is n. Now $\{c_i \mid \gamma_i \in \Pi\}$ is a model of P, hence it is also a model of $T *_D P$ iff there exists a model I of T such that $|Diff(I, M)| = k_{T,P} = n$.

Assume Π is satisfiable. Let X_1 be a model of Π, $Y_1 = \{y_i \mid x_i \notin X_1\}$ and $I = \{c_i \mid \gamma_i \in \Pi\} \cup X_1 \cup Y_1$. We show that I is a model of T. In fact, I satisfies $X \not\equiv Y$ by construction of Y_1, and also I satisfies $\bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i$, because for each clause $c_i \rightarrow \gamma_i$, either $c_i \notin I$ or γ_i is satisfied by X_1. Now observe that $|Diff(I, M)| = |X_1 \cup Y_1| = n$. Hence, M is a model of $T *_D P$. Since the models of Dalal's revision are also models of Weber's revision, we also have that M is a model of $T *_{Web} P$.

Now, observe that $Mod(P) = 2^C$, i.e. every subset of C is a model of P. Since X and Y are models of T, $\mu(X, P) = X$, $\mu(Y, P) = Y$. This implies that both X and Y are in $\delta(T, P) \doteq \bigcup_{I \in Mod(T)} \mu(I, P)$. Recall that $\Omega \doteq \cup\delta(T, P)$. Hence, $X \cup Y \subseteq \Omega$. Moreover, since $Var(P) \cap C = \emptyset$, it holds $\Omega = X \cup Y$. From the definition of Weber's revision, M is a model of $T *_{Web} P$ iff there exists a model I of T such that $I \backslash \Omega = M \backslash \Omega$. Since $\Omega = X \cup Y$, the last condition is equivalent to $I \cap C = \{c_i \mid \gamma_i \in \Pi\}$.

Assume Π is unsatisfiable and M is a model of $T *_{Web} P$. Then there exists a model I of T such that $I \cap C = M$. We claim that $I \cap X$ is a model of Π. In

fact, as for Dalal's revision, I satisfies $\bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i$. Simplifying Π_X with truth values of $I \cap C = \{c_i \mid \gamma_i \in \Pi\}$, we conclude that I satisfies $\bigwedge_{\gamma_i \in \Pi} \gamma_i$, which is exactly the formula Π. Since this formula contains only atoms from X, $I \cap X$ satisfies Π, hence Π is satisfiable and contradiction arises. Thus, M is not a model of $T *_{Web} P$. Since the set of models of $T *_{Web} P$ is a superset of the models of $T *_D P$, it follows that M is not a model of $T *_D P$. □

Theorem 26 *Model checking for WIDTIO revision is in* $\rightsquigarrow$P.

Proof. Given T and P, we can compute off-line $T' = T *_{Wit} P$, since, by definition, the size of T' is less than or equal to the size of T. Online, we only need to check whether $M \models T'$, that can be accomplished in polynomial time. As a consequence, model checking is in $\|\!\rightsquigarrow$P. □

Bounded Case

We prove that all these revision operators but Ginsberg's have a $\rightsquigarrow$P model checking.

Theorem 27 *Model checking for Forbus', Winslett's, Borgida's, Satoh's, Dalal's, Weber's, and WIDTIO revision is* $\rightsquigarrow$P*, if the formula P has constant size.*

Proof. Model checking for Forbus' and Winslett's revisions is in P, thus it is also in $\rightsquigarrow$P. Borgida's revision has a coNP complete model checking, but once determined whether T and P are consistent (which can be done in the preprocessing phase) the problem reduces to Winslett's revision.

The problem $M \models T *_D P$ can be solved as follows: let $k_{T,P}$ be the minimum value of $|Diff(I, J)|$ for $I \in Mod(T)$ and $J \in Mod(P)$. We have that $M \models T *_D P$ if and only if there exists a model $I \in Mod(T)$ such that $Diff(I, M) = k_{T,P}$. By definition, $k_{T,P} \leq |Var(P)|$. As a result, there are only a constant number of models I such that $Diff(I, M) = k_{T,P}$. Thus determining whether one of these models I is in $Mod(T)$ can be solved in linear time.

A similar algorithm can be used for Satoh's revisioni. First, we determine the set $K_{T,P}$. Any element of $K_{T,P}$ is a subset of $Var(P)$, thus $K_{T,P}$ is a set of constant-size sets. Given M, it holds $M \models T *_S P$ if and only if there exists a model $I \in Mod(T)$ such that $Diff(I, M) \in K_{T,P}$. Since $K_{T,P}$ has constant size, there are only a constant number of I such that $Diff(I, M) \in K_{T,P}$. Verifying whether $I \in Mod(T)$ for each of these is linear, thus $M \models T *_S P$ is a linear problem. □

Ginsberg's revision (as proved before) has the same compilability properties in the bounded and unbounded case.

Horn Case

We prove the level of compilability of revision in the Horn case. We need a lemma that is implicit in Lemma 7.1 of [EG92]. We give the full proof for the sake of completeness.

Lemma 8 *Let γ be a clause over X, and γ^{neg} be the clause obtained by replacing any positive literal x_i in γ with $\neg y_i$. Then, γ is satisfied by a model $X_1 \subseteq X$ if and only if $X_1 \cup Y_1$ is a model of γ^{neg}, where $Y_1 = \{y_i \mid x_i \notin X_1\}$.*

Proof. Let γ be the clause

$$\gamma = x_{i_1} \vee \cdots \vee x_{i_k} \vee \neg x_{i_{k+1}} \vee \cdots \vee \neg x_{i_m}$$

By definition, γ^{neg} is

$$\gamma^{neg} = \neg y_{i_1} \vee \cdots \vee \neg y_{i_k} \vee \neg x_{i_{k+1}} \vee \cdots \vee \neg x_{i_m}$$

Let X_1 be a model of γ. We will prove that $X_1 \cup Y_1$ is a model of γ^{neg}, where $Y_1 = \{y_i \mid x_i \notin X_1\}$. Since $X_1 \models \gamma$, there exists an index j such that $1 \leq j \leq k$ and $x_j \in X_1$, or $k+1 \leq j \leq m$ and $x_j \notin X_1$. In the first case, $y_j \notin Y_1$, and since γ^{neg} contains $\neg y_j$, we have $X_1 \cup Y_1 \models \gamma^{neg}$. In the other case, $x_j \notin X_1$ and $\neg x_j \models \gamma^{neg}$, thus $X_1 \cup Y_1 \models \gamma^{neg}$.

Let us assume that $X_1 \cup Y_1 \models \gamma^{neg}$, where $Y_1 = \{y_i \mid x_i \notin X_1\}$. We will prove that X_1 is a model of γ. Since $X_1 \cup Y_1 \models \gamma^{neg}$, there must be an index j such that $1 \leq j \leq k$ and $y_j \notin Y_1$, or $k+1 \leq j \leq m$ and $x_j \notin X_1$. In the first case, $x_j \in X_1$ and $x_j \models \gamma$, thus $X_1 \models \gamma$. In the other case, $x_j \notin X_1$ and $\neg x_j \models \gamma$, thus $X_1 \models \gamma$. □

Notice that γ^{neg} is a Horn clause. This theorem is useful since it relates the models of γ, that is a non-Horn clause, with the models of γ^{neg}, which is Horn.

Using this lemma, we can prove the level of compilability of the revision operators defined.

Theorem 28 *Model Checking for Dalal's revision is $\|\!\rightsquigarrow$NP complete, if T and P are Horn formulas.*

Proof. Membership: Determine off-line the value of the minimal distance between a model of T and a model of P, that is, the minimal value of $Diff(I, J)$ for $I \in Mod(T)$ and $J \in Mod(P)$. Given this value $k_{T,P}$, to check whether $M \models T *_D P$, guess a model I of T and verify, in polynomial time, if the difference between I and M is $k_{T,P}$.

Hardness: We prove the hardness using the previous lemma. Let Π be a set of propositional clauses over X, each composed by three literals. Let Π_X

be the set of *all* the clauses of three literals built over X. This set has size polynomial w.r.t. X.

We prove that Π is satisfiable if and only if $M \models T *_D P$, where

$$\begin{aligned} T &= \bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i^{neg} \wedge (\neg X \vee \neg Y) \\ P &= X \wedge Y \\ M &= \{c_i \mid \gamma_i \in \Pi\} \cup X \cup Y \end{aligned}$$

Where C is a set of new variables, appearing nowhere else, one-to-one with Π_X, and γ_i^{neg} is γ_i where each positive occurrence of x_i is replaced by $\neg y_i$. Note that T and P are a Horn theory and a Horn formula, respectively. Furthermore, γ_i is satisfied by $G \subseteq X$ if and only if $G \cup \{y_i \mid x_i \notin X\}$ is a model of γ_i^{neg}.

Consider the pair of models $I = X$ and $J = X \cup Y$. We have that $I \in Mod(T)$, $J \in Mod(P)$, and $|Diff(I, J)| = n$, where n is the number of variables in X. Notice also that $|Diff(L, O)|$, where L and O are models of T and P respectively, cannot be less than n, since T implies $\neg x_i \vee \neg y_i$, while P has $x_i \wedge y_i$: for each i, either x_i or y_i must be given a different truth value by L and O. As a result, $|Diff(L, O)| \geq n$. This means that the minimal number of elements in $Diff$ is exactly n.

Assume Π satisfiable. Let G be a model of Π, and consider the model

$$N = \{c_i \mid \gamma_i \in \Pi\} \cup G \cup \{y_i \mid x_i \notin G\}$$

Since N is a model of T, and $|Diff(N, M)| = n$, the model M is one of the models of the revised knowledge base $T *_D P$.

On the converse, suppose that $M \models T *_D P$. We prove that there exists a model of Π. First of all, $M \models T *_D P$ implies by definition that there exists a model N of T such that $|Diff(N, M)| = n$. Let $X_1 = N \cap X$ and $Y_1 = N \cap Y$. Since T implies $\neg x_i \vee \neg y_i$ and P implies $x_i \wedge y_i$, either x_i or y_i must be in $Diff(N, M)$, for each i. Since the number of elements of $Diff(N, M)$ must be n, it follows that a. $Diff(N, M)$ is composed only of variables in $X \cup Y$; and b. $Y_1 = \{y_i \mid x_i \notin X_1\}$. As a result, $N \cap C = M \cap C$. Thus, for each $\gamma_i \in \Pi$, it must be $c_i \in N$. Since N is a model of T and T implies $c_i \rightarrow \gamma_i^{neg}$, it holds also that $X_1 \cup Y_1$ is a model of any γ_i^{neg} such that $\gamma_i \in \Pi$, which implies that X_1 is a model of any clause in Π. □

Let us now turn to Ginsberg's revision.

Theorem 29 *Model Checking for Ginsberg's revision is* $\rightsquigarrow$P*, it T is a Horn set and P is a Horn formula.*

Proof. Since model checking is P, as proved in [LS96], and P $\subset \rightsquigarrow$P, the claim follows. □

Forbus's revision is as hard as in the general (non-Horn) case. This is exactly what happens for the complexity of the operator.

Theorem 30 *Model Checking for Forbus' revision is* $\|\!\sim$coNP *hard and* $\|\!\sim$NP *hard, in comp*Σ_2^p*, if* T *and* P *are Horn formulas.*

Proof. Membership: The compΣ_2^p membership follows from the complexity (which is Σ_2^p) of the model checking, as proved in [LS96].

Hardness: We prove the $\|\!\sim$coNP hardness using a statement similar to that of Lemma 6. Let Π be a set of clauses over X, each composed by three literals. Let Π_X be the set of all the three-literal clauses over X. We prove that Π is unsatisfiable if and only if $M \models T *_F P$, where

$$\begin{aligned} T &= X \wedge Y \wedge Z \\ P &= \neg Z \vee \Big[(\neg X \vee \neg Y) \wedge \bigwedge_{\gamma_i \in \Pi_X} (c_i \rightarrow \gamma_i^{neg})\Big] \\ M &= \{c_i \mid \gamma_i \in \Pi\} \cup X \cup Y \end{aligned}$$

where Z, Y and C are sets of new variables, of cardinality $|X|+1$, $|X|$ and $|\Pi_X|$, respectively. The clause γ_i is satisfied by $G \subseteq X$ if and only if $G \cup \{y_i \mid x_i \notin G\}$ satisfies γ_i^{neg}.

First, T has exactly one model for each subset of C. As for Dalal's proof, if $I \in Mod(T)$ and $J \in Mod(P)$, then $Diff(I, J) \geq n$, since T implies $x_i \wedge y_i \wedge z_i$, while P implies $\neg x_i \vee \neg y_i \vee \neg z_i$.

Suppose Π satisfiable. Let $G \subseteq X$ be a model of Π. Consider the models

$$\begin{aligned} I &= X \cup Y \cup Z \cup \{c_i \mid \gamma_i \in \Pi\} \\ J &= G \cup \{y_i \mid x_i \notin G\} \cup Z \cup \{c_i \mid \gamma_i \in \Pi\} \end{aligned}$$

We have $|Diff(I, J)| = n$, while $|Diff(I, M)| = n+1$. Furthermore, given any other model N of T, it holds $|Diff(N, M)| = |Diff(N, J)| + 1$, since N can differ from I only for the value of the c_i's, for which J and M agree. As a result, $M \not\models T *_F P$.

Suppose Π unsatisfiable. Let I be the model above. Let $C_1 = \{c_i \mid \gamma_i \in \Pi\}$. The models of P closest to I are of two kinds:

$$\begin{aligned} L &= X \cup Y \cup C_1 \\ N &= X_1 \cup Y_1 \cup Z \cup C_1 \end{aligned}$$

where $X_1 \subseteq X$ and $Y_1 \subseteq Y$. We have $|Diff(I, L)| = n+1$. Furthermore, since Π is unsatisfiable, from Lemma 8 follows that Y_1 cannot be the "opposite" of X_1. Since $\bigwedge(\neg x_i \vee \neg y_i)$ is true, there must be at least an index i such that $x_i \notin X_1$ and $y_i \notin Y_1$. As a result, $|X_1| + |Y_1| < n$, thus $Diff(I, N) > n$. As a result, M is a model of $T *_F P$.

The proof of $\|\!\sim$NP hardness is identical to that of Winslett's operator. □

Theorem 31 *Model Checking for Satoh's revision is* $\|\hookrightarrow$NP *complete, if* T *and* P *are Horn formulas.*

Proof. Membership: follows from the complexity result.

Hardness: We prove hardness using Lemma 6. Let Π be a set of clauses over X, each composed by three literals. Let Π_X be the set of all the clauses of three literals over X. We prove that Π is satisfiable if and only if $M \models T *_S P$, where

$$\begin{aligned} T &= \bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i^{neg} \wedge (\neg X \vee \neg Y) \wedge (X \wedge Y \rightarrow Z) \wedge [(\wedge Z) \rightarrow r] \wedge (r \equiv u) \\ P &= X \wedge Y \wedge \neg Z \wedge \neg r \\ M &= X \cup Y \cup \{u\} \cup \{c_i \mid \gamma_i \in \Pi\} \end{aligned}$$

where C is a set of new variables one-to-one with Π_X, Y, Z are sets of new variables, one-to-one with X, and u, and r are new variables; γ_i^{neg} is γ_i where each positive literal x_i is replaced by $\neg y_i$. Note that $G \subseteq X$ is a model of a clause γ_i if and only if $G \cup \{y_i \mid x_i \notin G\}$ is a model of γ_i^{neg}.

Assume that Π is satisfiable. Let $G \subseteq X$ be a model of Π. This model can be extended to form a model of T:

$$I = G \cup \{y_i \mid x_i \notin G\} \cup Z \cup \{r, u\}$$

The difference between I and M is

$$\mathit{Diff}(I, M) = X \backslash G \cup \{y_i \mid x_i \in G\} \cup Z \cup \{r\}$$

We claim that this difference is minimal, i.e. there is no other pair of interpretations J and N such that $J \in \mathit{Mod}(T)$, $N \in \mathit{Mod}(P)$, and $\mathit{Diff}(J, N) \subset \mathit{Diff}(I, M)$.

Assume that $\mathit{Diff}(J, N) \subseteq \mathit{Diff}(I, M)$. We will prove that $\mathit{Diff}(I, M) \subseteq \mathit{Diff}(J, N)$.

Let $X_1 = J \cap X$ and $Y_1 = J \cap Y$. Since $X \cup Y \subseteq N$, it follows that

$$\mathit{Diff}(J, N) = X \backslash X_1 \cup Y \backslash Y_1 \cup \ldots$$

Since $\mathit{Diff}(J, N) \cap X \subseteq \mathit{Diff}(I, M) \cap X$, it holds $X \backslash X_1 \subseteq X \backslash G$, which implies $G \subseteq X_1$. Since $X_1 \cup Y_1$ is a model of $\neg X \vee \neg Y$, we have $Y_1 \subseteq \{y_i \mid x_i \in G\}$, which in turn implies $Y \backslash \{y_i \mid x_i \in G\} \subseteq Y \backslash Y_1$. Since $Y \backslash \{y_i \mid x_i \in G\} = \mathit{Diff}(J, N) \cap Y$ and $Y \backslash Y_1 = \mathit{Diff}(I, M)$, we have $Y_1 = \{y_i \mid x_i \in G\}$, and thus $X_1 = G$. This implies $J \cap Z = I \cap Z = Z$, which implies $\{r, u\} \subseteq J$. Thus, $J = I$. Now, the model of P which is closer to I is M, thus $\mathit{Diff}(I, M) \subseteq \mathit{Diff}(J, N)$.

Assume Π unsatisfiable. We consider the models I of T that are closest to M, and then prove that $Diff(I, M)$ is never a minimal difference (i.e. there is always a pair of models J and N that are closer).

Let I be a model of T. Assume that $I \cap C = C_2 \neq M \cap C = C_1$. We will prove that there is a model N of P such that $Diff(I, N) \subset Diff(I, M)$. This is simple to prove:

$$Diff(I, M) = Diff(C_1, C_2) \cup Diff(I \cap (X \cup Y \cup Z \cup \{r, u\}), M \cap (X \cup Y \cup Z \cup \{r, u\}))$$

Now, let

$$N = C_2 \cup (M \cap (X \cup Y \cup Z \cup \{r, u\}))$$

which is model of P: this model is obtained from M by replacing C_1 with C_2. The difference between I and N is given by

$$Diff(I, N) = Diff(I \cap (X \cup Y \cup Z \cup \{r, u\}), M \cap (X \cup Y \cup Z \cup \{r, u\})) \subset Diff(I, M)$$

As a result, the difference between I and M is not minimal.

Now consider the other case, i.e. $I \cap C = M \cap C = C_1$. Let $X_1 = I \cap X$ and $Y_1 = I \cap Y$. Since Π is unsatisfiable, $\{\gamma_i^{neg} \mid \gamma_i \in \Pi\}$ cannot be satisfied by a model $X_1 \cup Y_1$ if $Y_1 = \{y_i \mid x_i \notin X_1\}$. As a result, since $T \cup C_1 \models \{\gamma_i^{neg} \mid \gamma_i \in \Pi\}$, the set Y_1 cannot be the "opposite" of X_1, i.e. there must be an i such that $x_i \notin X_1$ and $y_i \notin Y_1$.

When either x_i or y_i are true in a model of T, z_i is forced to be true in that model. In the model I, however, neither x_i nor y_i is true. Thus, z_i can be true or false. Suppose, without loss of generality, that the condition $x_i \notin X_1$ and $y_i \notin Y_1$ holds only for one value of i. If z_i is true, r also must be true (and thus u also must be true). This is the first possible model of T:

$$I_1 = C_1 \cup X_1 \cup Y_1 \cup Z \cup \{r, u\}$$

However, z_i can also be false. In this case r can be true or false, leading to the possible models I_2 and I_3.

$$\begin{aligned} I_2 &= C_1 \cup X_1 \cup Y_1 \cup Z \backslash \{z_i\} \cup \{r, u\} \\ I_3 &= C_1 \cup X_1 \cup Y_1 \cup Z \backslash \{z_i\} \end{aligned}$$

Since $Diff(I_2, M) \subset Diff(I_1, M)$, we do not need to consider I_1 any more (for sure, $Diff(I_1, M)$ is not a minimal difference). The difference between I_2, I_3 and M are

$$\begin{aligned} Diff(I_2, M) &= X \backslash X_1 \cup Y \backslash Y_1 \cup Z \backslash \{z_i\} \cup \{r\} \\ Diff(I_3, M) &= X \backslash X_1 \cup Y \backslash Y_1 \cup Z \backslash \{z_i\} \cup \{u\} \end{aligned}$$

Now, consider

$$N = C_1 \cup X \cup Y$$

N is a model of P, and

$$Diff(I_3, N) = X \backslash X_1 \cup Y \backslash Y_1 \cup Z \backslash \{z_i\}$$

which is strictly contained both in $Diff(I_2, M)$ and $Diff(I_3, M)$. As a result, there is no model I of T such that $Diff(I, M)$ is minimal. Thus, M is not a model of $T *_S P$. □

Theorem 32 *Winslett's and Borgida's revisions are* $\|\!\rightharpoonup$NP *complete, if* T *and* P *are Horn formulas.*

Proof. These two operators are in the same compilability class, since the only difference is the check of consistency of $T \wedge P$, which can be done off-line.

Membership: The problem of model checking is in NP for both these revision, thus it is also in compNP.

Hardness: We prove hardness using Lemma 6. Let Π be a set of clauses, each composed by three literals, and Π_X be the set of all the clauses of three literals over X. We prove that Π is satisfiable if and only if $M \models T *_W P$, where

$$\begin{aligned} T &= \bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i \wedge (\neg X \vee \neg Y) \\ P &= X \equiv Y \\ M &= \{c_i \mid \gamma_i \in \Pi\} \cup X \cup Y \end{aligned}$$

By Lemma 8, Π is satisfied by a model G if and only if T is satisfied by the model $G \cup \{y_i \mid x_i \notin G\}$.

Let I be a model of T. We prove that the models of P that are closest to I have the same value over the c_i's. Let J be a model of P such that $I \cap C = C_1$ and $J \cap C = C_2 \neq C_1$. We have

$$Diff(I, J) = Diff(C_1, C_2) \cup Diff(I \cap (X \cup Y), J \cap (X \cup Y))$$

Let $K = C_1 \cup (J \cap (X \cup Y))$. Clearly, K is a model P. Furthermore,

$$Diff(I, K) = Diff(I \cap (X \cup Y), J \cap (X \cup Y)) \subset Diff(I, J)$$

This proves that the models of T that are closest to I give the same truth value to the c_i's. As a result, M can be a model of $T *_W P$ if and only if there is a model $I \in Mod(P)$ such that $I \cap C = M \cap C$ and M is one of the model of P that are closest to I.

Let us suppose Π satisfiable. Let G be the model of Π, and consider the following model of T.

$$I = \{c_i \mid \gamma_i \in \Pi\} \cup G \cup \{y_i \mid x_i \notin X_1\}$$

We claim that $Diff(I, M)$ is minimal.

$$Diff(I, M) = X \backslash G \cup Y \backslash \{y_i \mid x_i \notin G\}$$

Consider any other model $N \in Mod(P)$, and assume that $Diff(I, N) \subset Diff(I, M)$.

Suppose for example that $x_i \notin Diff(I, N)$ and $x_i \in Diff(I, M)$. Consider the case $x_i \in I$. We have

a. $x_i \in N$
b. $x_i \notin M$
c. $y_i \notin I$

which imply

a. $y_i \in N$
b. $x_i \notin M$

$\Rightarrow$

a. $y_i \in Diff(I, N)$
b. $y_i \notin Diff(I, M)$

which is in contradiction with the hypothesis $Diff(I, N) \subset Diff(I, M)$.

Consider the case $x_i \notin I$. We have

a. $x_i \notin N$
b. $x_i \in M$
c. $y_i \in I$

which imply

a. $y_i \notin N$
b. $x_i \in M$

$\Rightarrow$

a. $y_i \in Diff(I, N)$
b. $y_i \notin Diff(I, M)$

which is in contradiction with the hypothesis $Diff(I, N) \subset Diff(I, M)$.

This part of the proof can be used to prove also that $y_i \notin Diff(I, N)$ together with $y_i \in Diff(I, M)$ is impossible.

As a result, there is no model N of P such that $Diff(I, N) \subset Diff(I, M)$, thus $M \models T *_W P$.

Suppose Π unsatisfiable. As said above, M can be a model of $T *_W P$ only if there is a model I of T such that $I \cap C = \{c_i \mid \gamma_i \in \Pi\} = C_1$, and M is one of its closest models.

Consider a generic model $I \in Mod(P)$.

$$I = C_1 \cup X_1 \cup Y_1$$

Since Π is unsatisfiable, it cannot be $Y_1 = \{y_i \mid x_i \notin X_1\}$. The interpretation I is a model of $\neg x_i \vee \neg y_i$, thus there exists an index i such that $y_i \notin X_1$ and $y_i \notin Y_1$. Now consider the model

$$N = M \backslash \{x_i, y_i\}$$

we have

$$\begin{aligned} \mathit{Diff}(I, M) &= X \backslash X_1 \cup Y \backslash Y_1 \\ \mathit{Diff}(I, N) &= X \backslash (X_1 \cup \{x_i\}) \cup Y \backslash (Y_1 \cup \{y_i\}) \end{aligned}$$

thus $\mathit{Diff}(I, N) \subset \mathit{Diff}(I, M)$: the interpretation M is not a model of $T *_W P$. The hardness of Borgida's revision is proved in a similar manner: setting $T' = T \cup \{w\}$ and $P' = P \wedge \neg w$, we have that $T' *_B P' = (T *_W P) \wedge \neg w$. As a result, Π is satisfiable if and only if $M \models T' *_B P'$. □

Theorem 33 *Model checking of $*_{Web}$ and $*_{Wit}$ is $\rightsquigarrow$*P*, if T and P are Horn formulas.*

Proof. As proved in [LS96], all these revision operators have a polynomial model checking. Since P $\subset \rightsquigarrow$P, the result follows. □

5.1.2 Query Answering

The results of query answering of belief revision are summarized in Table 5.2. The problem analyzed is the query answering: given T, P, and Q, decide whether $T * P \models Q$. The fixed part of the problem is composed by the two formulas T and P, while Q is the fixed part. The table has four columns. The first two contain the results when T and P are generic formulas, while the other two report the compilability of the operators when the involved formulas T, P, and Q are Horn. Another distinction is made on the size of the revision P. Namely, in many cases the revising formula P is much smaller than the current knowledge base T. In such cases the complexity often decreases. The same happens for the compilability results.

Many results of compilability of query answering follow from two simple lemmas.

Lemma 9 *If the problem of model checking for a revision is* compNP*, then the corresponding query answering is* compcoNP*.*

Proof. Since MC is compNP, we have

$$M \models T * P \quad \text{iff} \quad \langle f_1(\langle T, P \rangle), g(f_2(\langle T, P \rangle), M) \rangle \in S$$

	General Case		Horn Case	
	Unbounded	Bounded	Unbounded	Bounded
Ginsberg	$\|\!\sim\!\Pi_2^p$ complete	$\|\!\sim\!\Pi_2^p$ complete	$\|\!\sim\!$coNP complete	$\|\!\sim\!$coNP complete
Winslett/ Borgida	$\|\!\sim\!\Pi_2^p$ complete	$\|\!\sim\!$coNP complete	$\|\!\sim\!$coNP complete	$\sim\!$P
Satoh	$\|\!\sim\!\Pi_2^p$ complete	$\|\!\sim\!$coNP complete	$\|\!\sim\!$coNP complete	$\sim\!$P
Forbus	$\|\!\sim\!\Pi_2^p$ complete	$\|\!\sim\!$coNP complete	$\|\!\sim\!$NP hard	$\sim\!$P
Dalal	$\|\!\sim\!$coNP complete	$\|\!\sim\!$coNP complete	$\|\!\sim\!$coNP complete	$\sim\!$P
Weber	$\|\!\sim\!$coNP complete	$\|\!\sim\!$coNP complete	$\sim\!$P	$\sim\!$P
WIDTIO	$\|\!\sim\!$coNP complete	$\|\!\sim\!$coNP complete	$\sim\!$P	$\sim\!$P

Table 5.2: Compilability of query answering for belief revision operators.

where f_1 and f_2 are polysize functions, g is polynomial and S is an NP language. We can prove that

$$T * P \models Q \quad \text{iff} \quad \langle f_1(\langle T, P\rangle), g'(f_2(\langle T, P\rangle), Q)\rangle \in R \tag{5.1}$$

where g' is a polynomial function and R a coNP language. This would prove that QA is compcoNP. Define

$$\begin{aligned} g'(a, Q) &= \langle a, Q\rangle \\ R &= \{\langle a, \langle b, Q\rangle\rangle \mid \forall M \,.\, M \models Q \text{ or } \langle a, g(b, M)\rangle \notin S\} \end{aligned}$$

The function g' is clearly polynomial. The language R is coNP, since it involves a check of non-membership to the NP language S. We now prove the statement (5.1).

$$\begin{aligned} T * P \models Q \quad &\text{iff} \quad \forall M \,.\, M \models Q \text{ or } M \not\models T * P \\ &\text{iff} \quad \forall M \,.\, M \models Q \text{ or } \langle f_1(\langle T, P\rangle), g(f_2(\langle T, P\rangle), M)\rangle \notin S \\ &\text{iff} \quad \langle f_1(\langle T, P\rangle), \langle f_2(\langle T, P\rangle), Q\rangle\rangle \in R \\ &\text{iff} \quad \langle f_1(\langle T, P\rangle), g'(f_2(\langle T, P\rangle), Q\rangle) \in R \end{aligned}$$

This proves the claim. □

The second lemma allows the determination of hardness of QA, if the hardness of MC is known.

Lemma 10 *If MC is* $\|\!\sim$NP *hard then QA is* $\|\!\sim$coNP *hard.*

Proof. Since MC is $\|\!\sim$NP hard, we can reduce $*\mathsf{sat}$ to it with a nucomp reduction, that is,

$$\Pi \text{ satisfiable} \quad \text{iff} \quad \langle f_1(||\Pi||), g(f_2(||\Pi||), \Pi)\rangle \in MC$$

where f_1, f_2 are polysize functions, while g is polynomial. Now MC is actually

$$\langle\langle T, P\rangle, M\rangle \in \text{MC} \quad \text{iff} \quad M \models T * P$$

It can be reformulated as QA

$$\begin{array}{rll} M \models T * P & \text{iff} & T * P \not\models \neg Form(M) \\ & \text{iff} & \langle\langle T, P\rangle, \neg Form(M)\rangle \notin \text{QA} \end{array}$$

As a result,

$$\Pi \text{ is satisfiable} \quad \text{iff} \quad \langle f_1(||\Pi||), \neg Form(g(f_2(||\Pi||), \Pi))\rangle \notin \text{QA}$$

thus

$$\Pi \text{ is unsatisfiable} \quad \text{iff} \quad \langle f_1(||\Pi||), g'(f_2(||\Pi||), \Pi)\rangle \in \text{QA}$$

where $g'(a, b) = \neg Form(g(a, b))$ is clearly polynomial. As a result, the query answering problem for $*$ is $\|\!\sim$coNP hard. □

With a similar proof, we can conclude that if MC is $\|\!\sim$coNP hard, then QA is $\|\!\sim$NP hard, and if MC is $\|\!\sim\Sigma_2^p$ hard, QA is $\|\!\sim\Pi_2^p$ hard.

General Case

Theorem 34 *Query answering for Ginsberg's revision operator is* $\|\!\sim\Pi_2^p$ *complete.*

Proof. Membership follows from the fact that this operator is Π_2^p. Hardness is proved by reduction from the $\|\!\sim\Pi_2^p$ hard problem $*\forall\exists$QBF. This an adaptation of the proof of Π_2^p hardness given in [EG92].

Let X and Y be two sets of variables. We prove that given a set Π of clauses, each composed by three literals over $X \cup Y$, there exists an assignment

of the X such that for each assignment of the Y the set Π is false if and only if $T *_G P \not\models Q$, where

$$\begin{array}{rcl} T & = & X \cup Y \cup Z \cup C \cup D \cup \{r\} \\ P & = & (C \not\equiv D) \wedge (X \not\equiv Y) \wedge \Big[(\neg r \wedge \neg Y) \vee (\bigwedge_{\gamma_i \in \Pi_{X \cup Y}} \{c_i \rightarrow \gamma_i)\Big] \\ Q & = & \Big[\bigwedge\{c_i \mid \gamma_i \in \Pi\} \wedge \bigwedge\{\neg c_i \mid \gamma_i \notin \Pi\}\Big] \rightarrow r \end{array}$$

where $\Pi_{X \cup Y}$ is the set of all the clauses of three literals over the alphabet $X \cup Y$, and $|\Pi_{X \cup Y}| = m$. These formulas contains new variables: the variables C and D are m, while the variables of Z are n. Note that T has only one model, namely

$$I = X \cup Y \cup Z \cup C \cup D \cup \{r\}$$

thus $T *_G P = \mathit{Form}(I) *_W P$, which can be rewritten as

$$\mathit{Mod}(T *_G P) = \{J \in \mathit{Mod}(P) \mid \nexists K \in \mathit{Mod}(P) \,.\, \mathit{Diff}(I, K) \subset \mathit{Diff}(I, J)\}$$

Let

$$\begin{array}{rcl} C_1 & = & \{c_i \mid \gamma_i \in \Pi\} \\ D_1 & = & \{d_i \mid \gamma_i \notin \Pi\} \end{array}$$

If $J \cap C = C_1$ then J satisfies $\bigwedge_{\gamma_i \in \Pi_{X \cup Y}} c_i \rightarrow \gamma_i$ if and only if J satisfies Π.

Assume $\exists X \forall Y . \neg \Pi$. Let X_1 be the assignment of the x_i such that Π is false, and $Z_1 = \{z_i \mid x_i \notin X_1\}$. Consider the following model of P:

$$J = C_1 \cup D_1 \cup X_1 \cup Z_1$$

We prove that this is a model of $T *_W P$. This is not a model of Q, since it satisfies $\bigwedge\{c_i \mid \gamma_i \in \Pi\}$ and $\bigwedge\{\neg c_i \mid \gamma_i \notin \Pi\}$, but it does not contain r. As a result, $T *_W P \not\models Q$.

Let K be any other model of P: we prove that $\mathit{Diff}(I, K) \not\subset \mathit{Diff}(I, J)$. If K differs from J for some c_i or some x_i, then $\mathit{Diff}(I, K)$ is incomparable from $\mathit{Diff}(I, J)$. Indeed, if for example J contains c_i while K do not, then (since J and K must both satisfy $c_i \neq d_i$), then K contains d_i and J do not. Thus, $\mathit{Diff}(I, J)$ contains d_i but not c_i, while $\mathit{Diff}(I, K)$ contains c_i but not d_i.

If K has the same value of J w.r.t. the X and C, then the subformula $\bigwedge_{\gamma_i \in \Pi_{X \cup Y}} c_i \rightarrow \gamma_i$ is false for any Y. Thus K must satisfy $\neg r \wedge \neg Y$, and thus it is equal to J. As a result, there is no model K of P that is closer to I more than J. This implies that J is a model of $T *_W P$. Since J does not satisfy Q, this implies also $T *_W P \not\models Q$.

Assume $\forall X \exists Y.\Pi$. Let J be the generic model of $T *_W P$. We prove that it satisfies Q. If J has an evaluation of the C that differs from C_1, then Q is satisfied. On the converse, assume that J evaluates the C according to C_1, that is, $J \cap C = C_1$ and $J \cap D = D_1$. Let $X_1 = J \cap X$ and $Z_1 = J \cap Z$. The formula P has two kinds of models: those satisfying $\neg r \wedge \neg Y$ and those satisfying $\bigwedge_{\gamma_i \in \Pi_{X \cup Y}} c_i \rightarrow \gamma_i$. Let us assume that J satisfies the first one. Thus J is a model like

$$J = C_1 \cup D_1 \cup X_1 \cup Z_1$$

Since, for each truth evaluation of the X the formula Π is satisfiable, the model

$$K = C_1 \cup D_1 \cup X_1 \cup Z_1 \cup Y_1 \cup \{r\}$$

is a model of P, where Y_1 is the truth evaluation of Y that, together with X_1, satisfies Π. This holds because the second subformula of P is equivalent to Π for models with $K \cap C = C_1$.

Since $Diff(I, K) \subset Diff(I, J)$, the model J is not a model of $T *_W P$. This proves that the only models of P that are also in $T *_W P$ are models satisfying $\bigwedge_{\gamma_i \in \Pi_{X \cup Y}} c_i \rightarrow \gamma_i$. This formula does not contain r. As a result, if a model J does not contains r it is not one of the closest models to I, since the model $J \cup \{r\}$ is closer: indeed $Diff(I, J \cup \{r\}) = Diff(I, J) \backslash \{r\}$. As a result, the closest models to I contain r. These models satisfy Q. As a result, $T *_W P \models Q$. □

Theorem 35 *Query answering for Winslett's, Borgida's, Satoh's, and Forbus' revision operators is* $\|\!\sim \Pi_2^p$ *complete.*

Proof. Membership follows from Lemma 9, while hardness follows from Lemma 10, and from the fact that all these revision operators have a $\|\!\sim \Sigma_2^p$ complete query answering. □

Theorem 36 *Dalal's and Weber's query answering is* $\|\!\sim$coNP *complete.*

Proof. Follows from Lemma 9 and Lemma 10, and from the fact the model checking is $\|\!\sim$NP complete. □

Theorem 37 *The problem of query answering for WIDTIO revision is* $\|\!\sim$coNP *complete.*

Proof. Membership follows from Lemma 9: since model checking is $\sim$P, it follows that it is also $\|\!\sim$NP, thus query answering is in $\|\!\sim$coNP.

Hardness is easily proved from $*\mathsf{unsat}$. Indeed, y is unsatisfiable if and only if $\mathsf{true} *_{Wit} \mathsf{true} \models \neg y$. □

Bounded Case

The complexity of Ginsberg's revision in the case in which P is bounded by a constant is determined by the fact that limiting the size of P does not reduce the compilability of the problem.

The membership of all the other revision operators to $\|\!\sim$coNP is due to Lemma 9.

Theorem 38 *The problem of query answering for all the revision operators introduced but Ginsberg's one is* $\|\!\sim$coNP*.*

Proof. Since the problem of model checking is in $\sim$P, it is also in compNP. From Lemma 9 it follows that the problem of query answering is in $\|\!\sim$coNP. □

The result of hardness is proved in the next theorem.

Theorem 39 *The problem of query answering for all the revision operators introduced but Ginsberg's one is* $\|\!\sim$coNP *complete.*

Proof. We prove the claim by giving a nucomp reduction from the problem $*\mathsf{unsat}$ to any of the revision operators $*$ introduced. Define

$$\begin{aligned} f_1(x) &= \langle \mathsf{true}, \mathsf{true} \rangle \\ f_2(x) &= \epsilon \\ g(a, y) &= \neg y \end{aligned}$$

Given a propositional formula Π, we have that Π is unsatisfiable if and only if $\mathsf{true} * \mathsf{true} \models \neg\Pi$, which is indeed equivalent to say that $\langle\langle \mathsf{true}, \mathsf{true} \rangle, \neg\Pi\rangle$ is a member of the query answering problem. This holds for any of the revision operators introduced, since $\mathsf{true}*\mathsf{true}$ entails a formula if and only if the formula is valid. □

Horn Case

The first result is about Ginsberg's revision. The membership to the class is proved by Lemma 9, while the hardness is proved by reduction.

Theorem 40 *Query answering for Ginsberg's revision is* $\|\!\sim$coNP *complete.*

Proof. Membership follows from the fact that query answering is coNP, as proved in [EG92].

We prove hardness by reduction from the problem of query answering in Satoh's revision. Namely, we will prove that, if $Var(Q) \subseteq X$, then

$$T *_S P \models Q \quad \text{iff} \quad T' *_G P' \models Q$$

where $T' = \{w_1, \ldots, w_n\}$, and

$$P' = T[X/Y] \wedge P \wedge \bigwedge_{1 \leq i \leq n} w_i \rightarrow (x_i \equiv y_i)$$

where $W = \{w_i\}$ and $Y = \{y_i\}$ are sets of new variables, one-to-one with X.

We prove first that if $T *_S P \models Q$ then $T' *_G P' \models Q$. Let $K \in \mathit{Mod}(T' *_G P')$. We will prove that $K \models Q$.

By definition, there must be a $T'' \subseteq T'$ that is maximally consistent with P', and such that $K \in \mathit{Mod}(T'' \cup \{P\})$. Let

$$\begin{aligned} I &= \{x_i \mid y_i \in K\} \\ J &= K \cap X \end{aligned}$$

Clearly, I is a model of T and J is a model of P. Moreover, $x_i \in \mathit{Diff}(I, J)$ implies that y_i and x_i have different truth value in K, thus w_i is inconsistent with T''. As a result, $T'' \subseteq \{w_i \mid x_i \notin \mathit{Diff}(I, J)\}$. We prove that $\mathit{Diff}(I, J)$ is minimal. Suppose that it is not, and let I' and J' be two models of T and P such that $\mathit{Diff}(I', J') \subset \mathit{Diff}(I, J)$. Now, let

$$T''' = \{w_i \mid x_i \notin \mathit{Diff}(I', J')\}$$

We have that $T'' \subset T'''$. Furthermore, $\{y_i \mid x_i \in I'\} \cup J \cup T'''$ is a model of $T''' \cup \{P\}$, thus contradicting the hypothesis of T'' being maximally consistent with P.

Now, we have proved that $\mathit{Diff}(I, J)$ is minimal. As a result, J has the property that there exists a $I \in \mathit{Mod}(T)$ such that $\mathit{Diff}(I, J)$ is minimal. Thus $J \in \mathit{Mod}(T *_S P)$, and since $T *_S P \models Q$, we have $J \models Q$. Since Q contains only variables in X, and $J = K \cap X$, we have $K \models Q$. This proves that each model K of $T' *_G P'$ is also a model of Q, thus proving that $T' *_G P' \models Q$.

We prove now the converse, that is, if $T' *_G P' \models Q$ then $T *_S P \models Q$. Let J be a model of $T *_S P$. We prove that J is also a model of Q.

By definition, there exists a model I of T such that $\mathit{Diff}(I, J)$ is minimal. Let

$$T'' = \{w_i \mid x_i \notin \mathit{Diff}(I, J)\}$$

This set is consistent with P, and is also maximal (this is a consequence of the fact that $\mathit{Diff}(I, J)$ is minimal). Moreover, $K = \{y_i \mid x_i \in I\} \cup J \cup T''$ is a model of $T'' \cup P$, thus is also a model of $T' *_G P'$, and thus $K \models Q$. Since Q contains only variables in X, and $J = K \cap X$, we have $J \models Q$. □

The complexity of Dalal's, Satoh's, Winslett's, and Borgida's operators is reported in the next theorem.

Theorem 41 *Query answering for Dalal's, Satoh's, Winslett's, and Borgida's revisions is* $\|\!\sim$coNP *hard, in* compcoNP.

Proof. The compilability of QA of all these revisions follows from the compilability of MC and Lemma 9 and Lemma 10. □

Theorem 42 *Query answering for* $*_F$ *is* $\|\!\sim$NP *hard and* $\|\!\sim$coNP *hard, in* $comp\Pi_2^p$.

Proof. The claim can be proved from the compilability of MC, and a proof similar to the two Lemmas 9,10 at the beginning of this section. □

Theorem 43 *Query answering for Weber's revision is* $\rightsquigarrow$P.

Proof. First, compute Ω. Now, as proved in [CDLS95], $T *_{Web} P$ can be expressed as

$$T *_\Omega P \quad \text{iff} \quad T[S] \wedge P \models Q$$

where $S = \{x_i/y_i \mid x_i \in \Omega\}$, and the y_i's are new variables appearing nowhere else. Now, $T[S] \wedge P$ and Q are Horn formulas. Thus, deciding if the former implies the latter is a polynomial task. □

Theorem 44 *Query answering for WIDTIO revision is* $\rightsquigarrow$P.

Proof. By definition $T *_{Wit} P = (\wedge T') \wedge P$, where T' is a subset of T. As a result, $T *_{Wit} P \models Q$ if and only if $(\wedge T') \wedge P \models Q$ and the latter is polynomial since the formulas involved are Horn. □

5.2 Circumscription

Circumscribing a formula T is a way to implicitly represent negative information. The formula $CIRC(T; X, Y, Z)$ is T itself plus the hypothesis that atoms whose value is unknown are false. The problem mainly studied about circumscription is the entailment, that is, deciding whether a fact is implied by $CIRC(T; X, Y, Z)$. Formally, given two formulas T and Q, the problem is to decide whether $CIRC(T; X, Y, Z) \models Q$. An alternative way of representing knowledge is that of model checking, that is, assuming that a formula is used to represent a set of models, the interesting question is to decide whether $M \models CIRC(T; X, Y, Z)$. The problem of query answering (or entailment) is Π_2^p complete, while model checking is coNP complete.

Both these problem are thus intractable. However, a single knowledge base T must be in general queried many times with respect to different queries

$Q_1, \ldots, Q_m$ or $M_1, \ldots, M_m$. Thus, it makes sense to compile T once (even if this phase is long) if this allows the solution of the problems of query answering or model checking more efficiently. In this section we show that this is impossible, since the problem of query answering is $\|\!\rightarrow\Pi_2^p$ complete and model checking is $\|\!\rightarrow$coNP complete. Compiling circumscription in exponential time is instead possible, but this way of preprocessing is not practical.

Theorem 45 *The problem $M \models CIRC(T; X, Y, Z)$ is $\langle \text{EXPTIME}, \text{P} \rangle$, where T, X, Y, and Z are the fixed part and M is the varying part.*

Proof. Let $|Var(T)| = n$. Assume that M contains only variables that are also in T. The number of possible models over a set of n variables is exactly 2^n. As a result, it is possible to write all the interpretation over such alphabet in an exponential-size table.

This table has an element for each interpretation of the alphabet. Each element is a boolean. The first interpretation of the table is the empty interpretation, that assigning false to all the variables. The ordering of the table is by the Y first. Between interpretations with the same Y, the ordering is on the basis of the number of X present in the interpretation: first the interpretation with 0 atoms in X, then all the interpretations with 1 atom, then all the interpretations with 2 atoms and so on. Between the interpretations with the same number of atoms in X we adopt a lexicographic ordering. Finally, the ordering on the Z is lexicographic. Note that, given an interpretation, finding the corresponding element of the table is a polynomial problem.

The algorithm is a cycle over the table. Start by assigning true to all the elements of the table. At each step an interpretation I is analyzed. If the boolean value of the table is false, or the value is true but the interpretation is a model of T, assign false to all the interpretations obtained by adding another atom of X to I, and a (possibly empty) subset of Z. If the value is true and the interpretation is not a model of T, do nothing.

Each step requires exponential time, since the hard part is the assignment of false to the interpretations obtained by adding a subset of Z to I. Since there are at most 2^n such interpretations, the step is exponential. As a result, the whole procedure is exponential, since the number of interpretations is exponential.

Finding the element corresponding to an interpretation table is a polynomial problem. Given a model M, this is a minimal model of T if and only if it is a model of T and its value in the table is true. Indeed, the value of an interpretation I in the table is true if and only if there is no model I' of T such that $I' \cap Y = I \cap Y$ and $I' \cap X \subset I \cap X$. □

Query answering is harder than model checking. The problem is that $CIRC(T; X, Y, Z) \models Q$ has $\models Q$ as a subcase. In order to obtain a polynomial

time post-processing algorithm, the preprocessing phase should find all the tautologies of the propositional calculus in exponential time.

Theorem 46 *The problem* $CIRC(T; X, Y, Z) \models Q$ *is* $\langle \text{EXPTIME}, \text{NP} \rangle$*, if* T*,* X*,* Y*, and* Z *are the fixed part and* Q *the varying part.*

Proof. The preprocessing algorithm is the same. Given this table, deciding whether $CIRC(T; X, Y, Z) \not\models Q$ is equivalent to guess an interpretation which is a model of $CIRC(T; X, Y, Z)$ but not a model of Q. Using the table, both these checking can be made in polynomial time, thus the problem is coNP complete. □

Note that determining $\models Q$ is already coNP complete. Thus, this compilation decreases the cost of circumscription to the cost of simple (non-minimal) inference. Let us consider the subcase of circumscription in which $Y = \emptyset$. In such cases, the input is composed by T and X only, since Z is the set of variables not in X. Under the assumption that the input is only T and X, we can prove that the problem of query answering cannot be compiled any further with a finite preprocessing.

Theorem 47 *Let* FFINITE *be the set of functions whose output is a finite data structure. The problem of deciding whether* $CIRC(T; X, \emptyset, Z) \models Q$*, given* T *and* X *as fixed part and* Q *as varying part, is not in* $\langle \text{FFINITE}, \text{P} \rangle$*, unless* P=NP.

Proof. Let us assume that the problem is in $\langle \text{FFINITE}, \text{P} \rangle$. Thus there exists a function f and a polynomial language S such that

$$CIRC(T; X, \emptyset, Z) \models Q \quad \text{iff} \quad \langle f(\langle T, X \rangle), Q \rangle \in S$$

The Z is not part of the input, as it is implicit in T and X. Now, consider the problem of deciding whether Q is valid. This is equivalent to $CIRC(\mathsf{true}; \emptyset, \emptyset, Z) \models Q$, thus we have:

$$\begin{array}{lll} \models Q & \text{iff} & CIRC(\mathsf{true}; \emptyset, \emptyset, Z) \models Q \\ & \text{iff} & \langle f(\langle \mathsf{true}, \emptyset \rangle), Q \rangle \in S \end{array}$$

But $f(\langle \mathsf{true}, \emptyset \rangle)$ is a finite data structure. Since true and $\emptyset$ are constant, $f(\langle \mathsf{true}, \emptyset \rangle)$ is also a constant string. As a result, deciding whether Q is valid is reducible to the problem of membership to a polynomial language. As a result, P=coNP, which implies P=NP. □

The EXPTIME compilation is not practical, since an exponential-sized result of the compilation phase is in general not affordable. We show that a polysize compilation does not decrease the complexity. This is proved by showing that model checking is $\|\!\rightarrow$coNP complete, and query answering is $\|\!\rightarrow \Pi_2^p$ complete.

Theorem 48 *Deciding $M \models CIRC(T; X, Y, Z)$ is $\|\!\rightsquigarrow$coNP complete, if T is the fixed part and M is the varying part.*

Proof. Membership follows from the fact that the problem is coNP even without compilation. Hardness is proved by reduction from the $\|\!\rightsquigarrow$coNP complete problem $*$3unsat. Let Π be a set of clauses of three literals over the alphabet X. We prove that Π is unsatisfiable if and only if $M \models CIRC(T; C \cup D \cup X \cup \{r\}, \emptyset, \emptyset)$, where

$$\begin{aligned} T &= (C \not\equiv D) \wedge \left[(X \wedge r) \vee \bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i\right] \\ M &= \{c_i \mid \gamma_i \in \Pi\} \cup \{d_i \mid \gamma_i \notin \Pi\} \cup X \cup \{r\} \end{aligned}$$

where Π_X is the set of *all* the clauses of three literals over the variables X, C and D are two sets of new variables, one-to-one with the clauses of Π_X, and r is a new variable.

Let us assume Π satisfiable. Let X_1 be the model of Π. Let us consider the interpretation

$$M' = \{c_i \mid \gamma_i \in \Pi\} \cup \{d_i \mid \gamma_i \notin \Pi\} \cup X_1$$

We prove that a. M' is a model of T and b. $M' \subset M$. The second is easy to prove, since $M \cap (C \cup D)$ and $M' \cap (C \cup D)$ are identical, while M' has X_1 while M contains all the X plus r.

Let us prove $M' \in Mod(T)$. Since M' has c_i if and only if $\gamma_i \in \Pi$, the formula $\bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i$ is satisfied by M' if and only if $\bigwedge_{\gamma_i \in \Pi} \gamma_i$ is implied by M'. This formula is equivalent to Π, which has M' as a model. As a result, the model M is not a model of $CIRC(T; C \cup D \cup X \cup \{r\}, \emptyset, \emptyset)$.

Assume that Π is unsatisfiable. The formula T has two kinds of models:

1. Models with $I \cap C \neq M \cap C$.
2. Models with $J \cap C = M \cap C$.

Assume $I \cap C \subset M \cap C$. Let for example $c_i \notin I \cap C$ but $c_i \in M \cap C$. Since both I and M must satisfy $c_i \not\equiv d_i$ we have $d_i \in I \cap C$ and $d_i \notin M \cap C$. As a result, $I \not\subset M$.

Let J be a model such that $J \cap C = M \cap C$. Such a model cannot satisfy $\bigwedge_{\gamma_i \in \Pi_X} c_i \rightarrow \gamma_i$, since this formula can be simplified to Π for models with $J \cap C = \{c_i \mid \gamma_i \in \Pi\}$. As a result, J must satisfy $X \wedge r$, and thus it coincides with M. As a result, there is no model $M' \in Mod(T)$ such that $M' \subset M$, and thus $M \in Mod(CIRC(T; C \cup D \cup X \cup \{r\}, \emptyset, \emptyset))$. □

The problem of query answering cannot be make simpler by a preprocessing, as the next theorem proves.

Theorem 49 *Deciding* $CIRC(T; X, Y, Z) \models Q$ *is* $\|\!\rightarrow\Pi_2^p$ *complete.*

Proof. We reduce Satoh's revision to circumscription using a nucomp reduction. Since query answering in Satoh's revision is $\|\!\rightarrow\Pi_2^p$ hard, the same holds for circumscription.

Namely, we prove that, given three formulas T, P, and Q built on the same alphabet X, it holds

$$T *_S P \models Q \quad \text{iff} \quad CIRC(T_S; W, \emptyset, X \cup Y) \models Q$$

where $T_s = T[X/Y] \wedge P \wedge (W \rightarrow (X \equiv Y))$. The sets Y and W are sets of new variables, one-to-one with X. The notation $T[X/Y]$ is used to denote the formula obtained by replacing each x_i with the corresponding y_i in T.

This statement has been proved in [LS97]. The first step of the proof is to show that $CIRC(T_S; W, \emptyset, X \cup Y) \models Q$ implies that $T *_S P \models Q$. Assume that $CIRC(T_S; W, \emptyset, X \cup Y) \models Q$ and $T *_S P \not\models Q$. Thus, there exists a model M_X of $T *_S P$ such that $M_X \not\models Q$. Let M'_X be a model of T such that $Diff(M'_X, M_X) \in K_{T,P}$. We define $M_Y = \{y_i | x_i \in M'_X\}$ and

$$M_W = \{w_i | ((x_i \in M_X) \text{ and } (y_i \notin M_Y)) \text{ or } ((x_i \notin M_X) \text{ and } (y_i \in M_Y))\}$$

Let $M = M_X \cup M_Y \cup M_W$. Obviously, it holds that $M \models T_S$ and $M \not\models Q$. If M is a minimal model the thesis follows, so assume that there exists a model N of T_S such that N is less than M. Since N is a model of T_S and it cannot contain more literals of W than M, we have that $N_W = (N \cap W) \subset M_W$. Hence, the difference between $N_X = N \cap X$ and $N_Y = N \cap Y$ is smaller than the difference between M_X and M_Y. Thus, M_X is not one of the models of P closest to the models of T. As a consequence, M_X is not a model of $T *_S A$ and contradiction arises.

We now show that $T *_S P \models Q$ implies that $CIRC(T_S; W, \emptyset, X \cup Y) \models Q$. Assume that $T *_S P \models Q$ and $CIRC(T_S; W, \emptyset, X \cup Y) \not\models Q$. Thus, there exists a model M of $CIRC(T_S; W, \emptyset, X \cup Y)$ such that $M \not\models Q$. Let $M_X = M \cap X$, we show that $M_X \models T *_S P$. It immediately follows that $M_X \models P$, if M_X is one of the models of P closer to models of T the thesis follows, so assume to the contrary that there exists a $N_X \subseteq X$, different from M_X, such that $N_X \models P$ and the distance of M_X from the closest model of T is a strict superset of the distance of N_X from its closest model of T. Let N'_X be a model of T such that $Diff(N'_X, N_X) \in K_{T,P}$. Let $N_Y = \{y_i | x_i \in N'_X\}$,

$$N_W = \{w_i | ((x_i \in N_X) \text{ and } (y_i \notin N_Y)) \text{ or } ((x_i \notin N_X) \text{ and } (y_i \in N_Y))\}$$

and $N = N_X \cup N_Y \cup N_W$. Obviously N is a model of $T[X/Y] \wedge A \wedge (\neg W \equiv (X \equiv Y))$ moreover, N has less minimizing atoms than M. Hence, M is not a minimal model of $T[X/Y] \wedge A \wedge (\neg W \equiv (X \equiv Y))$, hence contradiction arises.

As a result, since Satoh's revision is $\|\!\sim\!\Pi_2^p$ hard and can be reduced to circumscription, the latter is $\|\!\sim\!\Pi_2^p$ hard too. □

5.3 Compilability in Reasoning about Actions

A domain description in $\mathcal{A}$ is a way of representing knowledge about a scenario involving actions. The information contained in such a knowledge base is extracted by asking whether a fact is true or not, that is, given a value proposition V, deciding whether $D \models V$. The problem of inference is coNP complete, thus intractable. Since a domain description is often queried many times w.r.t. many different value propositions, it makes sense to compile D once, if this allows the solving of the problem of inference in polynomial time.

In the general case, such a compilation is impossible. Indeed, we prove that the problem $D \models V$, when D is the fixed part and V is the varying part, is $\|\!\sim$coNP complete.

We prove this claim by giving a reduction from the problem $*$3unsat to the problem of entailment in $\mathcal{A}$.

Theorem 50 *The problem of deciding whether $D \models V$ in $\mathcal{A}$ is $\|\!\sim$coNP complete, if D is the fixed part of the input and V is the varying part.*

Proof. We prove the claim by reducing the problem of unsatisfiability of a set of clauses (each composed by three literals) to the problem of entailment in $\mathcal{A}$. Let Π be a set of clauses, each composed by three literals. We prove that Π is unsatisfiable if and only if $D \models V$, where

$$\begin{aligned} D &= \{ \text{ initially } \neg F\} \cup \{A_i \text{ causes } F \text{ if } \neg L_1, \neg L_2, \neg L_3 \mid \\ &\qquad\qquad \text{for each clause } \gamma_i = L_1 \vee L_2 \vee L_3 \in \Pi_X\} \\ V &= F \text{ after } A_1; \ldots; A_m \end{aligned}$$

where Π_X is the set of all the clauses of three literals over the alphabet X, and $\{A_i\}$ is a set of actions, one-to-one with the clauses of Π_X. The sequence of actions in the value proposition is composed exactly by the A_i with are in correspondence with the clauses γ_i in Π.

Note that D is build over Π_X, which depends only on the number of variables in X. The only dependence on the specific set of clauses is in the value proposition V.

Let us now prove the claim. Assume that Π is satisfiable. Let I be a model of Π. Consider the state $\sigma_0 = I$. Since the only value proposition in D is initially $\neg F$, this is a possible initial state, that is, (σ_0, Ψ_D) is a model of D. Note that there is no effect proposition that change the value of the fluents

L_i. Since Π is satisfied, for each clause $\gamma_i = L_1 \vee L_2 \vee L_3$ at least a literal is true. As a result, the action A_i does not change the state when executed, since at least one precondition of the action A_i causes F if $\neg L_1, \neg L_2, \neg L_3$ is false. This holds for each action A_i such that $\gamma_i \in \Pi$, since all the clauses in Π are satisfied by I. As a result, at the end of the sequence $A_1; \ldots; A_m$ the value of F is not modified, since all the clauses corresponding to the A_i in this sequence are in Π.

Let us assume that the set of clauses Π is unsatisfiable. Let (σ_0, Ψ_D) be a model of D. Since Π is unsatisfiable, for each interpretation over $\{L_i\}$ at least a clause must be false in that interpretation. Let $\gamma_i = L_1 \vee L_2 \vee L_3$ be the clause that is falsified by $I = \sigma_0$. Since all the literals it contains are false in I, it follows that all the preconditions of A_i causes F if $\neg L_1, \neg L_2, \neg L_3$ are true, starting from the initial state σ_0. As a result, the fluent F is true at the end of the sequence. This proof does not depends on the specific initial state chosen. As a result, the fluent F is true after the sequence for each possible initial state, thus V is implied by D.

This reduction is "almost" a nucomp reduction. The only difference is that D depends on the number of variables in Π and not on the size of Π. This problem can be easily overcame by assuming that Π is built over an alphabet with a number of elements equal to the size of the formula Π. This is always possible, since there are no constraints imposing that Π must use all the variables of the alphabet. □

This theorem proves that, in the general case, the problem of entailment in $\mathcal{A}$ cannot be compiled getting a polynomial compiled structure. In such cases, we have a choice: restrict the language. For example, fixing the length of the sequence in V allows such compilation.

Theorem 51 *Entailment in $\mathcal{A}$ is $\langle \Delta_2^p, \mathrm{P} \rangle$ if the size of V is fixed.*

Proof. The number of possible value propositions of size less or equal than a constant k is a polynomial. As a result, the preprocessing can solve the problem for each of these value propositions, and storing the results in a table. The on-line algorithm just look up the table to check whether the specific value proposition of the instance is entailed or not. □

Note that the case of fixed-size value proposition is not polynomial without preprocessing.

5.4 Problems on Graphs

In this section we analyze the compilability properties of some NP complete problems. Most of these problems come from graph theory. We also include

the problem of conjunctive query, which is a very relevant one in database theory.

5.4.1 Steiner Tree

Formally, the problem is defined as: given a graph G whose edges are labeled with integers, a number k and a set of nodes N', decide if G has a subtree of weight less or equal than k that contains all the nodes of N'.

This is a classical problem of graph theory, and is used to formalize problems of deciding if a set of points can be connected without exceeding a certain cost. For example, in network design, one wants to know if a set of nodes in a network will remain connected with a given probability (given the probability of failure of the edges). The graph and the weight of the edges are in general fixed (they depends on the structure of the network), while one may ask about the reliability of the network w.r.t. many possible sets of nodes. Thus, the variable part is the set of nodes that the tree must connect.

5.4.2 Network Flow

The problem is: given a graph and a collection of disjoint source-sink pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, decide if there exists a set of k disjoint paths, each from s_i to t_i.

This problem deals with the ability of a network to support a given traffic. The set of source-sink pairs represents the set of nodes that want to communicate, and thus they are the varying part of the problem.

5.4.3 Required Pairs

The definition of this problem is a bit complex: given a graph G, a set of pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, two nodes s and t and an integer k, decide if there exist a set of k paths from s to t such that for any pair (s_i, t_i) there is a path containing both s_i and t_i.

This problem comes from program testing. To certify a program, one must test all execution sequences in the program. Even for small programs, the number of them can be extremely large. A compromise is to test only a subset of the execution sequences. Asking that a pair of nodes are together in at least one path is equivalent to ask for a test set that ensure that two segments of code interact in the correct way (see [NH79] for a more detailed explanation). A given program must be tested many times, thus it would be useful to compile its structure in order to speed up the search of test sets.

5.4.4 Hamiltonian Cycle

This is one of the "classic" NP complete problems: given a graph, decide if there exists a circuit that contains each node exactly once Here we consider the variant in which the circuit must contain exactly a subset of nodes $N' \subseteq N$.

The problem Hamiltonian Cycle is a formalization of problems in which one must visit a set of point, minimizing the total distance traveled. One can hardly see a fixed and a varying part in this problem. However, the variant in which only a subset of nodes must be reached is a typical example of problem with a fixed and a varying part: the positions of the points (and thus the underlying graph) is fixed, while the set of points that must visited may change from time to time (think for example to the traveling salesman who has to visit a subset of the cities, or a postman who has to deliver mail only to a subset of people that live in a city). If this is the case, we can afford a long preprocessing time on the graph, if this make easier the finding of the Hamiltonian circuit.

5.4.5 Conjunctive Query

Given a conjunctive query in the relational calculus, and a set of relations, decide if the query is true.

This problem is the decision version of a problem that occurs in relational database systems: given a certain query, find the tuples that satisfies it. In practical settings, there is a set of typical queries that occur often, while the specific database is not known in advance. In this case it makes sense to compile the query in order to allow a fast answering.

Note that Theorem 52, proving the incompilability of this problem, is not in contrast to a result well known to databases researchers, that proves that Conjunctive Query can be solved in logarithmic space (thus in polynomial time) if the query is constant. This is a good opportunity to remark the difference between "constant" and "fixed". When we say that part of the input data is constant, we mean that this part will be small, or that we are not interested in analyzing the complexity when the size of it becomes large. Instead, when part of the input is fixed, we mean that it can be arbitrarily large but we can afford a long time of processing on it, because either it is known in advance, or there will be many input instances with the same fixed part.

5.4.6 Subgraph Isomorphism

This is the problem of decide whether, given two graphs G and H, there exists a subgraph of G that is isomorphic to H.

This problem is not compilable, either if the fixed part is the "big" graph, or the "small" one. We conjecture that the graph isomorphism problem (given two graphs, decide if they are isomorphic) is compilable to P.

Theorem 52 *The problems Steiner Tree, Network Flow, Required Pairs, Subset Hamiltonian Cycle, Conjunctive Query, and Subgraph Isomorphism are* $\|\not\rightarrow$NP *hard.*

This theorem shows that none of these problems can be compiled to P using a polysize compilation function, and this holds even if the preprocessing phase has access to the size of the varying part.

Chapter 6

Compact Representation

The problem of compact representation is the following: given a function f from boolean formulas to boolean formulas, is there a formula equivalent to $f(T)$ whose size is polynomial in the size of T? This problem is strictly related to that of compilability. Indeed, we prove that results of non-compilability can be translated into results about the non-existence of compact representations. The last section of the chapter contains the application of these result to the problem of belief revision (a belief revision operator is a function from a pair of formulas into a formula).

6.1 The Problem of Compact Representations

The problem we address is the following: given a formula (or a theory) T and a formula P, both Horn, whose size are $||T||$ and $||P||$ respectively, is it always possible to represent the revised knowledge base with a propositional formula whose size is polynomial w.r.t. $||T|| + ||P||$? Clearly, in the cases when it is not, it can be considered impossible, from a practical point of view (limitation of computer memory), to represent the revised knowledge with a propositional formula that can be effectively stored.

Let us give an example of the non-triviality of compact representation results. We want to determine if there is a formula T_1, equivalent to $T * P$, such that the size of T_1 is polynomial w.r.t. $||T|| + ||P||$. Let us consider the Ginsberg's revision of T with P, where

$$\begin{aligned} T &= \{x_1, \ldots, x_n, y_1, \ldots y_n\} \\ P &= \bigwedge_{1 \leq i \leq n} \neg x_i \vee \neg y_i \end{aligned}$$

The theory T and the formula P are consistent, while $T \cup \{P\}$ is not. The problem is that T contains both x_i and y_i, while P constrains at least one of

them to be false. The maximal subsets of T that are consistent with P are as follows: for each i there is a choice of deleting x_i or y_i from T. A choice of an index i does not influence the choice over j, thus we can delete x_i and x_j, as well as deleting x_i and y_j. The maximal subsets of T that are consistent with P are thus obtained by choosing exactly one formula in $\{x_i, y_i\}$ for any i. As a result,

$$\begin{array}{rcll} W(T,P) & = & \{ & \{x_1, x_2, \ldots, x_{n-1}, x_n\}, \\ & & & \{x_1, x_2, \ldots, x_{n-1}, y_n\}, \\ & & & \{x_1, x_2, \ldots, y_{n-1}, x_n\}, \\ & & & \{x_1, x_2, \ldots, y_{n-1}, y_n\}, \\ & & & \vdots \\ & & & \{y_1, y_2, \ldots, y_{n-1}, y_n\} \quad \} \end{array}$$

As a result, $W(T,P)$ is exponential, and thus $\vee W(T,P)$ is exponential as well. One may be tempted to conclude that revising T with P (using $*_G$) leads to an exponential k.b.. However, $\vee W(T,P)$ is equivalent to $\bigwedge_{1 \leq i \leq n} x_i \not\equiv y_i$, which is very small (linear in the size of P). This is our first point in the definition of "compact representation": the revised k.b. can be represented in many ways (i.e. all the propositional formulas equivalent to the one given by the definition of revision). Thus, for T and P above, we say that the result of revising T with P can be represented in polynomial space.

For the model based revisions the idea is similar: all of them are indeed defined in terms of sets of models, i.e. the definition is $Mod(T * P) =$ `a_certain_set_of_models`. As a result, there is no unique way to define the propositional formula $T * P$. Since we are looking for compact representations, we say that $*$ has a compact representation if and only if there is a formula T_1, such that $Mod(T_1) = Mod(T * P)$, and T_1 has size polynomial in the size of T and P.

Summarizing, we are not looking for the size of $T * P$ as follows from the definition, but for the size of the shortest formula equivalent to $T * P$.

The second point regards the definition of equivalence. In the last paragraphs, we assumed the usual definition of equivalence:

Definition 35 *Two formulas (or circuits, or theories) T_1 and T_2 are logically equivalent*

$$T_1 \equiv T_2 \qquad \textit{iff} \qquad Mod(T_1) = Mod(T_2)$$

This is a strong definition, in the sense that there are cases in which we would like to define equivalent two knowledge bases that do not satisfy this definition. Consider the following alternative: define an extended form of equivalence $\equiv^Q$ as follows.

$$T_1 \equiv^Q T_2 \text{ iff for each } R, \text{ it holds } T_1 \models R \text{ iff } T_2 \models R$$

The letter Q above the symbol of equivalence stresses the fact that this is a new kind of equivalence, different from the previous one. We could have written for example $\cong$ instead of $\equiv^Q$. The letter Q is there just to make easier to remember that this is an equivalence w.r.t. the set of implied queries. This second definition highlights the fact that knowledge bases such T_1 and T_2 are used for representing knowledge. In this case, the knowledge represented by T_1 is the set of its implied formulas, thus T_2 can be considered equivalent if it implies the same formulas.

These two forms of equivalence are similar. Indeed, if we allow R to be any formula, they coincide. Assume instead that R is limited to be a formula on the alphabet X, and allow T_1 and T_2 to be built on a larger alphabet:

Definition 36 *Two formulas (or circuits, or theories) T_1 and T_2 are query equivalent over an alphabet X*

$$T_1 \equiv_X^Q T_2 \text{ iff for any } R \text{ such that } Var(R) \subseteq X, \text{ it holds } T_1 \models R \text{ iff } T_2 \models R$$

Where not confusing, we omit the letter X in the symbol $\equiv_X^Q$. Let us analyze this definition in detail. First, this definition is different from the previous ones. Assume $X = \{x_1, x_2\}$, and consider $T_1 = x_1 \wedge y_1$, $T_2 = x_1 \wedge \neg y_2$. Clearly, $T_1 \not\equiv T_2$: they are not logically equivalent. However, for any R such that $Var(R) \subseteq X$, we have $T_1 \models R$ if and only if $T_2 \models R$. As a result, $T_1 \equiv_X^Q T_2$: these formulas are query equivalent over X.

Now we show why the concept of query equivalence is relevant to the study of compact representations. The propositional formulas involved in a revision (T and P) are usually built over a fixed alphabet X. These formulas are used to represent knowledge: we query the k.b. $T * P$ by checking which facts R are true in $T * P$, that is, verifying whether $T * P \models R$. The query R is usually built over the same alphabet X. Thus, when we are looking for a formula equivalent to $T * P$, if we are only interested in preserving queries over X, we can consider the query equivalence instead of the logical equivalence.

This can be useful, since formulas that are query equivalent to $T * P$ can be smaller than the logical equivalent ones. For example, let

$$T_1 = [(x_1 \vee x_2 \vee x_3 \vee x_4) \vee (x_5 \wedge \neg(x_1 \vee x_2 \vee x_3 \vee x_4))] \wedge (x_6 \vee \neg(x_1 \vee x_2 \vee x_3 \vee x_4))$$

the size of this formula can be reduced observing that $(x_1 \vee x_2 \vee x_3 \vee x_4)$ appears in many places. Thus, we can define a *macro* for it:

$$T_2 = [e \equiv (x_1 \vee x_2 \vee x_3 \vee x_4)] \wedge [e \vee (x_5 \wedge \neg e)] \wedge (x_6 \vee \neg e)$$

This new formula is smaller than the previous one. Nevertheless, $T_1 \equiv_X^Q T_2$. Note that these formulas are not logically equivalent, since T_1 has models in which e is false but $(x_1 \vee x_2 \vee x_3 \vee x_4)$ is true, while T_2 does not.

As works such [CDLS95] prove, there is a big gain in using query equivalence instead of logical equivalence: sometimes the use of macros can exponentially reduce the size of compact representations.

We now try to summarize the above argumentation:

1. We are trying to determine the size of formulas that are equivalent to $T * P$, rather than the size of $T * P$ itself.

2. There are two possible ways to define the equivalence: the logical (classical) equivalence, and the query equivalence (identity of the sets of implied formulas). The second one allows the use of macros, thus can give smaller representations.

We define compact representation for a belief revision operator $*$ a formula which is equivalent to $T * P$ and such that its size is polynomial in $||T|| + ||P||$. Namely, we have

Definition 37 *A compact representation w.r.t. logical equivalence is a formula (circuit, theory) which is logically equivalent to $T * P$ and has size polynomial in $||T|| + ||P||$.*

Definition 38 *A compact representation w.r.t. query equivalence is a formula (circuit, theory) which is query equivalent to $T * P$ and has size polynomial in $||T|| + ||P||$.*

6.2 Model Equivalence

In this section we introduce, following [CDSS97], a third kind of equivalence. This is motivated by the fact that knowledge is sometimes better represented by sets of models rather that sets of formulas [HV91].

We define, following [CDSS97], the so-called model equivalence. This is a notion similar to the query equivalence, in the case that k.b.'s are used to represent models instead of formulas. In the definition of query equivalence, we define two formulas to be equivalent when they represent the same sets of formulas, that is, when the set of implied formulas are the same. Define two formulas (or circuits, or theories) T_1 and T_2 to be model equivalent if and only if, for each model $I \in Mod(T_1)$ it holds $I \in Mod(T_2)$, and vice versa.

This definition coincides with that of logical equivalence. However, T_1 and T_2 may be built over different sets of variables, but we are only interested in the value of models over a fixed set of variables X.

Definition 39 *We say that T_1 is model equivalent to T_2 over X (written $T_1 \equiv_X^M T_2$) if and only if, for each model $I \in Mod(T_1)$ we can find in polynomial time a model $J \in Mod(T_2)$ such that $I \cap X = J \cap X$ (and vice versa).*

Where not confusing, we write simply $T_1 \equiv^M T_2$. With this definition in mind, we define the compact representation of a belief revision operator w.r.t. model equivalence as follows.

Definition 40 *A compact representation w.r.t. model equivalence for a belief revision operator $*$ is a formula (theory, circuit) that is model equivalent to $T * P$ and has size polynomial in the size of T and P.*

6.3 Compilability and Compact Representation

In this section we show the links between compilability of problems in belief revision and the existence of compact representations for them.

Consider first the logical equivalence. We can prove the following theorem

Theorem 53 *If $*$ has a compact representation w.r.t. logical equivalence, then its MC is $\rightsquigarrow$*P *and its QA is* compcoNP*.*

Proof. By hypothesis, there is a formula T_1 which is logically equivalent to $T * P$, and has size polynomial w.r.t. $||T|| + ||P||$. The problem of model checking for $*$ is $\rightsquigarrow$P, since $M \models T * P$ can be checked by verifying whether $\langle f(T, P), M\rangle \in S$, where

$$\begin{aligned} f(T, P) &= T_1 \\ S &= \{\langle T', M'\rangle \mid M' \text{ is a model of } T'\} \end{aligned}$$

By hypothesis T_1 has size polynomial, thus f is polysize. Furthermore, S is a polynomial language. As a result, the problem of model checking is $\rightsquigarrow$P.

The problem of query answering is compcoNP, since $T*P \models Q$ is equivalent to $\langle f(T, P), Q\rangle \in R$, where f is as above, and R is

$$R = \{\langle T', Q'\rangle \mid T' \text{ implies } Q'\}$$

R is a coNP language, thus the query answering is compcoNP. □

This result is useful for proving that an operator does *not* have a compact representation: if the model checking for a revision operator is not in $\rightsquigarrow$P, then there is no compact representation for it w.r.t. logical equivalence. For example, the problem of model checking for Satoh's revision is $\|\rightsquigarrow$NP hard, thus is not in $\rightsquigarrow$P, thus there is no compact representation w.r.t. logical equivalence.

A similar theorem can be proved for compact representations w.r.t. model and query equivalence.

Theorem 54 *If $*$ has a compact representation w.r.t. model equivalence, then its MC is in $\rightsquigarrow$*P *and its QA is in* compcoNP*.*

Proof. This theorem can be proved in the same manner of Theorem 53. □

Theorem 55 *If $*$ has a compact representation w.r.t. query equivalence, the its MC is in* compNP*, and its QA is in* compcoNP*.*

Proof. By hypothesis, there is a formula T_1, whose size is polynomial w.r.t. $||T|| + ||P||$, such that $T * P \models Q$ if and only if $T_1 \models Q$, for each formula Q over a set of variables X. The difference w.r.t. the previous theorem is that T_1 is allowed to have variables not in X, while in the problem of model checking and query answering we are only interested in models and formulas over X.

Consider the query answering first. To check whether $T * P \models Q$, one can verify whether $\langle f(T, P), Q\rangle \in S$, where

$$\begin{aligned} f(T, P) &= T_1 \\ S &= \{\langle T', Q'\rangle \mid Q' \text{ is a formula over } X \text{ and } T' \models Q'\} \end{aligned}$$

Notice that no restriction is imposed on the variables of T' in the language S. Note also that f is polysize and S is a coNP language. As a result, the query answering problem is compcoNP.

About the model checking: $M \models T*P$ if and only if $T*P \not\models \neg Form(\{M\})$, which is compcoNP. As a result, the model checking problem is compNP (in [CDLS96b] it is proved that co-compNP=compcoNP). □

As for the first theorem, these ones are useful for proving the non-existence of compact representations.

6.4 Compact Representations of Revision

The results of compact representation of the revision operators introduced are reported in Table 6.1. The table has six columns. The first three contains results about the general case, while the last three ones contains results about the restricted case in which P has size bounded by a constant number. In each of these cases, there are three columns because there are three possible forms of equivalence: logical, model, and query.

The negative results of the table follows from Theorem 53, 54, 55, and from the complexity of the revision operators involved. Let us state the results formally. The results about WIDTIO revision are trivial, since $T * P$ is a subset of $T \cup \{P\}$, thus its size is bounded by the size of T and P.

6.4.1 General Unbounded Case

In this section we prove the results of the first three columns of Table 6.1, that is, the case in which T and P are general formulas, and P is not bounded by a constant.

	General case			Bounded case		
	Logical equiv.	Model equiv.	Query equiv.	Logical equiv.	Model equiv.	Query equiv.
Ginsberg	NO	NO	NO	NO	NO	NO
Winslett	NO	NO	NO	YES	YES	YES
Borgida	NO	NO	NO	YES	YES	YES
Forbus	NO	NO	NO	YES	YES	YES
Satoh	NO	NO	NO	YES	YES	YES
Dalal	NO	NO	YES	YES	YES	YES
Weber	NO	NO	YES	YES	YES	YES
WIDTIO	YES	YES	YES	YES	YES	YES

Table 6.1: Has $*$ a compact representation?

Theorem 56 *The revision operators of Ginsberg, Winslett, Borgida, Forbus, Satoh, Dalal, and Weber do not have a compact representation w.r.t. logical and model equivalence.*

Proof. All these operators are not in $\rightsquigarrow$P, either because their model checking is $\|\!\not\rightsquigarrow$NP hard or $\|\!\not\rightsquigarrow$coNP hard. Assume, by contradiction, that they have a compact representation w.r.t. logical equivalence. By Theorem 53 it follows that they are in $\rightsquigarrow$P, which is a contradiction. Similarly, if they have a compact representation w.r.t. model checking, their model checking is $\rightsquigarrow$P. □

The next theorem is about the compact representations w.r.t. query equivalence.

Theorem 57 *The revision operators of Ginsberg, Winslett, Borgida, Forbus, and Satoh do not have a compact representation w.r.t. query equivalence.*

Proof. The complexity of model checking for all these operators is (at least) $\|\!\not\rightsquigarrow$coNP hard. As a result, none of them is in compNP. If there exists a compact representation w.r.t. query equivalent for one of them, then model checking is compNP, which is a contradiction. □

We now turn our attention to operators that are query-compactable. Let $Var(T) = Var(P) = X = \{x_1, \ldots, x_n\}$, and Y be be another set of (distinct) letters, one-to-one with X. Let $EXA(k, X, Y, W)$ denote a formula containing letters of X and Y, and possibly other letters W, which is true iff the Hamming distance between the values assigned to X and Y is exactly k. The revised theory $T *_D P$ can be expressed as $T[X/Y] \wedge P \wedge EXA(k_{T,P}, X, Y, W)$, where $k_{T,P}$ is the minimum distance between the models of T and P.

The formula $EXA(k, X, Y, W)$ can be constructed in several ways. For example, first compute the number of true exclusive-or's between each x_i and y_i. The binary representation of this number can be computed with a suitable adding circuit, that requires $O(n^2)$ half adders. This circuit can be expressed as a formula using $O(n^2)$ new letters. Secondly, write the formula that equals the bits of this number with those of the binary representation of k.

Theorem 58 *The formula $T[X/Y] \wedge P \wedge EXA(k_{T,P}, X, Y, W)$ is query equivalent to $T *_D P$.*

Proof. Let Q be a query on the alphabet X and $k_{T,P}$ be the minimal distance between the models of P and T. We prove the theorem by showing that $T[X/Y] \wedge P \wedge EXA(k_{T,P}, X, Y, W) \models Q$ iff $T *_D P \models Q$.

If. We show that $T[X/Y] \wedge P \wedge EXA(k_{T,P}, X, Y, W) \not\models Q$ implies $T *_D P \not\models Q$. Let M be a model of $T[X/Y] \wedge P \wedge EXA(k_{T,P}, X, Y, W)$ such that $M \not\models Q$. Thus, $M \cap X$ satisfies P, and, since M satisfies $EXA(k_{T,P}, X, Y, W)$, $M \cap X$ has a distance $k_{T,P}$ from $M \cap Y$, and $M \cap Y$ satisfies $T[X/Y]$. Since $k_{T,P}$ is by definition the minimal distance between a model of T and a model of P, $M \cap X$ is also a model of $T *_D P$. Therefore, $T *_D P \not\models Q$.

Only If. We show that if $T *_D P \not\models Q$ holds, then

$$T[X/Y] \wedge P \wedge EXA(k_{T,P}, X, Y, W) \not\models Q$$

Let M be a model of $T *_D P$ such that $M \not\models Q$. This model satisfies P. Let MT be a model of T having distance $k_{T,P}$ from M. Define M' as $M \cup \{y_i | x_i \in MT\}$. Obviously, M' satisfies both P and $T[X/Y]$, and by definition of MT it can be extended to an assignment to W so that it also satisfies $EXA(k_{T,P}, X, Y, W)$. Therefore, $T[X/Y] \wedge P \wedge EXA(k_{T,P}, X, Y, W) \not\models Q$. □

Now we show how Weber's revision can be compactly represented. Winslett, in [Win90] gives a similar proof, but only if the new formula has a size bounded by a constant. Our result, instead, does not need such a restriction. Let $\Omega = \{\omega_1, \ldots, \omega_k\}$ be the set of letters in the definition of Weber's revision operator, and $Z = \{z_1, \ldots, z_k\}$ be a new set of letters one-to-one with Ω.

Theorem 59 *The formula $T[\Omega/Z] \wedge P$ is query equivalent to $T *_{Web} P$.*

Proof. Let Q be a formula on the alphabet X. We show that $T *_{Web} P \models Q$ iff $T[\Omega/Z] \wedge P \models Q$.

If. Assume that $T *_{Web} P \models Q$ and $T[\Omega/Z] \wedge P \not\models Q$. As a consequence, for all models N of $T *_{Web} P$ we have that $N \models Q$ and there exists a model M of $T[\Omega/Z] \wedge P$ (on the alphabet $X \cup Z$) such that $M \not\models Q$. Since $M \models T[\Omega/Z]$, we construct a model M' on the alphabet X as follows: for all letters l of $X \backslash \Omega$, we have that $l \in M'$ iff $l \in M$. For all letters ω_i of Ω, $\omega_i \in M'$ iff $z_i \in M$.

Obviously, $M' \not\models Q$, and, by construction, $M' \models T$. Since $M \models P$, $M' \models T$ and $Diff(M, M') \subseteq \Omega$, it follows that M is a model of $T *_{Web} P$ that does not satisfy Q, hence contradiction arises.

Only If. Assume that $T *_{Web} P \not\models Q$ and $T[\Omega/Z] \wedge P \models Q$. As a consequence, for all models M of $T[\Omega/Z] \wedge P$ we have that $M \models Q$ and there exists a model N of $T *_{Web} P$ such that $N \not\models Q$. Since $N \models T *_{Web} P$, we construct a model N' on the alphabet $X \cup Z$ as follows: for all letters x_i of X, we have that $x_i \in N'$ iff $x_i \in N$. For all letters $z_i \in Z$, $z_i \in N'$ iff $\omega_i \in \Omega$. Obviously, $N' \not\models Q$, $N' \models P$, and, by construction, $N' \models T[\Omega/Z]$. Therefore, there exists a model of $T[\Omega/Z] \wedge P$ that does not satisfy Q, thus contradicting the hypothesis. □

We note that this representation of $T *_{Web} P$ increases the size of T only by the length of P whereas the compact representation of $T *_D P$ requires a formula whose size is quadratic in the number of the letters.

6.4.2 Bounded Case

Let us now consider the bounded case. Only Ginsberg's operator has not a compact representation, since the complexity and compilability are at the same level of the unbounded case. The situation for the other revision operators is more complex. As it turns out all of them admit a compact representation, w.r.t. logical equivalence, when the size of P is bounded. This implies also the existence of compact representation w.r.t. model and query equivalence. In this section we assume that there exists a constant k such that $||P|| \leq k$.

Without loss of generality, since the size of P is bounded we assume that the alphabet $Var(P)$ of P is included in the alphabet $Var(T)$ of T (e.g.we can add to T the formula $\bigwedge_{x \in Var(P)}(x \vee \neg x)$). Because of the assumption of $||P||$ being bounded by k, it follows that $|Var(P)| \leq k$. Without loss of generality, we assume $|Var(P)| = k$, and denote letters in $Var(P)$ as $\{v_1, \ldots, v_k\}$.

We use the following notation: for every set of letters H, we denote with $\hat{H}$ the set $\{\neg x | x \in H\}$. The formula $F[H/\hat{H}]$, where $H \subseteq Var(F)$, is F with each letter in H replaced by the corresponding letter in $\hat{H}$ (that is, its negation).

A useful property that we shall use in the sequel is the following.

Proposition 60 *For each interpretation M of the letters in $Var(F)$ and set $H \subseteq Var(F)$, $M \models F$ if and only if $Diff(M, H) \models F[H/\hat{H}]$.*

In other words, if an interpretation M satisfies a formula F then, for any given set of letters H, the model $Diff(M, H)$, that agrees with M on all letters in $Var(F) \backslash H$ and disagrees on all letters in H, satisfies the formula $F[H/\hat{H}]$, where all letters in H are replaced by their negation. For example, let $F = x_1 \wedge (x_2 \vee \neg x_3)$, $M = \{x_1\}$ and $H = \{x_2, x_3\}$. Note that $M \models$

F. Applying the definitions, we obtain that $Diff(M, H) = \{x_1, x_2, x_3\}$ and $F[H/\hat{H}] = x_1 \wedge (\neg x_2 \vee \neg\neg x_3)$. It follows that $\{x_1, x_2, x_3\} \models x_1 \wedge (\neg x_2 \vee \neg\neg x_3)$.

Another important property of model-based revision operators that we frequently use in this section is the following.

Proposition 61 *Let M be a model of T and $*$ one of the revision operators of Borgida, Dalal, Forbus, Satoh, Weber, and Winslett. Then, there exists a model N of $T * P$ such that $Diff(M, N) \subseteq Var(P)$.*

This property states that for every model of T there exists a model of $T * P$ whose distance is bounded by the letters of P. This is sometimes crucial in showing the existence of compact representations since it allows to focus our attention only on the letters of P. This proposition is mentioned by Eiter and Gottlob in [EG92, proof of Lemma 6.1].

We start with the compact representation for Winslett's operator. We exhibit a compact representation of $T *_W P$ which uses exactly the same alphabet of T and P. Basically, we exploit the fact that we can explicitly represent all assignments to $Var(P)$ in constant space, since $|Var(P)| = k$.

Let S be an arbitrary set of letters such that $S \subseteq Var(P)$. The formula

$$P \wedge \bigvee_{S \subseteq Var(P)} (T[S/\hat{S}] \wedge R)$$

where $\bigvee_{S \subseteq Var(P)}$ means a disjunction for all possible subsets S of $Var(P)$, is satisfied by a model N of P iff there is a model M of T (whose distance from N is given by the set S), satisfying formula R.

The set S represents the distance between N and M. Formula R specifies that there is no other model in P whose distance from M is less than S.

$$R = \neg \bigvee_{C \subseteq Var(P)\ ,\ Diff(C,S) \subset S} P[C/\hat{C}]$$

where $\bigvee_{C \subseteq Var(P)\ ,\ C \Delta S \subset S}$ means a disjunction for all possible subsets C of $Var(P)$, satisfying condition $Diff(C, S) \subset S$ (an equivalent condition is $C \neq \emptyset, C \subseteq S$). This formula imposes a condition over the distance C between two models of P. Namely, it forbids that C is between M and N.

The whole formula is

$$P \wedge \bigvee_{S \subseteq V(P)} (T[S/\hat{S}] \wedge \neg \bigvee_{C \subseteq V(P)\ ,\ Diff(C,S) \subset S} P[C/\hat{C}]) \tag{6.1}$$

Theorem 62 *Formula (6.1) has size linear in $||T||$ and is logically equivalent to $T *_W P$.*

Proof. The formula uses only letters of $Var(T)$, and has size linear in $||T||$, but exponential in $||P||$, namely $O(||T|| \cdot 2^{||P||} + 2^{2||P||})$. As for the logical equivalence, we split the proof in two parts.

(*1st part*) Let N be a model of $T *_W P$, i.e. $N \in Mod(P)$ and $\exists M \in Mod(T)$ such that $Diff(M, N)$ is minimal, i.e. there is no other N' such that $Diff(M, N') \subset Diff(M, N)$. We prove that N is also a model of formula (6.1). Let S be $Diff(M, N)$; the following propositions are equivalent:

1. $M \models T$
2. $Diff(M, S) \models T[S/\hat{S}]$
3. $N \models T[S/\hat{S}]$

(1 equivalent to 2 by Proposition 60; 2 equivalent to 3 by associativity of $Diff$). We prove now that $S \subseteq Var(P)$, by assuming that there is a letter l such that $l \in S$, $l \notin Var(P)$, and showing that a contradiction arises. Since S is $Diff(M, N)$, there are only two possibilities:

1. $l \in N$, $l \notin M$. Define N' as $N \backslash \{l\}$. Since $l \notin Var(P)$, $N' \models P$. Since $l \notin M$, $Diff(N', M) \subset S$, contradicting the hypothesis that there is no N' that is closer to M than N.
2. $l \notin N$, $l \in M$. Define N' as $N \cup \{l\}$. Since $l \notin Var(P)$, $N' \models P$. Since $l \in M$, $Diff(N', M) \subset S$, contradicting the hypothesis that there is no N' that is closer to M than N.

We now prove that N is a model of the remaining part of formula (6.1), i.e. that $N \models \neg \bigvee_{C \subseteq V(P)\ ,\ Diff(C,S) \subset S} P[C/\hat{C}]$. We do so by assuming that there is a set C such that $C \subseteq Var(P), Diff(C, S) \subset S$ and $N \not\models \neg P[C/\hat{C}]$, and showing that a contradiction arises. $N \not\models \neg P[C/\hat{C}]$ is equivalent to $N \models P[C/\hat{C}]$, which, by Proposition 60, is equivalent to $Diff(N, C) \models P$. Now, $Diff(M, Diff(N, C)) = Diff(Diff(M, N), C) = Diff(S, C)$. The last set is, by assumption, strictly contained in S. This leads to a contradiction: remind that by assumption N is a model of $T *_W P$; this means that $Diff(M, N)$ is minimal, i.e. S is minimal; but we found a model ($Diff(N, C)$) of P such that $Diff(M, Diff(N, C)) \subset S$, hence S cannot be minimal.

(*2nd part*) Let N be a model of formula (6.1). We prove that N is also a model of $T *_W P$, i.e. $N \in Mod(P)$ and $\exists M \in Mod(T)$ such that there is no N' with $Diff(M, N') \subset Diff(M, N)$.

First of all, $N \in Mod(P)$. Secondly, we know there is a set $S \subseteq Var(P)$ such that $N \models T[S/\hat{S}] \wedge \neg \bigvee_{C \subseteq Var(P)\ ,\ Diff(C,S) \subset S} P[C/\hat{C}]$. By Proposition 60, this is equivalent to $Diff(N, S) \models T \wedge \neg(\bigvee_{C \subseteq V(P)\ ,\ Diff(C,S) \subset S} P[C/\hat{C}])[S/\hat{S}]$. Define M as $Diff(N, S)$ (which implies $S = Diff(M, N)$). We have to prove

that S is a minimal distance between M and a model of P. We assume that this is not true, and prove that a contradiction follows. If S is not minimal, it follows that there must be a model N' of P which is closer to M than N, i.e. $Diff(M, N')subsetS$. Define C as $Diff(M, N')$, i.e.$N' = Diff(N, C)$. The following conditions are equivalent:

1. $N' \models P$
2. $Diff(N, C) \models P$
3. $N \models P[C/\hat{C}]$

Next step is to note the following chain of equalities:

$$\begin{aligned} Diff(C, S) &= Diff(Diff(N, N'), S) \\ &= Diff(Diff(N, N'), Diff(M, N)) \\ &= Diff(M, N') \end{aligned}$$

Using the fact that $Diff(M, N') \subset S$, we have that $Diff(C, S) \subset S$. This implies $C \subset S$, which in turn implies $C \subset Var(P)$. Contradiction follows by noticing that we found a set C such that $C \subseteq Var(P), Diff(C, S) \subset S$ and $N \models P[C/\hat{C}]$. □

Using Theorem 62 and the fact that $T *_B P$ is $T \wedge P$ if consistent and $T *_W P$ otherwise, we obtain:

Corollary 2 *There exists a formula of size linear in $||T||$ that is logically equivalent to $T *_B P$.*

We now exhibit a compact representation of $T *_F P$ w.r.t. logical equivalence. The main difference between Forbus' operator and Winslett's one relies on the fact that the notion of distance between models for the former one is based on set cardinality, while for the latter one on set containment. As a consequence, we obtain a formula very similar to the one obtained for Winslett's operator. In fact, the formula representing $T *_F P$ is the following:

$$P \wedge \bigvee_{S \subseteq Var(P)} (T[S/\hat{S}] \wedge \neg \bigvee_{C \subseteq Var(P)\ ,\ |Diff(C,S)|<|S|} P[C/\hat{C}]) \qquad (6.2)$$

(cf. formula (6.1) and note that here cardinality of sets is considered in the subscript of the last disjunction).

Theorem 63 *Formula (6.2) has size linear in $||T||$ and is logically equivalent to $T *_F P$.*

Proof. The formula uses only letters of $Var(T)$, and has size linear in $||T||$, but exponential in $||P||$. Namely, its size is $O(2^{2||P||})$. As for the other statement, we split the proof in two parts.

(*1st part*) Let N be a model of $T *_F P$, i.e. $N \in Mod(P)$ and $\exists M \in Mod(T)$ such that there is no $N' \in Mod(P)$ with $|Diff(M, N')| \leq |Diff(M, N)|$. We have to prove that N is also a model of formula (6.2).

The proof is omitted, as it can be obtained using a schema similar to the one used for the corresponding part of Theorem 62 (the only differences being the usage of set cardinality, of $\leq$ instead of $\subseteq$.

(*2nd part*) Let N be a model of formula (6.2). We prove that N is also a model of $T *_F P$, i.e. $N \in Mod(P)$ and exists $M \in Mod(T)$ such that $|M \Delta N|$ is minimal.

First of all, $N \models P$. Then we know there is a set $S \subseteq Var(P)$ such that $N \models T[S/\hat{S}] \wedge \neg \bigvee_{C \subseteq Var(P)\ ,\ |Diff(C,S)|<|S|} P[C/\hat{C}]$. By Proposition 60, this is equivalent to $(Diff(N, S) \models T \wedge \neg(\bigvee_{C \subseteq Var(P)\ ,\ |Diff(C,S)|<|S|} P[C/\hat{C}])[S/\hat{S}]$. Define M as $Diff(N, S)$ (which implies $S = Diff(M, N)$). We have to prove that $|S|$ is minimal. We assume that this is not true, and prove that a contradiction follows. If $|S|$ is not minimal, it follows that there must be a model N' of P which is closer to M than N, i.e.$|Diff(M, N')| < |S|$. Define C as $Diff(N, N')$, which implies $N' = Diff(N, C)$. The following conditions are equivalent:

1. $N' \models P$
2. $Diff(N, C) \models P$
3. $N \models P[C/\hat{C}]$

Next step is to note the following chain of equalities:

$$\begin{aligned} Diff(C, S) &= Diff(Diff(N, N'), S) \\ &= Diff(Diff(N, N'), Diff(M, N)) \\ &= Diff(M, N') \end{aligned}$$

Using the fact that $|Diff(M, N')| < |S|$, we have that $|Diff(C, S)| < |S|$.

The last step that is still missing is to prove that $C \subseteq Var(P)$ (note: this step is not necessary in the proof of Theorem 62). We assume the contrary, and show that this leads to a contradiction. If $C \not\subseteq Var(P)$, then there is a letter $l \in C$ such that $l \notin Var(P)$. Since $C = Diff(N, N')$, there are only two possibilities:

1. $l \in N$, $l \notin N'$. Define N'' as $N' \cup \{l\}$. Since $l \notin Var(P)$, $N'' \models P$. Since $l \notin Var(P)$, $l \notin S$, hence $l \notin Diff(N, M)$, hence $l \in M$, which implies $Diff(M, N'') \subset Diff(M, N')$, contradicting the above hypothesis.

2. $li \notin N$, $l \in N'$. Define N'' as $N' \backslash \{l\}$. Since $l \notin Var(P)$, $N'' \models P$. Since $l \notin Var(P)$, $l \notin S$, hence $l \notin Diff(N, M)$, hence $l \notin M$, which implies $Diff(M, N'') \subset Diff(N', N'')$, contradicting the above hypothesis.

Contradiction follows by noticing that we found a set C such that $C \subseteq Var(P), |Diff(C, S)| < |S|$ and $N \models P[C/\hat{C}]$. □

We conclude this section by showing an example of the formulae obtained by applying formula (6.2).

Example 2 *Let T and P be defined as:*

$$\begin{aligned} T &= a \wedge b \wedge c \wedge d \wedge e \\ P &= \neg a \vee \neg b \end{aligned}$$

T has just one model (call it M), while P has 3×2^3 models (each combination of the models of $\neg a \vee \neg b$ with $2^{\{c,d,e\}}$). $T \wedge P$ has no models. The minimum cardinality of differences between M and each model of P is 1, corresponding to models $\{a, c, d, e\}$ and $\{b, c, d, e\}$ of P. These are the models of $T *_F P$.

The set of variables of P is $Var(P) = \{a, b\}$. Applying (6.2), we get for $T *_F P$

$$P \wedge \bigvee_{S=\{\},\{a\},\{b\},\{a,b\}} (T[S/\hat{S}] \wedge \neg \bigvee_{C \subseteq Var(P)\ ,\ |Diff(C,S)|<|S|} P[C/\hat{C}])$$

i.e.

$$\begin{aligned} P \wedge (& (a \wedge b \wedge c \wedge d \wedge e) \\ \vee & ((\neg a \wedge b \wedge c \wedge d \wedge e) \wedge \neg \bigvee_{C=\{a\}} P[C/\hat{C}]) \\ \vee & ((a \wedge \neg b \wedge c \wedge d \wedge e) \wedge \neg \bigvee_{C=\{b\}} P[C/\hat{C}]) \\ \vee & ((\neg a \wedge \neg b \wedge c \wedge d \wedge e) \wedge \neg \bigvee_{C=\{a\},\{b\},\{a,b\}} P[C/\hat{C}]) \ ) \end{aligned}$$

i.e.

$$\begin{aligned} (\neg a \vee \neg b) \wedge (& (a \wedge b \wedge c \wedge d \wedge e) \\ \vee & ((\neg a \wedge b \wedge c \wedge d \wedge e) \wedge \neg(a \vee \neg b)) \\ \vee & ((a \wedge \neg b \wedge c \wedge d \wedge e) \wedge \neg(\neg a \vee b)) \\ \vee & ((\neg a \wedge \neg b \wedge c \wedge d \wedge e) \wedge \neg((a \vee \neg b) \vee (\neg a \vee b) \vee (a \vee b))) \ ) \end{aligned}$$

It is easy to verify that this formula has exactly two models: $\{b, c, d, e\}$ (the unique model of the subformula on the second line), and $\{a, c, d, e\}$ (the unique model of the subformula on the third line).

For Satoh's operator, the representation stems directly from the definition. We first compute off-line the set $K_{T,P}$ of minimal global differences between models of T and models of P. Then we represent Satoh's revision as:

$$P \wedge \bigvee_{S \in K_{T,P}} T[S/\hat{S}] \tag{6.3}$$

Theorem 64 *Formula (6.3) has size linear in $||T||$ and is logically equivalent to $T *_S P$.*

Proof. The size of the formula is $O(||P|| + ||T||^{|K_{T,P}|})$. By Proposition 61, each set in $K_{T,P}$ is contained in $Var(P)$ hence $K_{T,P}$ is a subset of the powerset of $Var(P)$, hence $|K_{T,P}|$ is bounded by 2^{ck}, for some constant c. Therefore, the size of Formula (6.3) is linear in $||T||$ and doubly exponential in k.

As for the logical equivalence, we split the proof in two parts.

(*1st part*) Let N be a model of $T *_S P$. By definition, $N \in Mod(P)$ and there exists $M \in Mod(T)$ such that $Diff(M, N) \in K_{T,P}$. Let S be $Diff(M, N)$; then $S \in K_{T,P}$, and $N = Diff(M, S)$. From Proposition 60, $M \in Mod(T)$ iff $Diff(M, S) \models T[S/\hat{S}]$, that is, $N \models T[S/\hat{S}]$. Hence N is a model of Formula (6.3).

(*2nd part*) Let N be a model of Formula (6.3). Then $N \models P$ and there is an $S \in K_{T,P}$ such that $N \models T[S/\hat{S}]$. From Proposition 60, this implies $Diff(N, S) \models T$. Let $M \doteq Diff(N, S)$. Then M is a model of T and $Diff(M, N) \in K_{T,P}$. Hence N is a model of $T *_S P$. □

Also for Dalal's operator, the representation stems directly from the definition of revision, once $k_{T,P}$ (the minimum cardinality of sets in $K_{T,P}$) is computed off-line. Dalal's revision can be represented as:

$$P \wedge \bigvee_{|S|=k_{T,P}} T[S/\hat{S}] \tag{6.4}$$

Theorem 65 *Formula (6.4) has size linear in $||T||$ and is logically equivalent to $T *_D P$.*

Proof. The size of the formula is $O(||P|| + ||T||^b)$, where b is the binomial coefficient $\binom{k}{k_{T,P}}$. Since sets in $K_{T,P}$ are subsets of $Var(P)$ by Proposition 61, $k_{T,P} \leq k = ||P||$, hence the above binomial is less than 2^k. Therefore, the size of Formula (6.4) is linear in $||T||$ and, at most, doubly exponential in k.

As for the other statement, the proof is similar to the one for the representation of Satoh's revision.

(*1st part*) Let N be a model of $T *_D P$. By definition, $N \in Mod(P)$ and there exists $M \in Mod(T)$ such that $|Diff(M, N)| = k_{T,P}$. Let S be

$Diff(M, N)$; then $|S| = k_{T,P}$, and $N = Diff(M, S)$. From Proposition 60, $M \in Mod(T)$ iff $Diff(M, S) \models T[S/\hat{S}]$, that is, $N \models T[S/\hat{S}]$. Hence N is a model of Formula (6.4).

(*2nd part*) Let N be a model of Formula (6.3). Then $N \models P$ and there is an S such that $|S| = k_{T,P}$ and $N \models T[S/\hat{S}]$. From Proposition 60, this implies $Diff(N, S) \models T$. Let M be $Diff(N, S)$. Then M is a model of T and $|Diff(M, N)| = k_{T,P}$. Hence N is a model of $T *_D P$. □

A similar representation can be found for Weber's operator. Computed off-line the set Ω, the explicit representation of $T *_{Web} P$ is

$$P \wedge \bigvee_{S \subseteq \Omega} T[S/\hat{S}] \tag{6.5}$$

Theorem 66 *Formula (6.5) has size linear in $||T||$ and is logically equivalent to $T *_{Web} P$.*

Proof. The size of the formula is $O(||P|| + ||T||^{|\Omega|})$, hence it is linear in $||T||$ and exponential in $|\Omega|$. Observe that $|\Omega|$ is bounded by $|Var(P)| = k$, since Ω is the union of $K_{T,P}$, which is a subset of $Var(P)$ by Property 61. As for the other statement, the proof is identical to the one for the representation of Satoh's revision, substituting $K_{T,P}$ with Ω. □

6.4.3 Horn Case

In this section we analyze the results of existence of compact representation for the Horn case. The results are summarized in Table 6.2. The bounded Horn case is not reported, since all the revisions but Ginsberg's have a compact representation even if T and P are non-Horn, and Ginsberg's revision has the same properties if P is bounded or not. The table contains the usual three columns corresponding to the three forms of equivalence considered.

The negative result follows from the compilability results of model checking and query answering. For the negative results of the first two columns, we state the following theorem.

Theorem 67 *The revision operators by Dalal, Forbus, Borgida, Winslett, and Satoh have no compact representation w.r.t. logical and model equivalence.*

Proof. The model checking for all these operators is (at least) $\|\!\rightsquigarrow$NP hard, thus not in $\rightsquigarrow$P. Since operators that have a compact representation w.r.t. logical and model equivalence have a $\rightsquigarrow$P model checking, it follows that all the revision operators mentioned have no compact representation. □

A similar negative result can be stated for Forbus' revision operator w.r.t. the query equivalence.

	Horn Case		
	Logical equiv.	Model equiv.	Query equiv.
Ginsberg	?	YES	YES
Dalal	NO	NO	YES
Forbus	NO	NO	NO
Satoh	NO	NO	YES
Borgida/Winslett	NO	NO	YES
Weber	?	YES	YES
WIDTIO	YES	YES	YES

Table 6.2: Has $*$ a compact representation (Horn case)?

Theorem 68 *Forbus' revision has no compact representation w.r.t. query answering.*

Proof. Follows from the fact that Forbus' operator have a $\|\!\sim$coNP hard model checking, thus it is not in $\|\!\sim$NP. □

The positive result of existence of compact representation for Dalal's and Weber's revisions w.r.t. query equivalence follows from the result in the general (non-Horn) case. However, in the table there are many existence results that are not implied by the results in the general case. We prove only the existence of a compact representation for Weber's revision w.r.t. model equivalence.

We need first a result about circuits. Consider two sets of variables

$$\begin{aligned} B &= \{b_j^i \mid 1 \leq i \leq n,\ 1 \leq j \leq m\} \\ H &= \{h_j^i \mid 1 \leq i \leq n,\ 1 \leq j \leq m\} \end{aligned}$$

We give a one-to-one correspondence between the interpretations over $B \cup H$ and the sets of m clauses over a set of n variables. In other words, given a set of clauses, we want to associate a truth evaluation over $B \cup C$ with it.

Let $\Pi = \{\gamma_1, \ldots, \gamma_m\}$ be a set of clauses. The interpretation associated with Π is

$$m(\Pi) = \{b_j^i \mid \gamma_i \models \neg x_j\} \cup \{h_j^i \mid \gamma_i \models x_j\}$$

Roughly speaking, we use the value associated to the b_j^i's to represent the negative literals in the clauses, while the h_j^i represents the positive ones. For example, let $X = \{x_1, x_2, x_3\}$ be the set of variables, and $\Pi = \{x_1, \neg x_2 \vee x_3\}$ the set of clauses. The interpretation $m(\Pi)$ associated with Π is $\{b_2^2, h_1^1, h_3^2\}$. Note that $m(\Pi)$ is an interpretation over $B \cup H$ and *not an interpretation over*

X. In other words, $m(\Pi)$ is an interpretation over a different alphabet. We remark the fact that a set of clauses Π is associated to a single interpretation $m(\Pi)$ over a different alphabet, since this may result confusing in the sequel. Notice that the function m is one-to-one, thus given an interpretation $R \subseteq B \cup H$, there exists exactly one set of clauses Π such that $m(\Pi) = R$. We denote this set by $\Pi = m^{-1}(R)$ as usual.

We define s the function from interpretations over $B \cup H$ to truth values, defined as follows.

$$s(R) = \begin{cases} \mathsf{true} & \text{if } m^{-1}(R) \text{ is satisfiable} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

R is associated to a set of clauses: the function s tells whether the set of clauses is satisfiable or not.

For example, if $R = \{b_1^1, h_1^2\}$ then the corresponding set of clauses is $\{\neg x_1, x_1\}$, which is unsatisfiable. As a result, $s(R) = \mathsf{false}$. Another example is $S = \{b_1^1, h_2^1, h_1^2\}$: the set of clauses associated with S is $\{\neg x_1 \vee x_2, x_1\}$ which is satisfiable. Thus $s(S) = \mathsf{true}$.

Now, s is a function from the set of truth assignments on $B \cup H$ to $\{\mathsf{true}, \mathsf{false}\}$, thus it can be represented with a propositional formula over the alphabet $B \cup H$. This formula can be represented also as a circuit of polynomial size.

Lemma 11 *There exists a circuit* sat *(over the variables $B \cup H$) such that*

1. *Its size is polynomial in $|B| + |H|$.*
2. *$R \subseteq B \cup H$ is a model of* sat *if and only if $m^{-1}(R)$ is satisfiable and is a Horn set.*

Proof. Although it is possible to give a constructive proof (an explicit representation of sat), for the sake of simplicity we prove it indirectly.

From [DG84] follows that given a representation of a Horn set, it is possible to decide its satisfiability with a polynomial-time algorithm. Thus, given an interpretation $R \subseteq B \cup H$, it is possible to decide in polynomial time if the associated set of clauses is Horn and satisfiable. Thus, decide if R is a model of the circuit sat is polynomial. Now, [BS90, Theorem 2.2] shows that every polynomial-time algorithm can be encoded in a polynomial-size circuit. Thus, it is possible to represent sat with a circuit whose size is polynomial in $|B| + |H|$. □

This circuit sat has a model $R \subseteq B \cup H$ if and only if R is associated to a satisfiable set of Horn clauses.

The next theorem regards the existence of a compact representation w.r.t. model equivalence of Weber's revision.

Theorem 69 *There exists a compact representation w.r.t. model equivalence for Weber's revision.*

Proof. A model J of P is also a model of $T *_{Web} P$ if and only if there is a model of T that agrees with it at least on the variables not in Ω. Given a model $J \in Mod(P)$, this is equivalent to saying:

$$J \models T *_{Web} P \;\Leftrightarrow\; \text{the Horn set } \left[T \cup \bigcup_{x_i \notin \Omega \; x_i \in J} x_i \cup \bigcup_{x_i \notin \Omega \; x_i \notin J} \neg x_i \right] \text{ is satisfiable}$$

Let $T = \{\gamma_1, \ldots, \gamma_m\}$, $|X| = n$, and $|\Omega| = k$. We give a circuit that represents $T *_{Web} P$. Let

$$\begin{aligned} B &= \{b^i_j \mid 1 \leq i \leq n \;\; 1 \leq j \leq m+n\} \\ H &= \{h^i_j \mid 1 \leq i \leq n \;\; 1 \leq j \leq m+n\} \end{aligned}$$

be two sets of $n(m+n)$ variables each. Consider the circuit

$$T_1 = P \wedge \mathsf{sat} \wedge REPR_T \wedge REPR_X$$

where sat is the circuit of Lemma 11 over the variables $B \cup H$, $REPR_X$ and $REPR_T$ are the formulas

$$\begin{aligned} REPR_T &= \bigwedge\{b^i_j \mid \gamma_i \models \neg x_j\} \wedge \bigwedge\{\neg b^i_j \mid \gamma_i \not\models \neg x_j\} \wedge \\ &\quad \bigwedge\{h^i_j \mid \gamma_i \models x_j\} \wedge \bigwedge\{\neg h^i_j \mid \gamma_i \not\models x_j\} \\ REPR_X &= \bigwedge_{x_j \notin \Omega} \{(x_j = \neg b^{m+j}_j) \wedge (x_j = h^{m+j}_j)\} \wedge \\ &\quad \bigwedge_{x_j \notin \Omega} \{\neg b^{m+i}_j \wedge \neg h^{m+i}_j | i \neq j\} \wedge \bigwedge_{\substack{x_j \in \Omega \\ m+1 \leq i \leq m+n}} (b^i_j \wedge h^i_j) \end{aligned}$$

In words: $B \cup H$ is used to represents a set of $n+m$ clauses over a set of n variables; the formula $REPR_T$ says that the first m clauses represented by $B \cup H$ are the clauses in T, and the formula $REPR_X$ says that the last n clauses represented by $B \cup H$ must represent the value of the variables x_i's, i.e. the $n+i$'th clause is x_i if x_i is true, $\neg x_i$ otherwise. In other terms, a model of this circuit must have a value of the first $n \times m$ elements of B and H that represents T, while the other $n \times n$ must represents the value of x_j if $x_j \notin \Omega$, otherwise they must represent tautological clauses.

The sat circuit is used to test whether the set of clauses represented by $B \cup H$ is satisfiable. Now, $B \cup H$ is T together with a set of clauses representing the value in a model of P of the variables not in Ω. As a result, the whole circuit has a model K if and only if

1. $K \cap X \in \mathit{Mod}(P)$
2. the set of clauses $T \cup \bigcup_{x_i \notin \Omega \ x_i \in J} x_i \cup \bigcup_{x_i \notin \Omega \ x_i \notin J} \neg x_i$ is satisfiable.

which is equivalent to say that $K \cap X \in \mathit{Mod}(T *_{Web} P)$. This proves that given a model K of T_1, we can determine in polynomial time a model of $T *_{Web} P$, since $K \cap X$ is a model of $T *_{Web} P$. The converse also holds: given a model J of $T *_{Web} P$, we can complete it to obtain a model of T_1 in the following manner:

$$\begin{aligned} K \quad = \quad & J \cup \{b_j^i \mid \gamma_i \models \neg x_j\} \cup \{h_j^i \mid \gamma_i \models x_j\} \cup \\ & \{b_j^{m+j} \mid x_j \notin J,\ x_i \notin \Omega\} \cup \{h_j^{m+j} \mid x_j \in J,\ x_i \notin \Omega\} \end{aligned}$$

Since this model can be determined in polynomial time, the claim is proved. □

Chapter 7

Succinctness

In this chapter we investigate the *space efficiency* of a Propositional Knowledge Representation (PKR) formalism. Informally, the space efficiency of a formalism F in representing a certain piece of knowledge α, is the size of the shortest formula of F that represents α. We assume that knowledge is either a set of propositional interpretations or a set of formulae (theorems). We provide a formal way of talking about the relative ability of PKR formalisms to compactly represent a set of models or a set of theorems. We introduce two new compactness measures, the corresponding classes, and show that the relative space efficiency of a PKR formalism in representing models/theorems is directly related to such classes. In particular, we consider formalisms for nonmonotonic reasoning, such as circumscription and default logic, as well as belief revision operators.

7.1 Motivations

During the last years a large number of formalisms for knowledge representation (KR) have been proposed in the literature. Given some (informal) knowledge of a domain, a knowledge engineer has to choose the most appropriate KR formalism to represent it. Formalisms can be chosen for their semantics, the complexity of inference, or other properties. We investigate their *space efficiency*. Informally, the space efficiency (or *succinctness*) of a formalism F is its ability to represent knowledge in little space. For this reason, the succinctness of a formalism is also called compactness. A formalism is more succinct than another if it can represent the same information in a more compact way (that is, with a shorter representation).

Let us consider the following example about the relationship between circumscription and propositional logic.

Example 3 *Your paper has been reviewed by three referees, and the editor*

of the journal told you that two of them agreed to reject the paper, while the other one does not. Let us formalize this information in propositional logics: the variable x_i represents the fact that the reviewer i like the paper. In propositional logic, the knowledge base can be represented for example as:

$$(\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3)$$

Using circumscription the same information can be represented as

$$CIRC(x_1 \vee x_2 \vee x_3; \{x_1, x_2, x_3\}, \emptyset, \emptyset)$$

In this case, circumscription is better than propositional logic, since it can represent the same information in less space.

It can be proved that (unless the polynomial hierarchy collapses) there are formulas that can be polynomially expressed using circumscription, but cannot in propositional logics.

Here we focus on propositional knowledge representation (PKR) formalisms, and assume that knowledge is either a set of propositional interpretations or a set of formulae (theorems). Consequently, we consider a PKR reasoning problem to be either model checking, or theorem proving. The basic definitions of succinctness etc. appeared for the first time in [CDLS96a].

Conceptually similar investigations are made in the field of relational database query languages. In that field one of the goals is to know exactly what information it is possible to extract from a database by means of a query. The typical result of such investigations characterizes the *expressiveness* of a query language, often saying that language A captures more/less/same queries than/as language B. Sometimes such results are conditional to non-collapse of some complexity classes.

7.1.1 State of the Art.

Most PKR formalisms are "translatable" one into another, although such translation may lead to an exponential increase of the size of the formula. Most of the translations have Propositional Logic (PL) as their target formalism. E.g., [BED91] from default logic to PL, [BED94] from disjunctive logic programs to PL, [Win89] from revised knowledge bases to PL, [GPP89] from circumscription to PL.

Only very recently researchers started analyzing the space efficiency of PKR formalisms; this kind of investigation includes questions such as "is exponential increase of the above mentioned translations intrinsic, or is it possible to design a polynomial-size translation?" In [CDS96] it was shown that many interesting fragments of default logic and circumscription cannot be expressed

by polynomial-time fragments of PL without super-polynomially increasing the size of formulae. It was proven that super-polynomial increase of the size is necessary when translating unrestricted propositional circumscription [CDSS97] into PL. The results of Chapter 6 can be seen as results on the possibility/impossibility of translating most operators for belief revision into PL. In [GKPS95] Gogic, Kautz, Papadimitriou and Selman analyzed the relative succinctness of several PKR formalisms in representing sets of models. Among other results, they showed that skeptical default logic represents sets of models more succinctly than circumscription. Unfortunately, the above results are based on ad-hoc proofs and do not help us to define equivalence classes for the space efficiency of KR formalisms.

In Chapter 6, however, the results of non-translability of belief revision into PL are proved by directly using result of non-compilability of these operators. In this chapter we show how the compilability results can be directly used to characterize the space efficiency of PKR formalisms.

7.1.2 Goal.

In KR the notion of *polynomial-time solvability* models the concept of *tractable* reasoning problem. Analogously, the notion of *polynomial many-one reducibility* models the relation existing between two reasoning problems whose time complexity is comparable. The latter notion allows one to say, e.g. that inference in PL is one of the hardest problems among those in coNP. Our goal is to provide a formal way of talking about the relative ability of PKR formalisms to compactly represent information, where the information is either a set of models or a set of theorems. In particular, we would like to be able to say that a specific PKR formalism provides "one of the most compact ways to represent models/theorems" among the PKR formalisms of a specific class.

7.1.3 Results.

We introduce two new compactness measures (*model* and *theorem compactness*) and the corresponding classes (model-C and thm-C, where C is a complexity class like P, NP, coNP, etc.). Such classes form two hierarchies that are isomorphic to the polynomial hierarchy. We show that the relative space efficiency of a PKR formalism is directly related to such classes. In particular, the ability of a PKR formalism to compactly represent sets of models/theorems is directly related to which classes of the model/theorem hierarchy it belongs to. Problems higher up in the model/theorem hierarchy can represent sets of models/theorems more compactly than formalisms that are in lower classes.

This classification is obtained through a general framework, and not by making direct comparisons, and ad-hoc proofs, between the various PKR for-

malisms. Furthermore, our approach also allows for a simple and intuitive notion of completeness for both model and theorem hierarchies. This notion precisely characterizes both the relation between formalisms at different levels, and the relations between problems at the same level. An interesting result is that two PKR formalisms in which model checking or inference belong to the same time complexity may belong to different compactness classes. This may suggest a criterion for choosing between two PKR formalisms in which reasoning has the same time complexity—namely, choose the more compact t one. Also, two PKR formalisms may belong to the same theorem compactness class, yet to different model compactness classes. This stresses the importance of first clarifying whether we want to represent models or theorems when choosing a PKR formalism.

7.2 Propositional KR Formalisms

Let us formally state what we mean for a propositional knowledge representation formalism (PKR). We consider a finite alphabet of propositional symbols $L = \{a, b, c, \ldots\}$, possibly with subscripts. We admit connectives of fixed arity for constructing well-formed formulae, as an example, $\wedge$ is the standard binary conjunction, the symbol for defaults $\frac{\cdot}{\cdot}$ of default logic [Rei80] is a ternary connective, etc. Since we restrict to propositional formalisms, we admit neither variable symbols, nor quantifiers. We distinguish between two kinds of formulae: *knowledge bases* and *queries*. The languages describing well-formed knowledge bases and well-formed queries may be different within the same formalism, e.g. this is the case for default logic, or logic programming. An *interpretation* for L is a mapping from L in $\{\mathsf{true}, \mathsf{false}\}$. A *model-theoretic semantics* for a formalism is a description of how to extend an interpretation for L to well-formed knowledge bases and queries. Observe that the semantics of a formula needs not to be obtained in an easy way from the semantics of its components and connectives: E.g., for skeptical default logic an interpretation I assigns true to a default theory $\langle D, W\rangle$ iff there is a (propositional) extension of $\langle D, W\rangle$ such that I assigns true to all of its formulae.

Definition 41 *A* model *of a knowledge base KB in a formalism F is an interpretation M that maps KB to* true *(written $M \models_F KB$)*

The same for a query Q (written $M \models_F Q$). Sometimes models will be denoted as sets of letters which are mapped into true.

Definition 42 *A* proof theory *of a formalism F is a definition of which queries are derivable from a knowledge bases in F.*

When a query Q is derivable from a knowledge base KB in F, we call Q a *theorem* of KB (written $KB \vdash_F Q$). Observe that some formalisms have only a proof theory, with no model-theoretic semantics, e.g. credulous default logic. When a formalism F has both a model-theoretic semantics and a proof theory, we impose the usual relation between them: $KB \vdash_F Q$ iff $\forall M : M \models_F KB$ implies $M \models_F Q$. When F is classical propositional logic PL, we omit the subscript from $\vdash$ and $\models$.

Assumption 1 *The information represented by a knowledge base in a PKR formalism is either its set of models or its set of theorems (or both).*

In this way, all PKR formalisms can be compared on the basis of which sets of models or theorems they can represent, and how succinct is the representation.

Assumption 2 *We consider only queries whose size is less than or equal to that of the knowledge base.*

As a consequence, propositional tautologies whose size is larger than that of the knowledge base are not considered among the theorems.

7.3 Succinctness of PKR Formalisms

Let us informally state what is the meaning of "succinctness". Let F_1 and F_2 be two formalisms. A certain piece of information can be represented in both formalisms, but with different sizes. For example, if a piece of information can be represented in F_2 with size k, while F_1 requires at least 2^k to represent the same information, then F_2 is more succinct in representing that information.

This way we are able to compare the representation of a *specific* piece of information. What we would like to define is a general way to compare the ability of a formalisms in representing *any* kind of information in a small amount of space.

Let us consider the formal definitions of propositional knowledge representation formalisms given so far. A formalism F is essentially composed by a proof theory and/or a model theoretic semantics, that is, by two binary relations $\vdash_F$ and $\models_F$. The relation $\vdash_F$ specifies the theorems of a knowledge base, while $\models_F$ specifies its models.

As a result, a propositional knowledge representation formalisms represents information as sets of models and/or sets of theorems. As a result, we have two ways of comparing formalisms, one considering the information represented by a knowledge base as its set of models, and the one in which the information is the set of theorems. Let us consider the former. From what said above, a tentative definition may be the following.

Definition 43 (Succinctness: first attempt) *The formalisms F_2 is at least as succinct as F_1, if, for each piece of information (set of models), the size of its shortest representation in F_2 is bounded by a polynomial in the size of its shortest representation in F_1.*

We can reformulate this definition as: given a knowledge base KB in F_1, the corresponding knowledge base in F_2 that has the same set of models, must be polynomially bounded by the size of KB. This is just a different way to express the same definition.

For example, circumscription can be translated to default logics as follows. Let T be a formula, and suppose that all the variables X in T have to be minimized. Then, $CIRC(T, X, \emptyset, \emptyset)$ has the same set of models of the default theory $\langle \{ \frac{:\neg x_i}{\neg x_i} \mid x_i \in X \}, T \rangle$.

This definition imposes that the models in the two formalisms must be exactly identical. However, there are cases in which there exists a way to express the same set of models in two formalisms, but not exactly in the same way. Consider for example the reduction from prerequisite-free normal default theory to circumscription [Eth87]. A prerequisite-free normal default is a default of the form $\frac{:P}{P}$. A default theory whose default rules are all prerequisite-free normal default can be translated to circumscription as follows. Let $\langle \{ \frac{:P_i}{P_i} \}, W \rangle$ be the default theory on the alphabet X. Using circumscription we can obtain $K = CIRC(W \wedge \{y_i \equiv P_i\}; Y, X, \emptyset)$ where Y is a set of new variables, one-to-one with X. The formula K has an interesting property: it has not the same models of the default theory, but each of its models can be polynomially obtained from a model of the default theory. Indeed, given a model I on the alphabet X, this is a model of the default theory if and only if $I \cup \{y_i \mid I \models P_i\}$ is a model of the circumscription.

As a result, once translated a knowledge base from default to circumscription, a model on the default theory can always be expressed as a model on the corresponding circumscription. This is a case in which our definition should consider circumscription as succinct as prerequisite-free normal default logic. However, this translation does not prove succinctness according to Definition 43. This is why we need a weaker form of reduction, that allows models to be different between the two formalisms.

7.4 Reductions among KR Formalisms

We base the definition of succinctness of formalisms on a notion of reduction. The idea is that, given a knowledge base expressed in one formalism, there must be another one in the other formalism, whose size is polynomial in the size of the first one. This reduction is thus a polysize function. Given two PKR formalisms F_1 and F_2, we define a reduction between them as a polysize

function f such that, for each given knowledge base KB in F_1, the function f applied to KB gives a knowledge base in F_2 that represents the same information. We denote the fact that f translate knowledge bases from F_1 to F_2 as $f : F_1 \mapsto F_2$.

Definition 44 *A reduction between two PKR formalisms F_1 and F_2 is a polysize function f such that, for each knowledge base KB in F_1, $f(KB)$ is a knowledge base in F_2 that represents the same information.*

The problem is that "... represents the same information" is an informal sentence. What is needed is a formal definition of equivalence between two knowledge base expressed in two different formalisms (in this case, the two knowledge bases to compare are KB and $f(KB)$). The same problem has been seen for the problem of compact representations, which is a specific case of succinctness. In the same way we did there, we use model equivalence and theorem equivalence for the definition of reduction.

Definition 45 (Model Preservation) *A polysize reduction $f : F_1 \mapsto F_2$ satisfies* model-preservation *if for each knowledge base KB in F_1 there exists a polynomial function g_{KB} such that for every interpretation M of the variables of KB it holds that $M \models_{F_1} KB$ iff $g_{KB}(M) \models_{F_2} f(KB)$.*

Let us consider the example of reduction of skeptical default logic to circumscription.

Example 4 *Let $\langle D, W\rangle$ be a prerequisite-free normal default theory. Let f and $g_{\langle D,W\rangle}$ be*

$$\begin{aligned} f(\langle D, W\rangle) &= W \wedge \{y_i \equiv P_i \mid P_i \in D\} \\ g_{\langle D,W\rangle}(I) &= I \cup \{y_i \mid I \models P_i\} \end{aligned}$$

Since, for each interpretation I, it holds that I models (under the default semantics) $\langle D, W\rangle$ if and only if $g_{\langle D,W\rangle}(I)$ models (under circumscriptive semantics) $f(\langle D, W\rangle)$, then f is a model-preserving reduction from prerequisite-free normal default theories to circumscription.

A form of reduction where theorems are preserved is the following one.

Definition 46 (Theorem Preservation) *A polysize reduction $f : F_1 \mapsto F_2$ satisfies* theorem-preservation *if for each knowledge base KB in F_1 there exists a polynomial function g_{KB} such that for every query Q on the variables of KB it holds that $KB \vdash_{F_1} Q$ iff $f(KB) \vdash_{F_2} g_{KB}(Q)$.*

An example of theorem-preserving (and non-model-preserving) polysize reduction from updated knowledge bases to PL is given in [Win89]. The reduction shown in the previous example is also theorem-preserving, using for g_{KB} the identity function.

We remark that our definitions of reduction are more general than those proposed in [GKPS95]. In particular, in [GKPS95] only a notion analogous to Definition 45 is considered, and only for the case when g_{KB} is the identity – i.e. models in the two formalisms should be identical.

7.5 Compilability and Succinctness

In this section we show how to use our compilability classes to compare the succinctness of PKR formalisms. Let F_1 and F_2 be two formalisms representing sets of models. We prove that any knowledge base in F_1 can be reduced, via a polysize reduction, to a knowledge base in F_2 satisfying model-preservation if and only if the compilability class of the problem of *model checking* (first input: KB, second input: interpretation) in F_1 contains the compilability class of the problem of model checking in F_2.

Similarly, we prove that theorem-preserving polysize reductions exist if and only if the compilability class of the problem of *inference* (first input: KB, second input: query) in F_1 contains the compilability class of the problem of inference in F_2.

In order to simplify the presentation and proof of the theorems we introduce some definitions.

Definition 47 (Model hardness and completeness) *Let F be a PKR formalism. If the problem of model checking for F belongs to the compilability class $\|\!\sim$C, we say that F is in* model-C*. Similarly, if model checking is $\|\!\sim$C complete (hard), we say that F is* model-C *complete (hard).*

Definition 48 (Theorem hardness and completeness) *If the problem of inference for the formalism F belongs to the compilability class $\|\!\sim$C, we say that F is in* thm-C*. Similarly, if inference is $\|\!\sim$C complete (hard), we say that F is* thm-C *complete (hard).*

These definitions implicitly define two hierarchies, the model hierarchy (model-C) and the theorem hierarchy (thm-C). As an example rephrased from [CDS96, Thm. 2], we characterize model and theorem classes of propositional logic (PL).

Theorem 70 *PL is in* model-P *and* thm-coNP *complete.*

We can now formally establish the connection between succinctness of representations and compilability classes. In the following theorems, the complexity classes C, C_1, C_2 belong to the polynomial hierarchy. As usual, we assume that the polynomial hierarchy does not collapse.

Theorem 71 *Let F_1 and F_2 be two PKR formalisms. If F_1 is* model-C *and F_2 is* model-C*-hard, then there exists a polysize reduction $f : F_1 \mapsto F_2$ satisfying model preservation.*

Proof. Recall that since F_1 is model-C complete, model checking in F_1 is in $\|\!\rightharpoonup\!\mathrm{C}$, and since F_2 is model-C-hard, model checking in F_1 is non-uniformly comp-reducible to model checking in F_2. That is, there exist two polysize binary functions f_1 and f_2, and a polynomial-time binary function g such that for every pair $\langle KB, M\rangle$ it holds that $M \models_{F_1} KB$ if and only if $g(f_2(KB, ||M||), M) \models_{F_2} f_1(KB, ||M||)$. Now observe that $||M||$ can be computed from KB by simply counting the letters appearing in KB; let f_3 be such a counting function, i.e., $||M|| = f_3(KB)$. Clearly, f_3 is polysize. Define the reduction f as $f(KB) = f_1(KB, f_3(KB))$. Since polysize functions are closed under composition, f is polysize. We show that f is a model-preserving reduction. In fact, given KB, we can compute $z = f_2(KB, ||M||) = f_2(KB, f_3(KB))$. Since f_2 and f_3 are polysize, z has polynomial size w.r.t. $||KB||$, hence w.r.t. $||M||$. Define $h_{KB}(M) = g(z, M)$. Clearly, h_{KB} is a polytime function, and from its construction, $M \models_{F_1} KB$ iff $h_{KB}(M) \models_{F_2} f(KB)$. □

Theorem 72 *Let F_1 and F_2 be two PKR formalisms. If F_1 is* model-C_1*-hard, F_2 is in* model-C_2*, and $\mathrm{C}_2 \subset \mathrm{C}_1$, then there is no polysize reduction $f : F_1 \mapsto F_2$ satisfying model preservation.*

Proof. We show that if such a reduction exists, then $\mathrm{C}_1/\mathrm{poly} \subseteq \mathrm{C}_2/\mathrm{poly}$. Let A be a complete problem for class C_1. We recall the that the problem $*A$ defined as $*A = \{\langle x, y\rangle \mid y \in A\}$ is $\|\!\rightharpoonup\!\mathrm{C}_1$ complete. Since model checking in F_1 is model $-$ C_1-hard, it follows that $*A$ is non-uniformly comp-reducible to model checking in F_1. That is, there exist two polysize binary functions f_1 and f_2, and a polynomial-time binary function g such that for every pair $\langle x, y\rangle$, it holds $\langle x, y\rangle \in *A$ if and only if $g(f_2(x, ||y||), y) \models_{F_1} f_1(x, ||y||)$. As a result, $y \in A$ if and only if $g(f_2(\epsilon, ||y||), y) \models_{F_1} f_1(\epsilon, ||y||)$.

Let $||y|| = n$. The knowledge base $f_1(\epsilon, ||y||)$ depends just on n, i.e., there is one knowledge base for each integer. Call it KB_n. Moreover, also $f_2(\epsilon, ||y||) = f_2(\epsilon, n)$ depends just on n: call it O_n. Observe that O_n has polynomial size w.r.t. n.

If there exists a polysize reduction $f : F_1 \mapsto F_2$ satisfying model preservation, then given the knowledge base KB_n there exists a polynomial function h_n such that $g(O_n, y) \models_{F_1} KB_n$ if and only if $h_n(g(O_n, y)) \models_{F_2} f(KB_n)$.

Therefore, the $\|\!\sim\!\mathrm{C}_1$ complete problem $*A$ can be non-uniformly reduced to a problem in $\|\!\sim\!\mathrm{C}_2$ as follows: Given y, from its size $||y|| = n$ one obtains (with an arbitrary preprocessing) $f(KB_n)$ and O_n. Then one checks whether the interpretation $h_n(g(O_n, y))$ (computable in polynomial given y and O_n) is a model in F_2 for $f(KB_n)$. From the fact that model checking in F_2 is in $\|\!\sim\!\mathrm{C}_2$, we have that $\|\!\sim\!\mathrm{C}_1 \subseteq \|\!\sim\!\mathrm{C}_2$, thus $\mathrm{C}_1/\mathrm{poly} \subseteq \mathrm{C}_2/\mathrm{poly}$. □

The above theorems show that the hierarchy of classes model-C exactly characterizes the space efficiency of a formalism in representing sets of models. In fact, two formalisms at the same level in the model hierarchy can be reduced one into the other via a polysize reduction (Theorem 71), while there is no polysize reduction from a formalism (F_1) higher up in the hierarchy into one (F_2) in a lower class (Theorem 72). In the latter case we say that F_1 is *more space-efficient* than F_2.

Analogous results (with similar proofs) hold for polysize reductions preserving theorems.

Theorem 73 *Let F_1 and F_2 be two PKR formalisms. If F_1 is* thm-C *complete and F_2 is* thm-C*-hard, then there exists a polysize reduction $f : F_1 \mapsto F_2$ satisfying theorem preservation.*

Theorem 74 *Let F_1 and F_2 be two PKR formalisms. If F_1 is* thm-C_1*-hard, F_2 is in* thm-C_2*, and $\mathrm{C}_2 \subset \mathrm{C}_1$, then there is no polysize reduction $f : F_1 \mapsto F_2$ satisfying theorem preservation.*

Theorems 71-74 show that compilability classes characterize very precisely the relative capability of PKR formalisms to represent sets of models or sets of theorems. For example, as a consequence of Theorems 70 and 74 there is no polysize reduction from PL to Horn clauses that preserves the theorems unless the polynomial hierarchy collapses. Kautz and Selman proved non-existence of such a reduction for a problem strictly related to ci in [KS92] using an ad-hoc proof.

7.6 Belief Revision and Circumscription

In Chapter 6 we analyzed the problem of finding a propositional formula that is equivalent to the result of a revision. We introduced three forms of equivalence: logical, model, and query. These results are directly related to the succinctness of formalisms. For instance, since Dalal's revision has a compact representation w.r.t. query equivalence, this means that a knowledge base expressed as an initial piece of information and an update can be translated into a single propositional formula that has the same implied formulas.

This is a translation between two propositional knowledge representation formalisms: from belief revision to propositional logics. The target of the translation is the propositional calculus. This idea can be extended to other target formalisms. In this section we show how belief revision can be translated to circumscription.

What we would like to obtain is a way to translate a knowledge base expressed as a pair $\langle T, P \rangle$ into a theory T_C such that $T * P$ is equivalent to $CIRC(T_C; X, Y, Z)$. As in Chapter 6, there are three forms of equivalence: logical, model, and query. The positive results are always obtained by directly proving that such a translation exists. On the other hand, the existence of such translation proves that the target formalism is at least as succinct as the initial one. As a result, if the original formalisms is in a higher succinctness class than the target one, such a translation is impossible.

Let us formally state the idea of the "translability" between formalisms.

Definition 49 *A translation from the formalism F_1 to the formalism F_2 is a polysize function f from the knowledge bases of F_1 to the knowledge bases in F_2.*

This translation, to be useful, must translate each knowledge base into an equivalent one. In the previous section we defined the equivalence on the basis of the existence of a function g_{KB} (depending on the original knowledge base), that translates the models (or the queries) of the first formalism into models (queries) of the second one. In order to capture the definitions of logical/model/query equivalence, the function g_{KB} must have some additional properties.

Definition 50 *A translation f from F_1 to F_2 respects logical equivalence if, for each interpretation I and knowledge base K, it holds $I \models_{F_1} K$ if and only if $I \models_{F_2} f(K)$.*

This is equivalent to the definition of succinctness, but the function g_{KB} is the identity function. This definition has been introduced in [GKPS95]. If F_2 is propositional logics, then there exists a translation from F_1 to F_2 that respects logical equivalence if and only if there exists a compact representation w.r.t. logical equivalence for F_1.

In a similar manner, a translation may respect model and query equivalence.

Definition 51 *A translation f from F_1 to F_2 respects model equivalence (over a set of variables X) if, for each interpretation I over the alphabet X one can find in polynomial time an interpretation J such that a. $I = J \cap X$, and b. $I \models_{F_1} K$ if and only if $J \models_{F_2} f(K)$, for any knowledge base K.*

Definition 52 *A translation f from F_1 to F_2 respects query equivalence (over a set of variables X) if, for each query Q over the alphabet X if holds $K \vdash_{F_1} Q$ if and only if $f(K) \vdash_{F_2} Q$, for any knowledge base K.*

Where not else specified, we assume that X is the set of variables over which the knowledge base in F_1 is built. These are straightforward extension of the concept of compact representation w.r.t. logical, model, and query equivalence, in the case the formalism that is the target of the translation is not the propositional logics. If F_2 is PL, then there exists a translation from F_1 to F_2 that respect logical equivalence if and only if there exists a compact representation w.r.t. logical equivalence for F_1. The same for model and query equivalence.

The following theorems extend Theorems 53, 54, 55.

Theorem 75 *If there exists a translation from F_1 to F_2 that respects logical or model equivalence, then the model class of F_1 is contained in the model class of F_2.*

This theorem is useful (as Theorems 53,54) to prove that a translation does not exist: if the model class of F_1 is not contained in the model class of F_2, then there is no translation from F_1 to F_2 that respects logical or model equivalence. A similar theorem holds for the queries.

Theorem 76 *If there exists a translation from F_1 to F_2 that respects query equivalence, then the theorem class of F_1 is contained in the theorem class of F_2.*

This theorem will also be used to show that in some cases a translation does not exist. Note that PL is model-P and thm-coNP, and this is why Theorems 53,54,55 compare the classes of model checking and query answering of a formalism to $\rightsquigarrow$P and compcoNP, respectively.

In [LS97] a complete set of translations from several belief revision operators and circumscription is provided. As a case study, we consider the translation from Satoh's revision to circumscription. First of all, model checking of Satoh's revision is $\|\!\rightsquigarrow\Sigma_2^p$ hard, while model checking for circumscription is compcoNP. An easy consequence is the following theorem.

Theorem 77 *There is no translation from Satoh's revision to circumscription that respect logical or model equivalence.*

Proof. The existence of such a translation would imply that the model class of Satoh's revision is contained in the model class of circumsription. But circumscription is model-coNP, while model checking for Satoh's revision is

$\|\!\rightarrow \Sigma_2^p$ hard, thus not in compcoNP, thus the model class of Satoh's revision is not contained in the class of circumscription. □

On the other hand, it is possible to give a translation from Satoh's revision to circumscription that respect query equivalence.

Theorem 78 *For each T, P, and Q, it holds $T *_S P \models Q$ if and only if $CIRC(T_S; W, \emptyset, X \cup Y) \models Q$, where*

$$T_S = T[X/Y] \wedge P \wedge (\neg W \equiv (X \equiv Y))$$

($T[X/Y]$ is the formula obtained by replacing each $x_i \in X$ with $y_i \in Y$ in T).

Proof. Let Q be a formula such that $Var(Q) \subseteq X$. We first show that $CIRC(T_S; W, \emptyset, X \cup Y) \models Q$ implies that $T *_S P \models Q$. Assume that $CIRC(T_S; W, \emptyset, X \cup Y) \models Q$ and $T *_S P \not\models Q$. Thus, there exists a model M_X of $T *_S P$ such that $M_X \not\models Q$. Let M'_X be a model of T such that $Diff(M'_X, M_X) \in K_{T,P}$. We define $M_Y = \{y_i | x_i \in M'_X\}$ and $M_W = \{w_i | ((x_i \in M_X) \text{ and } (y_i \notin M_Y)) \text{ or } ((x_i \notin M_X) \text{ and } (y_i \in M_Y))\}$. Now, let $M = M_X \cup M_Y \cup M_W$. Obviously, it holds that $M \models T_S$ and $M \not\models Q$. If M is a minimal model of T_S the thesis follows, so assume that there exists a model N of T_S such that N is less than M. Since N is a model of T_S and it cannot contain more literals of W than M, we have that $N_W = (N \cap W) \subset M_W$. Hence, the distance between $N_X = N \cap X$ and $N_Y = N \cap Y$ is smaller than the distance between M_X and M_Y. Thus, M_X is not one of the models of P closest to the models of T. As a consequence, M_X is not a model of $T *_S P$ and contradiction arises.

We now show that $T *_S P \models Q$ implies that $CIRC(T_S; W, \emptyset, X \cup Y) \models Q$. Assume that $T *_S P \models Q$ and $CIRC(T_S; W, \emptyset, X \cup Y) \not\models Q$. Thus, there exists a model M of $CIRC(T_S; W, \emptyset, X \cup Y)$ such that $M \not\models Q$. Let $M_X = M \cap X$, we show that $M_X \models T *_S P$. It immediately follows that $M_X \models P$, if M_X is one of the models of P closer to models of T the thesis follows, so assume to the contrary that there exists a $N_X \subseteq X$, different from M_X, such that $N_X \models P$ and the distance of M_X from the closest model of T is a strict superset of the distance of N_X from its closest model of T, Let N'_X be a model of T such that $N'_X \Delta N_X \in K_{T,P}$. Let $N_Y = \{y_i | x_i \in N'_X\}$, $N_W = \{w_i | ((x_i \in N_X) \text{ and } (y_i \notin N_Y)) \text{ or } ((x_i \notin N_X) \text{ and } (y_i \in N_Y))\}$ and $N = N_X \cup N_Y \cup N_W$. Obviously N is a model of $T[X/Y] \wedge P \wedge (\neg W \equiv (X \equiv Y))$ moreover, N is less than M. Hence, M is not a minimal model of $T[X/Y] \wedge P \wedge (\neg W \equiv (X \equiv Y))$, hence contradiction arises. □

Corollary 3 *There exists a translation from Satoh's revision to circumscription that respect query equivalence.*

Proof. Let us formally define the formalisms of revision and circumscription. A knowledge base for the revision formalism is a pair $\langle T, P\rangle$. The entailment is defined as $\langle T, P\rangle \vdash_S Q$ if and only if $T *_S P \models Q$. A knowledge base in the circumscription formalisms is a 4-tuple $\langle T, X, Y, Z\rangle$, where T is a propositional formula and X, Y, and Z are three set of mutually disjoint sets of variables, such that $Var(T) \subseteq X \cup Y \cup Z$. The entailment is defined as $\langle T, X, Y, Z\rangle \vdash_C Q$ if and only if $CIRC(T; X, Y, Z) \models Q$.

The translation is the following:

$$f(\langle T, P\rangle) = \langle T_S, W, \emptyset, X \cup Y\rangle$$

The previous theorem can be rephrased as: $\langle T, P\rangle \vdash_S Q$ if and only if $\langle T_S, W, \emptyset, X \cup Y\rangle \vdash_C Q$, if $Var(Q) \subseteq X$. As a result, $K \vdash_S Q$ if and only if $f(K) \vdash_C Q$, and thus f is a translation from Satoh's revision to circumscription that respect query equivalence. □

Other translations between belief revision and circumscription can be found in [LS97]. The positive results are summarized in Table 7.1.

$T *_S P$	$\Rightarrow$	$CIRC(T[X/Y] \wedge P \wedge (\neg W \equiv (X \equiv Y)); W, \emptyset, X \cup Y)$
$T *_W P$	$\Rightarrow$	$CIRC(T[X/Y] \wedge P \wedge (\neg W \equiv (X \equiv Y)); W, Y, X)$
$T *_B P$	$\Rightarrow$	$CIRC([T \vee (Y \equiv R)] \wedge T[X/Y] \wedge P \wedge$ $[\neg W \equiv (X \equiv Y)]; W, R, X \cup Y)$
$T *_F P$	$\Rightarrow$	$CIRC(T[X/Y] \wedge P \wedge (\neg W \equiv (X \equiv Y)) \wedge EQ(W, V) \wedge$ $BEGIN(V); V, Y, X \cup W)$
$T *_G P$	$\Rightarrow$	$CIRC(P \wedge \bigwedge_{f_i \in T}(y_i \equiv f_i); Y, \emptyset, X)$

$EQ(W, V)$ is a polynomial-size formula that is true if and only if W and V have exactly the same number of positive literals.

$BEGIN(V)$ states that the positive literals of V are its first ones.

Table 7.1: Translations from belief revision operators and circumscription

It can also be proved that all the AGM revision operators can be translated to circumscription, provided that the family of preorderings over interpretations is given, and the comparison between two interpretation can always be determined in polynomial time (see [LS97] for details).

Chapter 8

Related Work

In this chapter we relate our definition of compilability with previous work. Namely, we compare our classes of compilability with the non-uniform hierarchy by Karp and Lipton [KL80] and with the theory of fixed parameter tractability by Downey and Fellows [DF95b, DF95a].

8.1 Non-Uniform Hierarchy

In this section we compare our classes of non-uniform compilability with the non-uniform complexity hierarchy by Karp and Lipton [KL80]. The main conceptual difference is that while our classes are defined for problems of pairs (where one part of the instance are allowed to be preprocessed) in the definition of the non-uniform classes of Karp and Lipton the instance of the problem is considered as a whole.

Let us first recall the definition of non-uniform classes. We use the terminology of the previous chapters, but the definitions are equivalent to the original ones. Let C be a class of the polynomial hierarchy. The class C/poly is defined as follows.

Definition 53 *A problem A belongs to* C/poly *if there exists a polysize function f and a* C *problem S such that*

$$x \in A \quad \textit{iff} \quad \langle f(||x||), x\rangle \in S$$

The main difference between C/poly and $\|\!\rightarrowtail$C is that the polysize function takes as arguments not only the total size of the instance, but also the fixed part of the input. As a result, the class $\|\!\rightarrowtail$C is larger than C/poly.

Theorem 79 *It holds* C/poly $\subseteq \|\!\rightarrowtail$C.

Proof. Let A be a problem of pairs of strings, that is $A \subseteq \Sigma^* \times \Sigma^*$, as usual (the class $\|\!\rightharpoonup$C is defined only for problem of pairs). When A is a problem of pairs, the definition of C/poly becomes: there exists a polysize function f and a C language S such that $\langle x, y\rangle \in A$ if and only if $\langle f(||x|| + ||y||), \langle x, y\rangle\rangle \in S$. Let us define

$$\begin{aligned} f'(x, n) &= \langle ||x|| + n, x\rangle \\ S' &= \{\langle\langle k, x\rangle, y\rangle \mid \langle k, \langle x, y\rangle\rangle \in S\} \end{aligned}$$

It holds $\langle x, y\rangle \in A$ if and only $\langle f'(x, ||y||), y\rangle \in S'$. Since f' is polysize and S' is in C, this proves that A is in $\|\!\rightharpoonup$C. □

On the other hand, C/poly does not coincide with $\|\!\rightharpoonup$C, unless the former contains all the possible problems.

Theorem 80 *If there exists a problem that does not belong to* C/poly *then* C/poly $\not\equiv \|\!\rightharpoonup$C.

Proof. Let A be the problem of strings (not pairs) that does not belong to C/poly. Let us consider the problem $A*$ defined as

$$A* = \{\langle x, y\rangle \mid x \in A\}$$

This is the converse of the class $*A$, as x must be in A, while y can be any string. We prove that $A*$ is not in C/poly, but is in $\|\!\rightharpoonup$C.

Suppose that $A*$ is in C/poly. By definition there exists a polysize function f and a C language S such that $\langle x, y\rangle \in A*$ if and only if $\langle f(||x|| + ||y||), \langle x, y\rangle\rangle \in S$. Let f' and S' be defined as follows.

$$\begin{aligned} f'(k) &= f(k) \\ S' &= \{\langle k, y\rangle \mid \langle \epsilon, y\rangle \in S\} \end{aligned}$$

The function f' is polysize while the problem S' is in C. Furthermore, it holds $x \in A$ if and only if $\langle x, \epsilon\rangle \in A*$, which holds if and only if $\langle f(||x|| + ||\epsilon||), \langle x, \epsilon\rangle\rangle \in S$, which is in turn equivalent to $\langle f'(||x||), x\rangle \in S'$. This proves that A is in C/poly. Since this is false by hypothesis, the assumption $A* \in$ C/poly is also false.

We prove that $A*$ is instead in $\|\!\rightharpoonup$C. Let f and S be defined as

$$\begin{aligned} f(x, k) &= \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases} \\ S &= \{\langle 1, y\rangle \mid \text{any } y\} \end{aligned}$$

It holds $\langle x, y\rangle \in A*$ if and only if $\langle f(x, ||y||), y\rangle \in S$. Since f is polysize (its result is always composed by a single bit) and S is a constant-time problem, it holds $A* \in \|\!\rightharpoonup\!\mathrm{C}$. □

Finally, a relation between the collapse of the non-uniform compilability hierarchy and the collapse of the non-uniform complexity hierarchy.

Theorem 81 *It holds* $\|\!\rightharpoonup\!\mathrm{C} \subseteq \|\!\rightharpoonup\!\mathrm{C}'$ *iff* $\mathrm{C}/\mathrm{poly} \subseteq \mathrm{C}'/\mathrm{poly}$.

Proof. Let us assume $\mathrm{C}/\mathrm{poly} \subseteq \mathrm{C}'/\mathrm{poly}$, and let A be a C-complete problem. This implies that $*A$ is $\|\!\rightharpoonup\!\mathrm{C}$-complete and that $A \in \mathrm{C}'/\mathrm{poly}$. Thus, it holds that $*A \in \|\!\rightharpoonup\!\mathrm{C}'$. As a result, there is a $\|\!\rightharpoonup\!\mathrm{C}$-complete problem that belongs to $\|\!\rightharpoonup\!\mathrm{C}'$. Thus, $\|\!\rightharpoonup\!\mathrm{C} \subseteq \|\!\rightharpoonup\!\mathrm{C}'$.

Let us now assume $\|\!\rightharpoonup\!\mathrm{C} \subseteq \|\!\rightharpoonup\!\mathrm{C}'$. Let A be a generic C/poly problem. We prove that A is also $\mathrm{C}'/\mathrm{poly}$. By definition there exist a polysize function f and a C language S such that

$$y \in A \quad \text{iff} \quad \langle f(||y||), y\rangle \in S$$

As a result, the language of pairs $*A$ is in $\|\!\rightharpoonup\!\mathrm{C}$. Therefore, it is also in $\|\!\rightharpoonup\!\mathrm{C}'$ and there exist a polysize function f and a C' language S' such that

$$\langle x, y\rangle \in *A \quad \text{iff} \quad \langle f(x, ||y||), y\rangle \in S'$$

As a consequence, we have

$$y \in A \quad \text{iff} \quad \langle \epsilon, y\rangle \in *A \quad \text{iff} \quad \langle f(\epsilon, ||y||), y\rangle \in S'$$

Since $f(\epsilon, ||y||)$ is indeed a polysize function of $||y||$ alone, and S' is in C', we conclude that A is in $\mathrm{C}'/\mathrm{poly}$. □

This theorem proves also that the non-uniform compilability hierarchy is proper: indeed, Yap [Yap83] proved (extending some results by Karp and Lipton) that if the non-uniform complexity hierarchy collapses, so does (at one level higher) the polynomial hierarchy. For example, if $\mathrm{P}/\mathrm{poly} = \mathrm{NP}/\mathrm{poly}$ then $\Sigma_2^p{=}\Pi_2^p{=}\mathrm{PH}$. As a result, $\|\!\rightharpoonup\!\mathrm{P} = \|\!\rightharpoonup\!\mathrm{NP}$ implies $\Sigma_2^p{=}\Pi_2^p{=}\mathrm{PH}$ as well.

8.2 Fixed Parameter Tractability

The classes of fixed parameter tractability [DF95b, DF95a] were initially defined as extension of the non-uniform complexity classes. There is a strong formal similarity between the classes of fixed parameter tractability and the compilability class, and is the fact that the input is composed by two parts,

and the aim is to obtain an algorithm that is polynomial w.r.t. the second part. However, the rationale behind these two theories is different.

The difference can be expressed as follows: in [DF95b] "the main idea is to study languages that are tractable by the slice"; in [CDLS96b] the authors wants "to investigate the idea of processing off-line part of the input data".

These are different goals:

- In the fixed parameter classes the focus is on the cost of an algorithm, when a parameter of the input is "fixed". For example the problem of Vertex Cover is fixed parameter tractable because there exists an algorithm that can solve it whose cost is bounded by a polynomial, when the number k of nodes in the desired cover is assumed to be a constant.

- The idea of off-line processing is that some problems can be efficiently solved if a precompilation of a part of the input is allowed, and the only bound on this precompilation is that the compiled data structure does not have exponential size.

The class of fixed parameter tractable problems is defined as follows [DF95b].

Definition 54 *Let A be a parameterized problem. A is* uniformly fixed parameter tractable *($A \in$ uFPT) if there is an algorithm Φ, a constant c, and an arbitrary function $f : \mathbb{N} \to \mathbb{N}$ such that*

(a) the running time of $\Phi(\langle x, k\rangle)$ is at most $f(k)|x|^c$, and

(b) $\langle x, k\rangle \in A$ iff $\Phi(\langle x, k\rangle) = 1$.

Intuitively, a problem is fixed parameter tractable if its complexity is polynomial in the size of the x, and possibly exponential in the parameter. This is a way to determine which is the cause of the exponentiality, that is, which part of the instance causes the problem to be exponential. The similarity between uFPT and $\rightsquigarrow$P is that both classes consider problems in which the input is a pair of strings. Moreover, both definition aim at capturing the idea that a problem may be tractable w.r.t. only one part of the input.

However, there is a huge conceptual difference. The problems in uFPT can be solved by an algorithm that takes *both the first part of the input and the parameter*, while problems in $\rightsquigarrow$P are solved by a polynomial algorithm after the preprocessing of the fixed part. In both cases an exponential time is allowed. In the uFPT problems the time of computing is exponential w.r.t. the parameter, while in $\rightsquigarrow$P problems an exponential-time processing is allowed only on the fixed part, and this preprocessing *has no access to the varying part of the input.*

Graphically, the uFPT computation can be represented as in Figure 3.1: there is a single box that takes both the parts of the input. What is important is that the time spent by the algorithm is a function on the form $f(k)|x|^c$.

The $\rightsquigarrow$P computation is instead represented as in Figure 3.4: the fixed part of the input goes to the preprocessing block, which does not have knowledge of the varying part. After the preprocessing, a polynomial effort is enough to solve the problem.

Another (formal) difference is that the class $\rightsquigarrow$P contains undecidable problems. As a result, $\rightsquigarrow$P is not contained in uFPT. To make the comparison more meaningful, we consider the variant of $\rightsquigarrow$P that contains only decidable problems. Let FD be the class of decidable functions. Recall that $\rightsquigarrow$P is defined as $\langle \text{FPSIZE}, \text{P} \rangle$. We prove that $\langle \text{FPSIZE} \cap FD, \text{P} \rangle$ is strictly contained in the class uFPT.

Theorem 82 *It holds* $\langle \text{FPSIZE} \cap FD, \text{P} \rangle \subseteq \text{uFPT}$.

Proof. Let A be a $\langle \text{FPSIZE} \cap FD, \text{P} \rangle$ problem. As a result, there exists a polysize decidable function f and a polynomial language S such that $\langle x, y \rangle \in A$ if and only if $\langle f(x), y \rangle \in S$. As a result, given x and y, it is possible to solve the problem by determining $f(x)$ and then deciding whether $\langle f(x), y \rangle \in S$. If h is the time needed for the computation of f, and g is the time for the decision of membership of S, this means that the total time is $h + g$, where g is a polynomial. As a result, the problem is in uFPT. □

The converse does not hold. It can be proved that there are problem that are fixed parameter tractable, but not in $\rightsquigarrow$P. The problem is that the computation used in the uFPT definition needs both the parts of the input. On the converse, the preprocessing phase of $\rightsquigarrow$P (that is the only non-polynomial step allowed) uses only the fixed part of the input.

Theorem 83 *It holds* $\langle \text{FPSIZE} \cap FD, \text{P} \rangle \neq \text{uFPT}$.

Proof. Let us consider the problem fi (formula inference)

$$\text{fi} = \{\langle x, y \rangle \mid x \text{ and } y \text{ are propositional formulas and } x \models y\}$$

This problem is compcoNP complete, thus it is not in $\rightsquigarrow$P (unless the polynomial hierarchy collapses), and thus it is not in $\langle \text{FPSIZE} \cap FD, \text{P} \rangle$, either (the latter is a subset of $\rightsquigarrow$P). However, this problem is in uFPT. In order to formalize this problem in the fixed parameter framework, one of the two inputs of the problem must be a number. We assume that the representation of x is a number (the string that represents x can always be considered as a number).

Given x and y, generate all the models of x, and for each such model, verify whether it implies y. The running time of this procedure is $O(2^{||x||} \cdot ||y||)$, and is thus uniformly fixed parameter tractable (indeed, $||x||$ is the size of the number x, as a result it is $\log x$). Such an algorithm is exponential only in the size of the fixed part, but it requires the knowledge of the varying part to work. This procedure does not prove that fi is $\rightsquigarrow$P (or in $\langle \text{FPSIZE} \cap FD, \text{P} \rangle$). □

An easy corollary of the above two theorems is the relationship between $\langle \text{FPSIZE} \cap FD, \text{P} \rangle$ and uFPT.

Corollary 4 *It holds* $\langle \text{FPSIZE} \cap FD, \text{P} \rangle \subset \text{uFPT}$.

Since $\langle \text{FPSIZE} \cap FD, \text{P} \rangle$ is contained in $\rightsquigarrow$P, we also obtain that $\rightsquigarrow$P is different from uFPT. We now shortly recall why we choose to have undecidable problems in the class $\rightsquigarrow$P. By allowing the solution of undecidable problems in the preprocessing phase, we are just strengthening non-compilability results such as $\|\not\rightsquigarrow$coNP-hardness of diagn: any problem that is, e.g. $\|\not\rightsquigarrow$coNP hard is not compilable *even if we can solve arbitrarily hard problem in the preprocessing phase.* If we were to limit such power, e.g., by using $\langle \text{FPSIZE} \cap FD, \text{P} \rangle$ to be the class of problems that can be solved in polynomial time after preprocessing, the corresponding reduction (to be compatible with the classes of this new hierarchy) must have f_1 and f_2 decidable too. As a result, each reduction of this kind is also a nucomp reduction, but not the other way around. This implies that there are problems that can be proven to be not compilable using nucomp reductions and hardness, but can not if decidability is imposed. Thus, limiting the preprocessing phase to decidable problems does not give any advantage in proving that a problem is not compilable.

Chapter 9

Conclusions

The main contribution of this dissertation is the precise formalization of the idea of compilation. As shown in the Introduction, this idea of increasing the efficiency by compiling part of the input is not novel, but, till now, a precise formalization was not given. After all, algorithms themselves are a form of compilation: humans develop algorithms, even if this takes a large amount of time, in order to solve specific instances of problems in a more efficient way. Here the focus is on compilation done by computers rather than humans.

This dissertation contains the basic definitions of compilation, that is, the definition of the classes $\langle \mathrm{FC}, \mathrm{C} \rangle$, which give a measure of how much a problem can gain from a compilation. To this end, we define the FC reductions. Using these reductions we show how to prove hardness of problems w.r.t. classes $\langle \mathrm{FC}, \mathrm{C} \rangle$. We then extend this definition by introducing the class FPSIZE and the corresponding FPSIZE reductions. Since, in many cases, these reductions are not adequate, we introduced the non-uniform classes of compilability and their associated reductions.

These non-uniform classes and reductions are used to prove the incompilability of several problems of AI and graph theory. They are also useful in proving results on compact representations, and to compare AI formalisms from the point of view of their ability of expressing knowledge in little space.

We conclude this chapter with a list of possible improvements of the results presented in this dissertation, and with a list of possible future work based on the results on compilation.

There are two points we left open in the last chapters. The first is about the concept of monotonicity of reductions. What we proved was that the existence of a polynomial monotonic reduction from an NP hard problem to a problem of pairs implies the non-compilability (to P) of the latter. This is useful, since monotonicity is often easy to prove. However, this result does not give any

hint on how to prove, for example, that a problem is not compilable to NP (e.g. it is not possible to prove that a problem is $\|\!\rightsquigarrow\Sigma_2^p$ hard in this way.) This is not only a theoretical question. Many heuristics exist for NP problems. As a result, it may be useful to decide whether a problem is compilable to NP or not, but this cannot be proved using the concept of monotonicity. The open problem is to establish if the results on monotonicity can be extended in order to prove hardness of problems w.r.t. classes other than $\|\!\rightsquigarrow$NP.

Another open problem concerns the existence of compact representations w.r.t. logical equivalence. Indeed, it is possible to prove that a function (e.g. a revision operator) has a compact representation w.r.t. logical equivalence just by giving the formula that is logically equivalent to the result of the operator. If, in spite of all our efforts, we fail to find such a formula, we should try to prove that the function does not have a compact representation. The techniques presented in this dissertation only allow to prove that a function does not have a compact representation w.r.t. model equivalence. This means that, is we fail to find a compact representation w.r.t. logical equivalence for a function that have a compact representation w.r.t. model equivalence, it is not possible to say anything about the existence of a compact representation w.r.t. logical equivalence: we cannot prove that it exists, nor that it does not. A note on how this may be done for the standard semantics update (which has not been defined here) in the Horn case can be found in [Lib95].

Let us now consider some possible future directions of research. The first is the idea of using compilation together with approximation. Since many problems are not compilable if we look for an *exact* solution, we may find convenient to search for an approximate solution, after the preprocessing. The use of preprocessing may speed up the search, or even lead to better approximate solutions (that is, solutions closer to the exact solution).

Another question to investigate is whether there exists any relationship between the concept of compilation and that of expressiveness. The latter is well-studied in the database field, and is loosely related to complexity. Indeed, it has been proved that expressiveness implies complexity, but not vice versa. Compilability seems to be something between complexity and expressiveness.

Finally, the classes of compilability seems to help in the investigation of the complexity of problems in non-classical settings. Consider for example a problem of planning. There is an initial state, a goal, and a set of actions we can perform to achieve the goal. Once found the plan, it may be that we need to achieve a goal that is slightly different from the previous one. In this case, the old plan may be useful. This problem is somewhat "non-classical", since part of the input is not "part of the problem" but rather an hint in finding the solution. For such problems, the idea of compilation may be of help.

Bibliography

[AGM85] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[ATBJ87] D. Allemang, M. C. Tanner, T. Bylander, and J. R. Josephson. Computational complexity of hypothesis assembly. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI'87)*, pages 1112–1117, 1987.

[Bak91] A. Baker. Nonmonotonic reasoning in the framework of the situation calculus. *Artificial Intelligence*, 49:5–23, 1991.

[BED91] R. Ben-Eliyahu and R. Dechter. Default logic, propositional logic and constraints. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, pages 379–385, 1991.

[BED94] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[Bou93] C. Boutilier. Revision sequences and nested conditionals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 519–525, 1993.

[BS90] R. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 14, pages 757–804. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[Cad92] M. Cadoli. The complexity of model checking for circumscriptive formulae. *Information Processing Letters*, 44:113–118, 1992.

[CDLS95] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. The size of a revised knowledge base. In *Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of*

Database Systems (PODS'95), pages 151–162, 1995. Extended version available as Technical Report DIS 34-96, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", November 1996.

[CDLS96a] M. Cadoli, F. Donini, P. Liberatore, and M. Schaerf. Comparing space efficiency of propositional knowledge representation formalisms. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 364–373, 1996.

[CDLS96b] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Feasibility and unfeasibility of off-line processing. In *Proceedings of the Fourth Israeli Symposium on Theory of Computing and Systems (ISTCS'96)*, pages 100–109. IEEE Computer Society Press, 1996. URL = `ftp://ftp.dis.uniroma1.it/PUB/AI/papers/cado-etal-96.ps.gz`.

[CDS96] M. Cadoli, F. M. Donini, and M. Schaerf. Is intractability of non-monotonic reasoning a real drawback? *Artificial Intelligence*, 88(1–2):215–251, 1996.

[CDSS97] M. Cadoli, F. M. Donini, M. Schaerf, and R. Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997.

[Dal88] M. Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'88)*, pages 475–479, 1988.

[DF95a] R. G. Downey and M. F. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[DF95b] R. G. Downey and M. F. Fellows. Parameterized complexity, February 1995. Manuscript, 411 pages. To appear by Springer in 1998.

[DG84] W. P. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[EG92] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[EG93] T. Eiter and G. Gottlob. Propositional circumscription and extended closed world reasoning are Π_2^p-complete. *Theoretical Computer Science*, 114:231–245, 1993.

[Eth87] D. V. Etherington. *Reasoning with incomplete information.* Morgan Kaufmann, Los Altos, Los Altos, CA, 1987.

[For89] K. D. Forbus. Introducing actions into qualitative simulation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1273–1278, 1989.

[FUV83] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *Proceedings of the Second ACM SIGACT SIGMOD Symposium on Principles of Database Systems (PODS'83)*, pages 352–365, 1983.

[Gär88] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* Bradford Books, MIT Press, Cambridge, MA, 1988.

[Gin86] M. L. Ginsberg. Conterfactuals. *Artificial Intelligence*, 30:35–79, 1986.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, Ca, 1979.

[GKPS95] G. Gogic, H. A. Kautz, C. Papadimitriou, and B. Selman. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 862–869, 1995.

[GL93] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[GPP89] M. Gelfond, H. Przymusinska, and T. Przymusinsky. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:49–73, 1989.

[HM87] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[HV91] J. Y. Halpern and M. Y. Vardi. Model checking vs. theorem proving: A manifesto. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR'91)*, 1991. Also in V. Lifshitz, ed. *Artificial Intelli-*

gence and Mathematical Theory of Computation. Papers in Honor of John McCarthy, Academic Press, San Diego, 1991.

[Kar72] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, 1972.

[KL80] R. M. Karp and R. J. Lipton. Some connections between non-uniform and uniform complexity classes. In *Proceedings of the Twelfth ACM Symposium on Theory of Computing (STOC'80)*, pages 302–309, 1980.

[KM89] H. Katsuno and A. O. Mendelzon. A unified view of propositional knowledge base updates. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1413–1419, 1989.

[KM91] H. Katsuno and A. O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52:263–294, 1991.

[KS92] H. A. Kautz and B. Selman. Forming concepts for fast inference. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 786–793, 1992.

[Leh95] D. Lehmann. Belief revision, revised. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1534–1540, 1995.

[Lib95] P. Liberatore. Compact representation of revision of Horn clauses. In Xin Yao, editor, *Proceedings of the Eighth Australian Joint Artificial Intelligence Conference (AI'95)*, pages 347–354. World Scientific, 1995.

[Lib97a] P. Liberatore. The complexity of belief update. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 68–73, 1997.

[Lib97b] P. Liberatore. The complexity of iterated belief revision. In *Proceedings of the Sixth International Conference on Database Theory (ICDT'97)*, pages 276–290, 1997.

[Lib97c] P. Liberatore. The complexity of the language $\mathcal{A}$. *Electronic Transactions on Artificial Intelligence*, 1(1–3):13–37, 1997. URL = `http://www.ep.liu.se/ea/cis/1997/006/`.

[LS95a] P. Liberatore and M. Schaerf. Arbitration: A commutative operator for belief revision. In *Proceedings of the Second World Conference on the Fundamentals of Artificial Intelligence (WOCFAI'95)*, pages 217–228, 1995.

[LS95b] F. Lin and Y. Shoham. Provably correct theories of action. *Journal of the ACM*, 42(2):293–320, 1995.

[LS96] P. Liberatore and M. Schaerf. The complexity of model checking for belief revision and update. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 556–561, 1996.

[LS97] P. Liberatore and M. Schaerf. Reducing belief revision to circumscription (and viceversa). *Artificial Intelligence*, 93(1–2):261–296, 1997.

[McC80] J. McCarthy. Circumscription - A form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[MH69] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[NH79] S. Ntafos and S. Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transaction on Software Engigneering*, SE-5:520–529, 1979.

[PR86] Y. Peng and J. Reggia. Plausibility of diagnostic hypothesis. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI'86)*, pages 140–145, 1986.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Rei91] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, New York, 1991.

[Sat88] K. Satoh. Nonmonotonic reasoning by minimal belief revision. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88)*, pages 455–462, 1988.

[Tos78] P. Tosatti. *L'amico delle conversazioni.* Tipografia pontificia ed arcivescovile dell'immacolata concezione, 1878.

[Wil94] M. Williams. Transmutations of knowledge systems. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 619–629, 1994.

[Win89] M. Winslett. Sometimes updates are circumscription. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 859–863, 1989.

[Win90] M. Winslett. *Updating Logical Databases.* Cambridge University Press, 1990.

[Yap83] C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.

Index

List of Symbols

$\min(S, \preceq)$	minimal elements of the set S w.r.t. the ordering $\preceq$
$\max(S, \preceq)$	maximal elements
$Diff$	symmetric difference
$\mid \mid$	number of elements of a set
$\|\| \ \|\|$	size of a string[1]
$\models$	logical implication, models
$\models_F$	models of a knowledge base in the formalism F
$\vdash_F$	theorems of a knowledge base in the formalism F
Mod	set of models of a formula
$Form$	inverse of Mod
γ^{neg}	the clause obtaining by replacing each positive literal in the clause γ with its opposite
Var	variables appearing in a formula
$T[X/Y]$	formula obtained by replacing each $x_i \in X$ with $y_i \in Y$
$\hat{X}$	$\{\neg x_i \mid x_i \in X\}$
Π_X	set of all the clauses of three literals over the alphabet X
C	generic class of decision problems
FC	generic class of finding problems
P	class of the polynomial problems
NP, coNP, Σ_i^p, Π_i^p, Δ_i^p	classes of the polynomial hierarchy
PH	polynomial hierarchy
uFPT	class of the fixed parameter tractable problems
C/poly	class of C non-uniformly computable problems
EXPTIME	class of the exponential problems
FPSIZE	class of the finding problems whose solution has polynomial size

[1] Since any element can be represented as a string, this notation refers to any element.

$\langle \mathrm{FC}, \mathrm{C} \rangle$	classes of problems whose complexity is C after an FC preprocessing
$\rightsquigarrow$C	class of problems compilable to C
$\|\!\!\sim\!\!\rightarrow$C	class of problems non-uniformly compilable to C
$\leq^{\mathrm{P}}$	polynomial reducibility
$\leq^{\mathrm{FC}}$	FC reducibility
$\leq_{comp}$	reducibility via a comp reduction
$\leq_{nucomp}$	reducibility via a nucomp reduction
$*A$	the problem $\{\langle x, y \rangle \mid y \in A\}$
$*$	generic revision
$*_G, *_D, \ldots$	specific revision operators
$k_{T,P}$	minimal distance between models of T and P
$K_{T,P}$	set of the minimal differences between models of T and models of P
$\leq_T$	ordering associated to T, according to an AGM revision
κ_T	ranking associated to T
$\kappa_{[T;P_1,\ldots,P_m]}$	ranking associated to the result of revising T w.r.t. $P_1, \ldots, P_m$
$T * [P_1, \ldots, P_m]$	result of iterated revision
CIRC	circumscription of a formula
$\frac{P:Q}{Q}$	default
MC	model checking
QA	query answering
$\mapsto$	polysize reduction between formalisms
thm-C	formalisms whose query answering is $\|\!\!\sim\!\!\rightarrow$C
model-C	formalisms whose model checking is $\|\!\!\sim\!\!\rightarrow$C
initially , causes , after	syntax of $\mathcal{A}$
Φ	transition function in $\mathcal{A}$
Ψ_D	transition function associated to the domain description D
σ	state
(σ_0, Φ)	interpreted structure (semantics of $\mathcal{A}$)
(N, E)	graph: N is the set of nodes, E is the set of edges
K_m	complete graph with m nodes

Dottorato di Ricerca in Informatica
Collana della tesi
Collection of Theses

V-93-1 Marco Cadoli. *Two Methods for Tractable Reasoning in Artificial Intelligence: Language Restriction and Theory Approximation.* June 1993.

V-93-2 Fabrizio d'Amore. *Algorithms and Data Structures for Partitioning and Management of Sets of Hyperrectangles.* June 1993.

V-93-3 Miriam Di Ianni. *On the complexity of flow control problems in Store-and-Forward networks.* June 1993.

V-93-4 Carla Limongelli. *The Integration of Symbolic and Numeric Computation by p-adic Construction Methods.* June 1993.

V-93-5 Annalisa Massini. *High efficiency self-routing interconnection networks.* June 1993.

V-93-6 Paola Vocca. *Space-time trade-offs in directed graphs reachability problem.* June 1993.

VI-94-1 Roberto Baldoni. *Mutual Exclusion in Distributed Systems.* June 1994.

VI-94-2 Andrea Clementi. *On the Complexity of Cellular Automata.* June 1994.

VI-94-3 Paolo Giulio Franciosa. *Adaptive Spatial Data Handling.* June 1994.

VI-94-4 Andrea Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues.* June 1994.

VI-94-5 Andrea Sterbini. *2-Thresholdness and its Implications: from the Synchronization with PVchunk to the Ibaraki-Peled Conjecture.* June 1994.

VII-95-1 Piera Barcaccia. *On the Complexity of Some Time Slot Assignment Problems in Switching Systems.* June 1995.

VII-95-2 Michele Boreale. *Process Algebraic Theories for Mobile Systems.* June 1995.

VII-95-3 Antonella Cresti. *Unconditionally Secure Key Distribution Protocols.* June 1995.

VII-95-4 Vincenzo Ferrucci. *Dimension-Independent Solid Modeling.* June 1995.

VII-95-5 Esteban Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging.* June 1995.

VII-95-6 Michele Flammini. *Compact Routing Models: Some Complexity Results and Extensions.* June 1995.

VII-95-7 Giuseppe Liotta. *Computing Proximity Drawings of Graphs.* June 1995.

VIII-96-1 Luca Cabibbo. *Querying and Updating Complex-Object Databases.* May 1996.

VIII-96-2 Diego Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.* May 1996.

VIII-96-3 Marco Cesati. *Structural Aspects of Parameterized Complexity.* May 1996.

VIII-96-4 Flavio Corradini. *Space, Time and Nondeterminism in Process Algebras.* May 1996.

VIII-96-5 Stefano Leonardi. *On-line Resource Management with Application to Routing and Scheduling.* May 1996.

VIII-96-6 Rosario Pugliese. *Semantic Theories for Asynchronous Languages.* May 1996.