

# Knowledge Compilation: A Sightseeing Tour (tutorial notes – ECAI’08)

Pierre Marquis<sup>1</sup>

## Abstract.

Pioneered two decades ago, knowledge compilation (KC) has been for a few years acknowledged as an important research topic in AI. KC is concerned with the pre-processing of pieces of available information for improving the computational efficiency of some tasks based on them. In AI such tasks typically amount to inference or decision making. KC gathers a number of research lines focusing on different problems, ranging from theoretical ones (where the key issue is the compilability one, i.e., determining whether computational improvements can be guaranteed via pre-processing) to more practical ones (mainly the design of compilation algorithms for some specific tasks, like clausal entailment). The (multi-criteria) choice of a target language for KC is another major issue. In these tutorial notes, I review a number of the most common results of the literature on KC in the propositional case. The tour includes some attractions, especially algorithms for improving clausal entailment and other forms of inference; a visit of the compilability district and a promenade following the KC map are also included.

## 1 INTRODUCTION

These tutorial notes are about *knowledge compilation (KC)*, a family of approaches proposed so far for addressing the intractability of a number of AI problems [12]. The key idea underlying KC is to pre-process parts of the available information (i.e., turning them into a compiled form using a compilation function  $comp$ ) for improving the computational efficiency of some tasks of interest. Thus, KC amounts to a *translation* issue. Two phases are usually considered within KC:

- *Off-line phase*: it aims at turning some pieces of information  $\Sigma$  into a *compiled form*  $comp(\Sigma)$ .
- *On-line phase*: its purpose is to exploit the compiled form  $comp(\Sigma)$  (and the remaining pieces of information  $\alpha$ ) to achieve the task(s) under consideration.

KC has emerged some years ago as a new direction of research in AI (thus “KC” appears for a couple of years as a key word in the calls for papers of several AI conferences). The introduction of the name “KC” itself dates back to the late 80’s/beginning of the 90’s [71, 60, 75]. From there, many developments have been done, both from the theoretical side (with the introduction of new concepts, algorithms, etc.) and from the practical side (benchmarks, pieces of

software, applications, etc.). One of the objectives of these notes is to review the main ones, focusing on the propositional case.

If KC has been identified as a research topic in the quite recent past, the idea of pre-processing pieces of information for improving computations (which typically means “saving computation time”) is an old one. It has many applications in Computer Science (even before the modern computer era). One of the most salient example for illustrating it is the notion of *table of logarithms*. Such tables have been introduced during the 17<sup>th</sup> century and used for centuries as “compiled forms” for improving many computations (it was still used three decades ago in french secondary schools). The principle is first to compute the values of  $\log_{10}(x)$  (using formal series developments) for many real numbers  $x \in \Sigma \subseteq [1, 10)$  and to store pairs  $comp(\Sigma) = \langle x, \log_{10}(x) \rangle$  in the table (the off-line phase, done once for all), then to take advantage of the properties of  $\log_{10}$  to simplify forthcoming computations (the on-line phase). For instance, assume that one wants to compute the “value” of  $\alpha = \sqrt[5]{1234}$ . Since  $\sqrt[5]{1234} = (1.234 \times 10^3)^{\frac{1}{5}}$ , we get that  $\log_{10}(\sqrt[5]{1234}) = \log_{10}((1.234 \times 10^3)^{\frac{1}{5}}) = \frac{\log_{10}(1.234)+3}{5}$ ; it is thus enough to look up the “value” of  $\log_{10}(1.234)$  in the table

$$\langle 1.234, 0.09131516 \rangle \in comp(\Sigma)$$

compute  $\frac{0.09131516+3}{5} = 0.618263032$ , and finally look up the antecedent by  $\log_{10}$  of the resulting value 0.618263032 in the table ( $\langle 4.152054371, 0.618263032 \rangle \in comp(\Sigma)$ ) (or use an antilog table); we get that  $\sqrt[5]{1234} = 4.152054371$  without a heavy computational effort (mainly an addition and a division). Of course, the computational effort spent in the off-line phase (the table generation) makes sense because it is amortized over the many facilitated computations it enables.

The idea of pre-processing pieces of information for improving computations has been also widely used for years in many areas of Computer Science. Of course, one immediately thinks about the compilation of computer languages which gives rise to much research for the 50’s (a “compiled form”  $comp(\Sigma)$  is here a piece of object code, which runs typically more quickly than the corresponding source code  $\Sigma$  on the given data  $\alpha$  since one level of interpretation has been removed). But database indexes, lookup tables heavily used in computer graphics and other caching techniques considered for instance for static optimization of code also amount to the same key idea as the one underlying KC.

From the terminology point of view, what makes KC different from previous compilation techniques considered outside AI is that it is “knowledge-based”. However, this is not so much a major distinction since within KC, “knowledge” must not be taken in the technical sense of “true beliefs” but in a rather broad sense: roughly, “knowledge” amounts to pieces of information (“declarative knowledge”)

<sup>1</sup> Université Lille-Nord de France, Artois, CRIL UMR CNRS 8188, France, email: marquis@cril.univ-artois.fr

and ways to exploit them (this is also the meaning of “knowledge” in “knowledge representation”). Pieces of information are typically encoded as formulas  $\Sigma, \alpha, \dots$  in a *logic-based language*  $L$ , where for the sake of generality, a logic is viewed as a pair  $\langle L, \vdash \rangle$  where  $\vdash$  is an *inference relation* (a binary relation over  $L$ ).

What are the *tasks* to be computationally improved via KC? Of course, they are *domain-dependent* in the general case. Nevertheless, for many applications, such tasks require *combinations of basic, domain-independent queries and transformations*. Queries are about extracting pieces of information from compiled forms while transformations correspond to update operations on compiled forms. In a logic-based setting, basic queries are the inference one and the consistency one:

- **Inference:** Given  $\Sigma, \alpha \in L$ , does  $\Sigma \vdash \alpha$  hold?
- **Consistency:** Given  $\Sigma \in L$ , does there exist  $\alpha \in L$  such that  $\Sigma \not\vdash \alpha$  holds?

Basic transformations include conditioning, closures under connectives  $\otimes$  and forgetting:

- **Conditioning:** Make some elementary propositions true (or false) in  $\Sigma \in L$ .
- **Closures under connectives:** Compute a representation in  $L$  of  $\alpha \otimes \beta$  from  $\alpha \in L$  and  $\beta \in L$ .
- **Forgetting:** When defined, compute a representation of the most general consequence w.r.t.  $\vdash$  of  $\Sigma \in L$  not containing some given elementary propositions.

For instance, the consistency-based diagnoses of a system (à la de Kleer and Reiter) can be computed as the models of the propositional formula obtained by forgetting every elementary proposition except those denoting the components states, in the formula obtained by conditioning the system description (which expresses the behaviour of each component of the system and the way they are connected) by the available observations (typically the inputs and outputs of the system). Thus, if models generation, forgetting and conditioning are computationally easy queries/transformations when the system description has been first turned into a compiled form, the generation of diagnoses is easy as well, and the computational effort spent in compiling the system description can be balanced over many sets of observations. Interestingly, there exist *target languages for KC* ensuring that such queries and transformations can be achieved in polynomial time, especially the languages  $\text{PI}$  and  $\text{DNNF}$  (which will be detailed later on).

Typically, among the basic queries and transformations under consideration, they are some *intractable ones* (formally, the corresponding decision or function problem is NP-hard in the sense of Cook reduction). Indeed, for such queries and transformations, no polynomial-time algorithms (for our deterministic computers) are known and it is conjectured that none of them exists (this is the famous  $\text{P} \neq \text{NP}$  conjecture, perhaps the most important conjecture in Computer Science). For instance, in the consistency-based diagnosis setting as described above, the problem (query) of determining whether a diagnosis exists for a given set of observations is already NP-complete.

Now, a fundamental question is the assessment one: “when does KC achieve its goal?”. Roughly speaking, KC makes sense when some of the queries and transformations of interest become “less intractable” once some pieces of the available information have been compiled first, provided that the computational effort spent during the off-line phase is “reasonable”. Especially, if the pieces of information to be compiled (the so-called *fixed part*) vary frequently, it is

unlikely that the computational resources required to derive the compiled form could be balanced, by considering sufficiently instances of the computational problem under consideration, with the same fixed part and different *varying parts*. Fortunately, the assumption that some pieces of information are not very often subject to change is valid in many scenarios (e.g. in a diagnostic setting, it is reasonable to assume that the system description does not often change, or at least varies less frequently than the sets of observations to be considered). Deriving more accurate answers to the fundamental evaluation question requires to make precise what “less intractable” and “reasonable” mean.

Basically, deciding whether KC achieves its goal of computationally improving a given query or transformation depends on whether the task is considered at the *problem level* or at the *instance level*. At the problem level, “less intractable” just means that some sources of intractability have been removed via the compilation phase (for decision problems, assuming that the fixed part of any instance belongs to the target language for KC under consideration leads to a restriction of the problem, which is at a lower level of the polynomial hierarchy than the unrestricted problem). In many cases, “less intractable” means tractable, i.e., solvable in (deterministic) polynomial time. “Reasonable” means that the size of the compiled form of the fixed part is guaranteed to be polynomial in the size of the fixed part. This is a mandatory requirement for avoiding an unbiased evaluation: consider an NP-hard problem for which the most efficient resolution algorithm one knows runs in  $\mathcal{O}(2^n)$  time in the worst case (where  $n$  is the input size); assume that there exists a second resolution algorithm for the same problem running in  $\mathcal{O}(n)$  time in the worst case, provided that the input has been compiled first: would it be reasonable to say that the source of intractability has been removed via compilation and that KC is successful here? Well, certainly not if the size of the compiled form is exponential in the size of the instance considered at start! In such a case, the second algorithm will run on an input exponentially larger than the first one, so that its running time may easily exceed the running time of the first algorithm on the original instance. Furthermore, the overall algorithm obtained here following the “compiled approach” (i.e., the sequential combination of the compilation algorithm used to generate the compiled form followed by the second algorithm described above) requires exponential space to work (since the compiled form has to be stored), while this is not necessarily the case for the first algorithm, corresponding to a “direct, uncompiled” approach to the resolution of the same problem.

Now, at the instance level, one does not focus on worst-case computations on arbitrarily large instances of the problem under consideration but on a *given set of instances*, where some instances share the same fixed part. One typically computes some statistics about the time needed to solve all the instances sharing the same fixed part (for each such fixed part), assuming that the fixed part has been compiled or without such an assumption (i.e., following a “direct, uncompiled” approach), and we compare them. For instance, if one considers the problem of generating a consistency-based diagnosis for a system, one starts with a set of instances, where for the same system description (the fixed part), one encounters several sets of observations (the varying part). Then we compile the system descriptions and for each system description, we measure the runtimes needed to compute a consistency-based diagnosis for each instance sharing the system description under consideration. We do the same but considering the system descriptions provided at start (i.e., the “uncompiled ones”). For each system description such that the overall runtime required for solving all the associated instances assuming that the system descrip-

tion has been compiled is lower than the overall runtime required for solving all the associated instances without this assumption, we have some evidence that KC proved useful (at least in the limit). In order to get a more complete assessment, one can also compute for each fixed part a number of instances from which the compilation time is balanced.

As usual when dealing with complexity considerations,<sup>2</sup> the two evaluation levels have their pros and their cons and are actually complementary ones. On the one hand, the evaluation at the problem level can prove too coarse-grained to get an accurate view of the improvements offered by KC for a given application. Indeed, an application corresponds to a specific instance of a problem (or to a finite set of such instances), but not to the problem itself (where the focus is laid on the worst case and arbitrarily large instances are considered). Thus, it can be the case that KC proves useful in practice for solving some instances of interest of a problem, even if it does not make the problem itself “less intractable” or if it does not ensure that the size of the compiled form of a fixed part is polynomial in the size of the fixed part. On the other hand, performing an evaluation of a KC technique at the instance level is more informationally demanding: one needs a significant set of instances, one also needs a “baseline algorithm” corresponding to a “direct, uncompiled” approach to the problem. It is usually difficult to reach a consensus on what “significant” and “baseline” mean here.

Much work achieved so far within KC is concerned with (*classical*) *propositional logic*. This can be explained by the fact that while being one of the simplest logic one may consider (and as such, embedded in many more sophisticated logics), it proves sufficiently expressive for a number of AI problems like the diagnostic one, as considered above; furthermore, while decidable, propositional reasoning is intractable; thus, the inference problem (i.e., the entailment one) is coNP-complete, the consistency one is NP-complete, the forgetting problem is NP-hard. Accordingly, propositional reasoning is a good candidate for KC: some computational improvements would be welcome!

In these notes, I review a number of previous works on KC described so far in the literature. Those works can be classified according to the main issue they consider:

- How to *compile* propositional formulas in order to improve clausal entailment?
- How to *evaluate* from a theoretical point of view whether KC can prove useful?
- How to *choose* a target language for the KC purpose?
- What about KC for *other forms of propositional inference*?

Each of the following sections is centered on one of those issues, except the next section where some formal preliminaries about propositional languages are provided, and the (short) final concluding section. For space reasons, I assume that the reader has some background about computational complexity, especially the classes P, NP, coNP,  $\Theta_2^P$ ,  $\Delta_2^P$ ,  $\Pi_2^P$  and more generally all the classes of the polynomial hierarchy PH (see e.g. [69] for a presentation of those complexity classes). Please note that the main objective of this document is to serve as a support for the corresponding tutorial; if a number of previous approaches are reviewed in it (and the corresponding

<sup>2</sup> That a problem is NP-hard does not say anything about the greatest integer  $n$  such that all the instances of size  $\leq n$  of the problem can be solved within a “reasonable” amount of time, say less than 600s (observe also that making this statement more robust would require a definition of the underlying computational model – in practice, of the computer used for the experiments).

references are given as pointers for further readings), it is far from being exhaustive.

## 2 PROPOSITIONAL LANGUAGES

The propositional languages which have been considered as target languages for KC are typically subsets (also referred to as “fragments”) of the following NNF language:

**Definition 1.** Let  $PS$  be a denumerable set of propositional variables (*atoms*). A formula in NNF is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with *true*, *false*,  $x$  or  $\neg x$ ,  $x \in PS$ ; and each internal node is labeled with  $\wedge$  or  $\vee$  and can have arbitrarily many children. The size of a sentence  $\Sigma$  in NNF, denoted  $|\Sigma|$ , is the number of its DAG arcs plus the number of its DAG nodes. Its height is the maximum number of edges from the root to some leaf in the DAG.

From here on, if  $C$  is a node in an NNF formula, then  $Var(C)$  denotes the set of all variables that label the descendants of node  $C$ . Moreover, if  $\Sigma$  is an NNF sentence rooted at  $C$ , then  $Var(\Sigma)$  is defined as  $Var(C)$ . For any subset  $V$  of  $PS$ ,  $L_V$  denotes the subset of *literals* generated from the atoms of  $V$ , i.e.,  $L_V = \{x, \neg x \mid x \in V\}$ .  $x \in PS$  is said to be a *positive literal* and  $\neg x$  a *negative literal*.  $\sim l$  is the *complementary literal* of  $l \in L_{PS}$ : if  $l = x$  is a positive literal, then  $\sim l = \neg x$ ; if  $l = \neg x$  is a negative literal, then  $\sim l = x$ .

The language of NNF formulas can be viewed as a generalization of a standard propositional language consisting of Negation Normal Form formulas having a tree shape (formally, NNF includes this language as a proper subset).

NNF (and its subsets) are classically interpreted (i.e., the semantics of a formula is a Boolean function):

**Definition 2.** Let  $\omega$  be an interpretation (or “world”) on  $PS$  (i.e., a total function from  $PS$  to  $BOOL = \{0, 1\}$ ). The semantics of a formula  $\Sigma \in \text{NNF}$  in  $\omega$  is the truth value  $\llbracket \Sigma \rrbracket(\omega)$  from  $BOOL$  defined inductively as follows:

- if  $\Sigma = \text{true}$  (resp. *false*), then  $\llbracket \Sigma \rrbracket(\omega) = 1$  (resp. 0).
- if  $\Sigma \in PS$ , then  $\llbracket \Sigma \rrbracket(\omega) = \omega(\Sigma)$ .
- if  $\Sigma = \neg \alpha$ , then  $\llbracket \Sigma \rrbracket(\omega) = 1 - \llbracket \alpha \rrbracket(\omega)$ .
- if  $\Sigma = \wedge(\alpha_1, \dots, \alpha_n)$ , then  $\llbracket \Sigma \rrbracket(\omega) = \min(\{\llbracket \alpha_1 \rrbracket(\omega), \dots, \llbracket \alpha_n \rrbracket(\omega)\})$ .
- if  $\Sigma = \vee(\alpha_1, \dots, \alpha_n)$ , then  $\llbracket \Sigma \rrbracket(\omega) = \max(\{\llbracket \alpha_1 \rrbracket(\omega), \dots, \llbracket \alpha_n \rrbracket(\omega)\})$ .

An interpretation  $\omega$  is said to be a *model* of  $\Sigma$ , noted  $\omega \models \Sigma$ , if and only if  $\llbracket \Sigma \rrbracket(\omega) = 1$ . If  $\Sigma$  has a model, it is *satisfiable* (or consistent); otherwise, it is *unsatisfiable* (or inconsistent). If every interpretation  $\omega$  on  $PS$  is a model of  $\Sigma$ ,  $\Sigma$  is *valid*, noted  $\models \Sigma$ . If every model of  $\Sigma$  is a model of  $\alpha$ , then  $\alpha$  is a *logical consequence* of  $\Sigma$ , noted  $\Sigma \models \alpha$ . Finally, when both  $\Sigma \models \alpha$  and  $\alpha \models \Sigma$  hold,  $\Sigma$  and  $\alpha$  are *equivalent*, noted  $\Sigma \equiv \alpha$ .

One usually distinguishes between two families of subsets of NNF: *flat* and *nested* subsets. Flat NNF formulas are those of height at most 2, and their set is the  $\mathfrak{F}$ -NNF language.

NNF formulas which do not have a tree-like structure can be viewed as compact representations of the corresponding formulas having a tree shape, obtained by sharing subformulas. When considering non-flat formulas, this feature can have a dramatic effect on the representation size (i.e., exponential savings can be achieved).

Well-known flat subsets result from imposing on NNF formulas combinations of the following properties:

- **Simple-disjunction:** The children of each or-node are leaves that share no variables (the node is a *clause*).
- **Simple-conjunction:** The children of each and-node are leaves that share no variables (the node is a *term*).

**Definition 3.**

- The language CNF is the subset of  $\mathcal{F}$ -NNF satisfying simple-disjunction. For every positive integer  $k$ ,  $k$ -CNF is the subset of CNF formulas where each clause contains at most  $k$  literals.
- The language DNF is the subset of  $\mathcal{F}$ -NNF satisfying simple-conjunction.

The following subset of CNF, *prime implicates*, has been quite influential in Computer Science and in AI:

**Definition 4.** The language  $\text{PI}$  is the subset of CNF in which each clause entailed by the formula is entailed by a clause that appears in the formula; and no clause in the formula is entailed by another.

$\text{PI}$  is also known as the language of Blake formulas. A dual of  $\text{PI}$ , *prime implicants*  $\text{IP}$ , can also be defined:

**Definition 5.** The language  $\text{IP}$  is the subset of DNF in which each term entailing the formula entails some term which appears in the formula; and no term in the formula is entailed by another term.

Some other subsets of CNF are also interesting for the KC purpose, for instance the set  $\text{HORN-CNF}$  of Horn CNF formulas, i.e., conjunctions of clauses containing at most one positive literal. It is well-known that the consistency issue for such formulas (the **CO** query) can be addressed in linear time (intuitively, a restricted form of interaction between clauses – unit-resolution – is sufficient to determine whether a  $\text{HORN-CNF}$  formula is contradictory or not).  $\text{HORN-CNF}$  (as well as the other fragments considered here) enables also a polynomial-time conditioning operation (i.e., it satisfies the **CD** transformation):

**Definition 6.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **CO** if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  to 1 if  $\Sigma$  is consistent, and to 0 otherwise.

**Definition 7.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **CD** if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  and every consistent term  $\gamma$  to a formula from  $\mathbf{L}$  that is logically equivalent to  $\Sigma \mid \gamma$ , i.e., the formula obtained by replacing in  $\Sigma$  every occurrence of a variable  $x \in \text{Var}(\gamma)$  by true if  $x$  is a positive literal of  $\gamma$  and by false if  $\neg x$  is a negative literal of  $\gamma$ .

Conditioning has a number of applications, and corresponds to *restriction* in the literature on Boolean functions. The main application of conditioning is due to a theorem, which says that  $\Sigma \mid \gamma$  is consistent if and only if  $\Sigma \mid \gamma$  is consistent. Therefore, if a language satisfies **CO** and **CD**, then it must also satisfy **CE**:

**Definition 8.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **CE** if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  and every clause  $\gamma$  from NNF to 1 if  $\Sigma \models \gamma$  holds, and to 0 otherwise.

Interesting nested subsets of NNF are obtained by imposing some of the following requirements [24]:

- **Decomposability:** An and-node  $C$  is decomposable if and only if the conjuncts of  $C$  do not share variables. That is, if  $C_1, \dots, C_n$  are the children of and-node  $C$ , then  $\text{Var}(C_i) \cap \text{Var}(C_j) = \emptyset$  for  $i \neq j$ . An NNF formula satisfies the decomposability property if and only if every and-node in it is decomposable.

- **Determinism:** An or-node  $C$  is deterministic if and only if each pair of disjuncts of  $C$  is logically contradictory. That is, if  $C_1, \dots, C_n$  are the children of or-node  $C$ , then  $C_i \wedge C_j \models \text{false}$  for  $i \neq j$ . An NNF formula satisfies the determinism property if and only if every or-node in it is deterministic.
- **Decision:** A decision node  $N$  in an NNF formula is one which is labeled with *true*, *false*, or is an or-node having the form  $(x \wedge \alpha) \vee (\neg x \wedge \beta)$ , where  $x$  is a variable,  $\alpha$  and  $\beta$  are decision nodes. In the latter case,  $d\text{Var}(N)$  denotes the variable  $x$ . An NNF formula satisfies the decision property when its root is a decision node.
- **Ordering:** Let  $<$  be a total, strict ordering over the variables from  $PS$ . An NNF formula satisfying the decision property satisfies the ordering property w.r.t.  $<$  if and only if the following condition is satisfied: if  $N$  and  $M$  are or-nodes, and if  $N$  is an ancestor of node  $M$ , then  $d\text{Var}(N) < d\text{Var}(M)$ .
- **Smoothness:** An or-node  $C$  is smooth if and only if each disjunct of  $C$  mentions the same variables. That is, if  $C_1, \dots, C_n$  are the children of or-node  $C$ , then  $\text{Var}(C_i) = \text{Var}(C_j)$  for  $i \neq j$ . An NNF formula satisfies the smoothness if and only if every or-node in it is smooth.

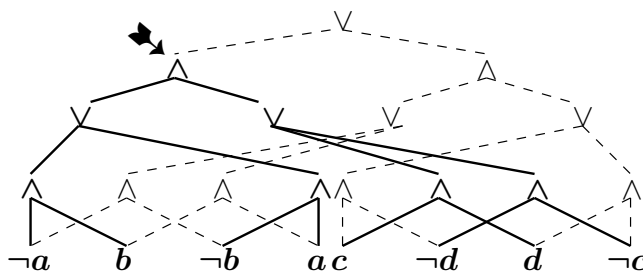


Figure 1. A formula in NNF. The marked node is decomposable.

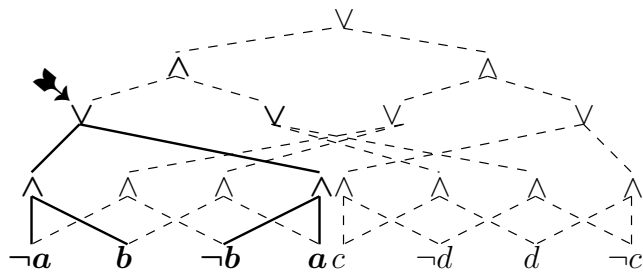


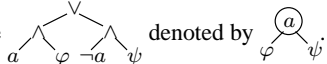
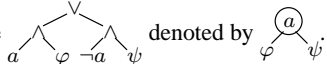
Figure 2. A formula in NNF. The marked node is deterministic and smooth.

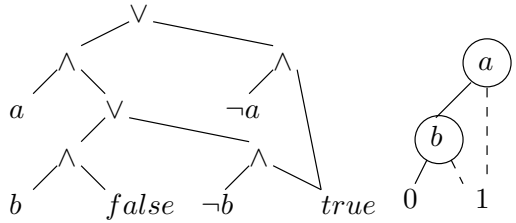
**Example 1.** Consider the and-node marked  $\star$  on Figure 1. This and-node  $C$  has two children  $C_1$  and  $C_2$  such that  $\text{Var}(C_1) = \{a, b\}$  and  $\text{Var}(C_2) = \{c, d\}$ ; node  $C$  is decomposable since the two children do not share variables. Each other and-node in Figure 1 is also decomposable and, hence, the NNF formula in this figure is decomposable. Consider now the or-node marked  $\star$  on Figure 2; it has two children corresponding to subformulas  $\neg a \wedge b$  and  $\neg b \wedge a$ .

Those two subformulas are jointly inconsistent, hence the or-node is deterministic. Furthermore, the two children mention the same variables  $a$  and  $b$ , hence the or-node is smooth. Since the other or-nodes in Figure 2 are also deterministic and smooth, the NNF formula in this figure is deterministic and smooth.

**Definition 9.**

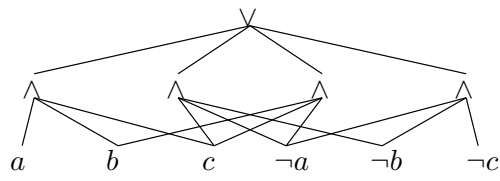
- The language  $\text{DNNF}$  is the subset of NNF of formulas satisfying decomposability.
- The language  $\text{d-DNNF}$  is the subset of NNF of formulas satisfying decomposability and determinism.
- The language  $\text{OBDD}_{<}$  is the subset of NNF of formulas satisfying decomposability, decision and ordering.
- The language  $\text{MODS}$  is the subset of  $\text{DNF} \cap \text{d-DNNF}$  of formulas satisfying smoothness.

Binary decision diagrams [11] are usually depicted using a more compact notation: labels *true* and *false* are denoted by 1 and 0, respectively; and each decision node  denoted by . The  $\text{OBDD}_{<}$  formula on the left part of Figure 3 corresponds to the binary decision diagram on the right part of Figure 3.



**Figure 3.** On the left part, a formula in the  $\text{OBDD}_{<}$  language. On the right part, a more standard notation for it.

A  $\text{MODS}$  encoding of a propositional formula mainly consists of the explicit representation of the set of its models over the set of variables occurring in it. Figure 4 depicts a formula of  $\text{MODS}$  using the NNF representation; this formula is equivalent to the DNF formula  $(a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge \neg c)$ .



**Figure 4.** A formula from the  $\text{MODS}$  fragment.

**3 THE CLAUSAL ENTAILMENT PROBLEM**

One of the main applications of compiling a propositional formula is to enhance the efficiency of clausal entailment (a key query for propositional reasoning). Accordingly, much effort has been devoted

to the design and the evaluation of compilation functions *comp* targeting propositional fragments satisfying **CE**. One often makes a distinction between the approaches satisfying **CE** (called “exact compilation methods”) and approaches ensuring that a (proper) subset of all clausal queries can be answered exactly in polynomial time (“approximate compilation methods”). In the first case, *comp* is said to be equivalence-preserving. In the second case, the subset of queries under consideration is not explicitly given as part of the entailment problem; it can be either intensionally characterized, for instance as the set of all clauses generated from a given set of variables or literals, all the Horn clauses, all the clauses from  $k$ -CNF (for a fixed parameter  $k$ ), etc. It can also be unspecified at the start and derived as a consequence of the compilation technique under used. For instance, one can associate to any propositional formula  $\Sigma$  one of the logically weakest  $\text{HORN-CNF}$  formula

$$\text{comp}_l(\Sigma)$$

implying  $\Sigma$  as well as one of the logically strongest  $\text{HORN-CNF}$  formula

$$\text{comp}_u(\Sigma)$$

implied by  $\Sigma$  [75, 76]. While  $\text{comp}_u(\Sigma)$  is unique up to logical equivalence, exponentially many non-equivalent  $\text{comp}_l(\Sigma)$  may exist. Once  $\text{comp}_l(\Sigma)$  and  $\text{comp}_u(\Sigma)$  are computed, they can be used to answer in polynomial time every clausal query  $\gamma$  s.t.  $\text{comp}_l(\Sigma) \not\models \gamma$  (1) or  $\text{comp}_l(\Sigma) \models \gamma$  (2) (due to the transitivity of  $\models$ ); the answer is “no” in case (1) and “yes” in case (2). Horn (clausal) queries can be answered exactly using  $\text{comp}_l(\Sigma)$ . Such approaches have been extended to other (incomplete, yet tractable) fragments of propositional logic and to the first-order case (see e.g. [30, 31, 32, 9, 77, 33]).

Both families of methods (i.e., the “exact” methods and the “approximate” ones) include approaches based on prime implicates (or the dual concept of prime implicants), see [71]. As explained in the previous section, the prime implicates (resp. the prime implicants) of a formula are the logically strongest clauses (resp. the logically weakest terms) implied by (resp. implying) the formula (one representative per equivalence class is considered, only). Introduced by Quine in the 50’s, those two notions have been heavily used in Computer Science (because they are key notions for the problem of minimizing circuits).

Many algorithms for computing prime implicates/prime implicants have been designed (see [58] for a survey). Like the conjunctive formulas  $\alpha_1 \wedge \dots \wedge \alpha_n$  which are decomposable NNF formulas, the prime implicates formulas satisfy a (first) *separability property* (the conjunctive one) stating that:

a clause  $\gamma$  is a logical consequence of a conjunctive formula  $\alpha_1 \wedge \dots \wedge \alpha_n$  if and only if there exists  $i \in 1 \dots n$  s.t.  $\gamma$  is a logical consequence of  $\alpha_i$ .

A similar (yet distinct) second separability property (the disjunctive one) is satisfied by all disjunctive formulas from NNF:

a formula  $\alpha$  is a logical consequence of a disjunctive formula  $\alpha_1 \vee \dots \vee \alpha_n$  if and only if for every  $i \in 1 \dots n$  s.t.  $\alpha$  is a logical consequence of  $\alpha_i$ .

Beyond  $\text{PI}$ , those two separability properties underly many target fragments for  $\text{KC}$  satisfying **CE**, especially  $\text{DNNF}$  [24] (and its subsets  $\text{d-DNNF}$ ,  $\text{OBDD}_{<}$ ,  $\text{DNF}$ ,  $\text{MODS}$ ). This can be explained by considering Shannon decompositions of formulas. Indeed, the Shannon decomposition of a formula  $\Sigma$  over a variable  $x$  is a formula

$$(\neg x \wedge (\Sigma \mid \neg x)) \vee (x \wedge (\Sigma \mid x))$$

equivalent to  $\Sigma$  exhibiting several separable subformulas:

- $\neg x \wedge (\Sigma \mid \neg x)$  and  $x \wedge (\Sigma \mid x)$  are disjunctively separable in it;
- $\neg x$  and  $\Sigma \mid \neg x$  are conjunctively separable in the subformula  $\neg x \wedge (\Sigma \mid \neg x)$ ;
- $x$  and  $\Sigma \mid x$  are conjunctively separable in the subformula  $x \wedge (\Sigma \mid x)$ .

Performing in a systematic way the Shannon decomposition of a formula  $\Sigma$  over the variables occurring in it leads to formulas from fragments satisfying **CE**. Using or not a fixed decomposition ordering gives rise to distinct fragments. Since the DPLL procedure [28] for the satisfiability problem SAT can be used to do such decompositions, several target fragments for KC can be characterized by the traces achieved by this search procedure [46].

Note that the two separability properties can be combined to other approaches for improving clausal entailment from the computational side, like the restriction-based ones, e.g. the approach consisting in focusing on HORN-CNF formulas. Thus, [57, 59] generalize the notion of prime implicates to theory prime implicates. Basically, a theory  $\Phi$  for a formula  $\Sigma$  is any logical consequence of  $\Sigma$  satisfying **CE**; for instance, if  $\Sigma$  is a CNF formula, an associated theory is the conjunction of all its Horn clauses. The theory prime implicates of  $\Sigma$  w.r.t.  $\Phi$  are the logically strongest clauses implied by  $\Sigma$  modulo  $\Phi$  (i.e., classical entailment is replaced by entailment modulo  $\Phi$ ). The compiled form of  $\Sigma$  is given by the pair consisting of  $\Phi$  and the set of theory prime implicates of  $\Sigma$  modulo  $\Phi$  (keeping one representative per equivalence class modulo  $\Phi$ ). Clausal queries  $\gamma$  can be answered in polynomial time from such a compiled form since  $\Sigma \models \gamma$  holds if and only if there exists a theory prime implicate  $\delta$  of  $\Sigma$  w.r.t.  $\Phi$  such that

$$\Phi \wedge \delta \models \gamma$$

holds (a generalization of the first separability property). Since each  $\delta$  is a clause  $l_1 \vee \dots \vee l_k$ , this amounts to determining whether

$$\Phi \models \sim l_i \vee \gamma$$

for  $i \in 1 \dots k$  (here  $\sim l_i$  denotes the complementary literal of  $l_i$ ). The fact that  $\Phi$  satisfies **CE** concludes the proof. Many algorithms for computing prime implicates can be extended to the theory prime implicates case [57, 59].

Another approach consists in taking advantage of the second separability property to derive compiled forms as DNF formulas [74] or more generally as disjunctions of formulas satisfying **CE**, for instance formulas from the HORN-CNF[ $\vee$ ] fragment, which consists of so-called *Horn covers*, i.e., disjunctions of HORN-CNF formulas (see [10, 37] for details). Again, case analysis at work in the DPLL procedure can be used for generating such compiled forms.

Finally, other compilation techniques aim at exploiting the power of unit-resolution (which is refutationally complete for a more general class than the set of all Horn CNF formulas) so as to make clausal entailment tractable. The idea is to add some clauses (typically some prime implicates of the input CNF formula  $\Sigma$ ) to  $\Sigma$  so as to ensure the unit-refutation completeness of the resulting compiled form. See [60, 61, 29, 72, 44] for approaches based on this idea.

All those techniques prove interesting in the sense that they allow to improve clausal entailment for some propositional formulas even if none of them ensures such an improvement for every propositional formula (especially because for each of them, once can find families of formulas such that the sizes of the compiled forms of the formulas

from such families are not polynomially bounded in the sizes of the formulas). As we will see in the next section, it seems that one cannot do better (under the standard assumptions of complexity theory). All the techniques sketched above are typically incomparable, which means that the sets of instances for which they are helpful cannot be compared w.r.t. inclusion.

## 4 THE COMPILABILITY ISSUE

Roughly speaking, a decision problem is said to be compilable to a given complexity class  $\mathbf{C}$  if it is in  $\mathbf{C}$  once the fixed part of any instance has been pre-processed, i.e., turned off-line into a data structure of size polynomial in the input size. As explained in the introduction, the fact that the pre-processing leads to a compiled form of polynomial space is crucial.

In order to formalize such a notion of compilability, Cadoli and his colleagues introduced new classes (compilability classes) organized into hierarchies (which echo PH) and the corresponding reductions (see the excellent pieces of work in [16, 13, 14, 53, 15]). This allows to classify many AI problems as compilable to a class  $\mathbf{C}$ , or as not compilable to  $\mathbf{C}$  (usually under standard assumptions of complexity theory - the fact that the polynomial hierarchy PH does not collapse).

First of all, in order to address the compilability of a decision problem, one needs to consider it as a language  $L$  of pairs  $\langle \Sigma, \alpha \rangle$ : the *fixed part*  $\Sigma$  will be subject to pre-processing, while the remaining *varying part*  $\alpha$  will not. For instance, considering the clausal entailment problem, a standard partition consists in taking a formula  $\Sigma$  (representing a belief base) as the fixed part and a clause or even a CNF formula  $\alpha$  as the varying one; this just reflects the fact that the base typically changes less often than the queries.

Several families of classes can be considered as candidates to represent what “compilable to  $\mathbf{C}$ ” means. One of them is the family of **compC** classes:

**Definition 10.** *Let  $\mathbf{C}$  be a complexity class closed under polynomial many-one reductions and admitting complete problems for such reductions. A language of pairs  $L$  belongs to **compC** if and only if there exists a polysize function  $comp$ <sup>3</sup> and a language of pairs  $L' \in \mathbf{C}$  such that  $\langle \Sigma, \alpha \rangle \in L$  if and only if  $\langle comp(\Sigma), \alpha \rangle \in L'$ .*

Obviously enough, for every admissible complexity class  $\mathbf{C}$ , we have the inclusion  $\mathbf{C} \subseteq \mathbf{compC}$  (choose  $comp$  as the identity function).

In order to prove the membership to **compC** of a problem it is enough to follow the definition (hence, exhibiting a polysize compilation function  $comp$ ). Things are more complex to prove that a given problem does *not* belong to some **compC** (under the standard assumptions of complexity theory). In this purpose, an interesting tool consists of the non-uniform complexity classes **C/poly**:

**Definition 11.** *An advice-taking Turing machine is a Turing machine that has associated with it a special “advice oracle”  $A$ , which can be any function (not necessarily a recursive one). On input  $s$ , a special “advice tape” is automatically loaded with  $A(|s|)$  and from then on the computation proceeds as normal, based on the two inputs,  $s$  and  $A(|s|)$ . An advice-taking Turing machine uses polynomial advice if its advice oracle  $A$  satisfies  $|A(n)| \leq p(n)$  for some fixed polynomial  $p$  and all non-negative integers  $n$ . If  $\mathbf{C}$  is a class of languages defined in terms of resource-bounded Turing machines, then **C/poly***

<sup>3</sup> A function  $comp$  is polysize if and only if there exists a polynomial  $p$  such that  $|comp(\Sigma)|$  is bounded by  $p(|\Sigma|)$  for every  $\Sigma$  in the domain of  $comp$ .

is the class of languages defined by Turing machines with the same resource bounds but augmented by polynomial advice.

It has been proven that if  $\text{NP} \subseteq \text{P/poly}$  then  $\Pi_2^P = \Sigma_2^P$  (hence PH collapses at the second level) [47], and that if  $\text{NP} \subseteq \text{coNP/poly}$  then  $\Pi_3^P = \Sigma_3^P$  (hence PH collapses at the third level) [80]. Based on the result by Karp and Lipton [47], one can prove that the language of pairs  $\langle \Sigma, \alpha \rangle$  where  $\Sigma$  is a propositional formula and  $\alpha$  is a CNF formula (corresponding to the clausal entailment problem where the input is splitted into a fixed part  $\Sigma$  and a varying part  $\alpha$ ) does not belong to  $\text{compP}$  unless PH collapses at the second level. The proof is as follows [48]: let  $n$  be any non-negative integer. Let  $\Sigma_n^{\text{max}}$  be the CNF formula

$$\bigwedge_{\gamma_i \in 3-C_n} \neg \text{holds}_i \vee \gamma_i$$

where  $3 - C_n$  is the set of all 3-literal clauses that can be generated from  $\{x_1, \dots, x_n\}$  and the  $\text{holds}_i$  are new variables, not among  $\{x_1, \dots, x_n\}$ . The size  $|\Sigma_n^{\text{max}}|$  of  $\Sigma_n^{\text{max}}$  is in  $\mathcal{O}(n^3)$  since  $\Sigma_n^{\text{max}}$  contains  $\mathcal{O}(n^3)$  clauses containing four literals. Obviously enough, each 3-CNF formula  $\alpha_n$  built up from the set of variables  $\{x_1, \dots, x_n\}$  is in bijection with the subset  $S_{\alpha_n}$  of the variables  $\text{holds}_i$  s.t.  $\gamma_i$  is a clause of  $\alpha_n$  if and only if  $\text{holds}_i \in S_{\alpha_n}$  (each variable  $\text{holds}_i$  can be viewed as the name of the clause where it appears, hence selecting a clause just amounts to selecting its name). Let  $\gamma_{\alpha_n}$  be the clause

$$\bigvee_{\text{holds}_i \in S_{\alpha_n}} \neg \text{holds}_i.$$

It is easy to check that  $\alpha_n$  is unsatisfiable if and only if

$$\Sigma_n^{\text{max}} \models \gamma_{\alpha_n}.$$

Suppose now that we have a polysize compilation function  $\text{comp}$  such that for every propositional formula  $\Sigma$  and every clause  $\gamma$ , determining whether  $\text{comp}(\Sigma) \models \gamma$  can be done in (deterministic) polynomial time. Then we would be able to determine whether any 3-CNF formula  $\alpha$  is satisfiable using a deterministic Turing machine with a polynomial advice  $A$ : if  $|\text{Var}(\alpha)| = n$ , then the machine loads

$$A(n) = \text{comp}(\Sigma_n^{\text{max}}).$$

Once this is done, it determines whether  $\gamma_{\alpha}$  is entailed by  $\text{comp}(\Sigma_n^{\text{max}})$ , which is in P. Since 3-SAT is complete for NP, this would imply  $\text{NP} \subseteq \text{P/poly}$ , and, as a consequence, the polynomial hierarchy would collapse at the second level.

The size of the language of varying parts has clearly an impact on the compilability issue; indeed, if for each fixed part  $\Sigma$ , there are only polynomially many possible varying parts  $\alpha$ , then the corresponding language of pairs  $L$  is compilable to P (i.e., it belongs to  $\text{compP}$ ). Indeed, it is enough during the off-line phase to consider successively every  $\alpha$  and to store it in a lookup-table  $\text{comp}(\Sigma)$  whenever  $\langle \Sigma, \alpha \rangle$  belongs to  $L$ . For every  $\Sigma$ , the size of  $\text{comp}(\Sigma)$  is polynomially bounded in the size of  $\Sigma$  and determining on-line whether  $\langle \Sigma, \alpha \rangle \in L$  amounts to a lookup operation. For instance, the restriction of the clausal entailment problem, viewed as a language of pairs where the varying part  $\alpha$  reduces to a literal (or more generally to a clause containing at most  $k$  literals, where  $k$  is a fixed parameter) belongs to  $\text{compP}$ . This shows that there exist problems which are intractable when considered “all-at-once” (literal entailment is  $\text{coNP}$ -complete in propositional logic) and tractable when they are viewed as languages of pairs (literal entailment is in  $\text{compP}$ ). By the

way, the fact that only polynomially many varying parts  $\alpha$  are possible is not a necessary condition for the membership to  $\text{compP}$ ; thus, the term entailment problem, viewed as a language of pairs where the varying part  $\alpha$  reduces to a term (or more generally to a  $k$ -CNF formula where  $k$  is a fixed parameter) belongs to  $\text{compP}$  since implying a conjunction simply amounts to implying each of its conjuncts ([62] would say that the set of all terms – and more generally every  $k$ -CNF language – has an efficient basis). Clearly, the number of terms over a set of  $n$  atoms is not polynomially bounded in  $n$ .

A notion of  $\text{comp}$ -reduction suited to the compilability classes  $\text{compC}$  has been pointed out, and the existence of complete problems for such classes proven. However, it appears that many non-compilability results from the literature cannot be rephrased as  $\text{compC}$ -completeness results. For instance, it is unlikely that clausal entailment is  $\text{compcoNP}$ -complete (it would make  $\text{P} = \text{NP}$ ). In order to go further, one needs to consider the compilability classes  $\text{nu-compC}$  [15]:

**Definition 12.** Let  $\mathbf{C}$  be a complexity class closed under polynomial many-one reductions and admitting complete problems for such reductions. A language of pairs  $L$  belongs to  $\text{nu-compC}$  if and only if there exists a binary polysize function  $\text{comp}$  and a language of pairs  $L' \in \mathbf{C}$  such that for all  $\langle \Sigma, \alpha \rangle \in L$ , we have:

$$\langle \Sigma, \alpha \rangle \in L \text{ if and only if } \langle \text{comp}(\Sigma, |\alpha|), \alpha \rangle \in L'.$$

Here “nu” stands for “non-uniform”, which indicates that the compiled form of  $\Sigma$  may also depend on the size of the varying part  $\alpha$ . Observe that this was the case in the proof showing that clausal entailment is not in  $\text{compP}$  unless PH collapses (each compiled form  $\Sigma_n^{\text{max}}$  depends on  $n$ , the number of variables on which  $\alpha$  is built).

Obviously enough, for each admissible complexity class  $\mathbf{C}$ , we have the inclusions  $\text{compC} \subseteq \text{nu-compC}$  (showing that  $\mathbf{C} \subseteq \text{nu-compC}$ ) and  $\mathbf{C/poly} \subseteq \text{nu-compC}$ . Furthermore, under reasonable assumptions (see [15]), all those inclusions are strict. Thus the  $\text{nu-compC}$  classes generalize the previous compilability classes.

A notion of non-uniform  $\text{comp}$ -reduction suited to the compilability classes  $\text{nu-compC}$  has also been pointed out (it includes the notion of (uniform)  $\text{comp}$ -reduction), and the existence of complete problems for such classes. For instance, the clausal entailment problem is  $\text{nu-compcoNP}$ -complete.

Inclusion of compilability classes  $\mathbf{C/poly}$ ,  $\text{compC}$ ,  $\text{nu-compC}$  similar to those holding in the polynomial hierarchy exist (see [15]). It is strongly believed that the corresponding compilability hierarchies are proper: if one of them collapses, then the polynomial hierarchy collapses at well (cf. Theorem 2.12 from [15]). For instance, if the clausal entailment problem is in  $\text{nu-compP}$ , then the polynomial hierarchy collapses. Accordingly, in order to show that a problem is not in  $\text{compC}$ , it is enough to prove that it is  $\text{nu-compC}'$ -hard, where  $\mathbf{C}'$  is located higher than  $\mathbf{C}$  in the polynomial hierarchy. Since complete problems for any  $\text{nu-compC}$  class can be easily derived from complete problems for the corresponding class  $\mathbf{C}$  of the polynomial hierarchy,  $\text{nu-compC}$ -complete problems appear as a very interesting tool for proving non-compilability results.

The compilability of a number of AI problems, including diagnosis, planning, abduction, belief revision, closed-world reasoning, and paraconsistent inference from belief bases has been investigated in the following papers [51, 50, 13, 52, 15, 64, 22, 27, 54, 55]. Other compilability classes incorporating a notion of parameterization have been pointed out and studied in [20].

## 5 THE KNOWLEDGE COMPILATION MAP

An important issue to be addressed in KC is the choice of a *target language*, i.e., the representation language of compiled forms. Obviously, this is a domain-dependent issue since it depends on the tasks we would like to improve via KC, computationally speaking. However, as explained in the introductory section, many tasks can be decomposed into domain-independent basic queries and transformations, so that one can focus on such queries and transformations instead of the tasks themselves.

Roughly speaking, a query is an operation that returns information about a compiled form without changing it. A transformation, on the other hand, is an operation that returns a modified compiled form, which is then operated on using queries.

The choice of a target language for KC is typically based on a number of different criteria:

- the computational complexity of the queries of interest when the fixed part is represented in the KC language
- the computational complexity of the transformations of interest when the fixed part is represented in the KC language
- the expressiveness and the spatial efficiency (succinctness) of the language.

The KC map presented in [26] is such a multi-criteria evaluation of dozen propositional fragments, i.e., subsets of a propositional language NNF.

The queries considered in [26] are tests for consistency **CO** (i.e., the SAT problem), validity **VA**, implicates (clausal entailment) **CE**, implicants **IM**, equivalence **EQ**, and sentential entailment **SE**. Models counting **MC** and models enumeration **ME** have also been considered.

**Definition 13.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **VA** if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  to 1 if  $\Sigma$  is valid, and to 0 otherwise.

A number of target compilation languages support a polynomial-time equivalence/sentential entailment test:

**Definition 14.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **EQ (SE)** if and only if there exists a polynomial-time algorithm that maps every pair of formulas  $\Sigma, \alpha$  from  $\mathbf{L}$  to 1 if  $\Sigma \equiv \alpha$  ( $\Sigma \models \alpha$ ) holds, and to 0 otherwise.

Note that sentential entailment (**SE**) is stronger than clausal entailment and equivalence. Therefore, if a language  $\mathbf{L}$  satisfies **SE**, it also satisfies **CE** and **EQ**.

The following dual to **CE** has also been considered:

**Definition 15.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **IM** if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  and every term  $\gamma$  from NNF to 1 if  $\gamma \models \Sigma$  holds, and to 0 otherwise.

The next queries also prove important in many applications:

**Definition 16.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **CT** if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  to a nonnegative integer that represents the number of models of  $\Sigma$  (in binary notation).

**Definition 17.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **ME** if and only if there exists a polynomial  $p(\cdot, \cdot)$  and an algorithm that outputs all models of an arbitrary formula  $\Sigma$  from  $\mathbf{L}$  in time  $p(n, m)$ , where  $n$  is the size of  $\Sigma$  and  $m$  is the number of its models (over variables occurring in  $\Sigma$ ).

A number of transformations have also been considered in [26], especially closures:

**Definition 18.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies  $\wedge \mathbf{C}$  ( $\vee \mathbf{C}$ ) iff there exists a polynomial-time algorithm that maps every finite set of formulas  $\Sigma_1, \dots, \Sigma_n$  from  $\mathbf{L}$  to a formula of  $\mathbf{L}$  that is logically equivalent to  $\Sigma_1 \wedge \dots \wedge \Sigma_n$  ( $\Sigma_1 \vee \dots \vee \Sigma_n$ ).

**Definition 19.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies  $\neg \mathbf{C}$  if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  to a formula of  $\mathbf{L}$  that is logically equivalent to  $\neg \Sigma$ .

If a language satisfies one of the above properties, it is said to be *closed* under the corresponding operator. Closure under logical connectives is important for two key reasons. First, it has implications on how compilers are constructed for a given target language. For example, if a clause can be easily compiled into some language  $\mathbf{L}$ , then closure under conjunction implies that compiling a CNF formula into  $\mathbf{L}$  is easy. Second, it has implications on the class of polynomial-time queries supported by the target language: if a language  $\mathbf{L}$  satisfies **CO** and is closed under negation and conjunction, then it must satisfy **SE** (to test whether  $\Sigma \models \alpha$ , all we have to do, by the refutation theorem, is test whether  $\Sigma \wedge \neg \alpha$  is inconsistent). Similarly, if a language satisfies **VA** and is closed under negation and disjunction, it must satisfy **SE** by the deduction theorem.

It is important to stress here that some languages are closed under a logical operator, only if the number of operands is bounded by a constant. We will refer to this as *bounded closure*.

**Definition 20.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies  $\wedge \mathbf{BC}$  ( $\vee \mathbf{BC}$ ) iff there exists a polynomial-time algorithm that maps every pair of formulas  $\Sigma_1$  and  $\Sigma_2$  from  $\mathbf{L}$  to a formula of  $\mathbf{L}$  that is logically equivalent to  $\Sigma_1 \wedge \Sigma_2$  ( $\Sigma_1 \vee \Sigma_2$ ).

A key transformation is that of *forgetting* (which can be viewed as closure under existential quantification) [56] [49]:

**Definition 21.** Let  $\Sigma$  be a propositional formula, and let  $\mathbf{X}$  be a subset of variables from PS. The forgetting of  $\mathbf{X}$  from  $\Sigma$ , denoted  $\exists \mathbf{X}.\Sigma$ , is a formula that does not mention any variable from  $\mathbf{X}$  and for every formula  $\alpha$  that does not mention any variable from  $\mathbf{X}$ , we have  $\Sigma \models \alpha$  precisely when  $\exists \mathbf{X}.\Sigma \models \alpha$ .

Therefore, to forget variables from  $\mathbf{X}$  is to remove any reference to  $\mathbf{X}$  from  $\Sigma$ , while maintaining all information that  $\Sigma$  captures about the complement of  $\mathbf{X}$ . Note that  $\exists \mathbf{X}.\Sigma$  is unique up to logical equivalence.

**Definition 22.** Let  $\mathbf{L}$  be a subset of NNF.  $\mathbf{L}$  satisfies **FO** if and only if there exists a polynomial-time algorithm that maps every formula  $\Sigma$  from  $\mathbf{L}$  and every subset  $\mathbf{X}$  of variables from PS to a formula from  $\mathbf{L}$  equivalent to  $\exists \mathbf{X}.\Sigma$ . If the property holds for singleton  $\mathbf{X}$ , we say that  $\mathbf{L}$  satisfies **SFO**.

Forgetting is an important transformation as it allows to focus/project a formula on a set of variables. It has numerous applications, including diagnosis, planning, reasoning under inconsistency, etc.

Other important criteria for the evaluation of target languages for KC are the expressiveness one and the succinctness one [43]. Those two notions are modeled by pre-orders over the subsets of the propositional language used as a baseline (here NNF):

**Definition 23.** Let  $\mathbf{L}_1$  and  $\mathbf{L}_2$  be two subsets of NNF.  $\mathbf{L}_1$  is at least as expressive as  $\mathbf{L}_2$ , denoted  $\mathbf{L}_1 \leq_e \mathbf{L}_2$ , iff for every formula  $\alpha \in \mathbf{L}_2$ , there exists an equivalent formula  $\beta \in \mathbf{L}_1$ .

Expressiveness captures the ability of encoding information; the minimal elements w.r.t.  $\leq$  of the power set of NNF are called *complete fragments*. In each of them, every Boolean function can be represented: complete fragments are fully expressive. Many of the fragments considered in Section 2 are complete ones. Such fragments are interesting as target languages for KC because one does not have to care about situations when the propositional information to be compiled cannot be represented within the fragment.

Now, there also exist a number of valuable *incomplete* fragments, like KROM–CNF (also known as the binary fragment since it consists of conjunctions of clauses containing at most two literals), HORN–CNF (as considered in the previous sections) and the affine fragment. Their importance comes from the fact that they offer polynomial-time queries and transformations which are not guaranteed by all the complete fragments [37], especially CE.

Succinctness is a refinement of expressiveness which considers the representation sizes:

**Definition 24.** Let  $\mathbf{L}_1$  and  $\mathbf{L}_2$  be two subsets of NNF.  $\mathbf{L}_1$  is at least as succinct as  $\mathbf{L}_2$ , denoted  $\mathbf{L}_1 \leq_s \mathbf{L}_2$ , iff there exists a polynomial  $p$  such that for every formula  $\alpha \in \mathbf{L}_2$ , there exists an equivalent formula  $\beta \in \mathbf{L}_1$  where  $|\beta| \leq p(|\alpha|)$ .

Observe that the definition does not require that there exists a function *comp* computing  $\beta$  given  $\alpha$  in *polynomial time*; it is only asked that a *polysize* function *comp* exists.

The KC map gathers the multi-criteria evaluations of a number of propositional fragments, including many of those considered in Section 2, as well as other fragments. Examples of results which can be found in the KC map are:

**Proposition 1.**

DNNF satisfies **CO, CE, ME, CD, FO,  $\vee$ C**.  
 $\text{d-DNNF}$  satisfies **CO, VA, CE, IM, CT, ME, CD**.  
 OBDD $_{<}$  satisfies **CO, VA, CE, IM, EQ, CT, ME, CD, SFO,  $\wedge$ BC,  $\vee$ BC,  $\neg$ C**.  
 DNF satisfies **CO, CE, ME, CD, FO,  $\wedge$ BC,  $\vee$ C**.  
 PI satisfies **CO, VA, CE, IM, EQ, SE, ME, CD, FO,  $\vee$ BC**.  
 IP satisfies **CO, VA, CE, IM, EQ, SE, ME, CD,  $\wedge$ BC**.  
 MODS satisfies **CO, VA, CE, IM, EQ, SE, CT, ME, CD, FO,  $\wedge$ BC**.

Many “negative results” can also be found in the KC map, e.g. the fact that DNNF does not satisfy any of **VA, IM, EQ, SE, CT,  $\wedge$ BC,  $\neg$ C** unless  $\mathbf{P} = \mathbf{NP}$ , and similar results for the other fragments.

The KC map also indicates how the fragments compare one another w.r.t. expressiveness/succinctness. For instance, we have the following strict succinctness ordering:

$$\text{DNNF} <_s \text{d-DNNF} <_s \text{OBDD}_{<} <_s \text{MODS}$$

Here, “positive” translatability results (i.e., showing that  $\mathbf{L}_1 \leq_s \mathbf{L}_2$  for some given languages  $\mathbf{L}_1$  and  $\mathbf{L}_2$ ) can be obtained by exhibiting a polysize translation function *comp*; “negative” results (i.e., showing that  $\mathbf{L}_2 \not\leq_s \mathbf{L}_1$ ) can be obtained using combinatorial arguments... but also non-compileability results! For instance, in order to show that DNNF  $\not\leq_s$  CNF unless PH collapses, it is enough to remember that DNNF satisfies CE and that the clausal entailment problem from CNF formulas is not in compP unless PH collapses.

Based on the KC map, the choice of a target language for the KC purpose can be realised by listing first the queries and transformations required by the application under consideration and by considering the expressiveness/succinctness criteria. For instance, as explained in the introduction, the consistency-based diagnoses of a

system can be computed as the models of the formula obtained by forgetting every elementary proposition except those denoting the components states in the formula obtained by conditioning the system description by the available observations. Thus **ME, FO, CD** are required by this application. Among the fragments ensuring them are DNNF, DNF, PI, MODS. The fact that DNNF and PI are the most succinct ones<sup>4</sup> can be viewed as an explanation of the success of those two fragments for model-based diagnostic.

For further extensions of the KC map, the reader might look at the following papers, where other propositional fragments have been considered and investigated following the criteria at work in the KC map: [79, 35, 78, 70, 37, 36].

## 6 BEYOND PURE PROPOSITIONAL REASONING

Quite recently, KC has been considered as an approach for improving some computational tasks which go beyond pure propositional reasoning. This includes works where the underlying setting is still propositional but so to say, non-classical, and also works escaping from the propositional case.

In the first family, a number of papers have been centered on the issue of *reasoning under inconsistency*. Inconsistency-tolerant reasoning is one of the major topic in AI from its very beginning (especially because it is closely connected to the problem of reasoning in presence of exceptions). It is well-known that classical propositional inference (i.e., the  $\models$  relation) is inadequate for the task of reasoning under inconsistency because it trivializes: every formula is a logical consequence of any inconsistent formula (this is the famous *ex falso quodlibet sequitur*). This calls for other approaches and many of them can be found in the AI literature: paraconsistent logics, belief revision, belief merging, reasoning from preferred consistent subsets, knowledge integration, argumentative logics, etc. The variety of existing approaches can be explained by the fact that there is no consensus on what should be concluded from an inconsistent formula, and that paraconsistency can be achieved in various ways, depending on the exact nature of the problem at hand (hence, the available information). Each of them has its own pros and cons, and is more or less suited to different inconsistency handling scenarios.

Now, beyond the problem of modeling what reasoning from inconsistent pieces of information should be, it turns out that, unfortunately, it is difficult from a computational point of view. Thus, in the propositional case, it is at least as hard as classical inference and typically harder than it. This explains why KC has been considered as a resort here.

As a matter of example, I focus in the following on the inference problem from (propositional) stratified belief bases; this illustration gives also the opportunity to present, so to say, “at work” some notions described in the previous sections (especially the concept of compileability and several target languages for KC).

**Definition 25.** A stratified belief base (SBB)  $\Sigma$  is an ordered pair  $\Sigma = \langle \Delta, \leq \rangle$ , where  $\Delta = \{\phi_1, \dots, \phi_n\}$  is a finite set of formulas from  $\text{PROP}_{\mathbf{P}}\mathbf{S}$  and  $\leq$  is a total pre-order over  $\Delta$  (i.e., a reflexive and transitive relation over  $\Delta$  s.t. for every  $\phi_i, \phi_j$  belonging to  $\Delta$ , we have  $\phi_i \leq \phi_j$  or  $\phi_j \leq \phi_i$ ). Every subset  $S$  of  $\Delta$  is a subbase of  $\Sigma$ .

It is equivalent to define  $\Sigma$  as a finite sequence  $(\Delta_1, \dots, \Delta_k)$  of subbases of  $\Delta$ , where each  $\Delta_i$  ( $i \in 1 \dots k$ ) is the non-empty set

<sup>4</sup> We know that PI is not strictly more succinct than DNNF, but we ignore whether the converse also holds.

which contains all the minimal elements of  $\Delta \setminus (\bigcup_{j=1}^{i-1} \Delta_j)^5$  w.r.t.  $\leq$ .  $(\Delta_1, \dots, \Delta_k)$  can be computed in time polynomial in the size of  $\langle \Delta, \leq \rangle$  (assuming that  $\leq$  is explicitly represented as a set of pairs). Clearly enough,  $\{\Delta_1, \dots, \Delta_k\}$  is a partition of  $\Delta$ . Each subset  $\Delta_i$  ( $i \in 1 \dots k$ ) is called a *stratum* of  $\Sigma$ , and  $i$  is the priority level of each formula of  $\Delta_i$ . Intuitively, the lower the priority level of a formula the higher its plausibility. Given a subbase  $S$  of  $\Sigma$ , we note  $S_i$  ( $i \in 1 \dots k$ ) the subset of  $S$  defined by  $S_i = S \cap \Delta_i$ . We also note  $Var(\Sigma) = \bigcup_{i=1}^k Var(\Delta_i)$ .

In the following, we assume that  $\Delta_1$  is a singleton, consisting of the (consistent) conjunction of all certain beliefs (i.e., the pieces of *knowledge*) of  $\Delta$ . Slightly abusing notations, we will identify  $\Delta_1$  with the (unique) formula it contains. Note that this assumption can be done without loss of generality since when no certain beliefs are available, it is sufficient to add *true* to  $\Delta$  as its unique minimal element w.r.t.  $\leq$  (and this will not lead to any significant computational overhead).

There are several ways to use the information given by an SBB corresponding to several epistemic attitudes. Inference from an SBB  $\Sigma$  is typically considered as a two-step process, consisting first in characterizing some preferred consistent subbases of  $\Sigma$  and then considering classical inference from some of those subbases. Many policies (or generation mechanisms) for the selection of preferred consistent subbases can be defined. In formal terms, a policy  $\mathcal{P}$  is a mapping that associates to every SBB  $\Sigma$  a set  $\Sigma_{\mathcal{P}}$  consisting of all the preferred consistent subbases of  $\Sigma$  w.r.t.  $\mathcal{P}$ . Four policies are often considered: the *possibilistic* policy, the *linear order* policy, the *inclusion-preference* policy, and the *lexicographic* policy.

**Definition 26.** Let  $\Sigma = (\Delta_1, \dots, \Delta_k)$  be an SBB.

- The set  $\Sigma_{\mathcal{PO}}$  of all the preferred subbases of  $\Sigma$  w.r.t. the *possibilistic* policy is the singleton  $\{\bigcup_{i=1}^{s-1} \Delta_i\}$ , where  $s$  is the lowest integer ( $1 \leq s \leq k$ ) s.t.  $\bigcup_{i=1}^s \Delta_i$  is inconsistent.
- The set  $\Sigma_{\mathcal{LO}}$  of all the preferred subbases of  $\Sigma$  w.r.t. the *linear order* policy is the singleton  $\{\bigcup_{i=1}^k \Delta_i\}$ , where  $\Delta_i$  ( $i \in 1 \dots k$ ) is defined by  $\Delta_i = \Delta_i$  if  $\Delta_i \cup \bigcup_{j=1}^{i-1} \Delta_j$  is consistent,  $\emptyset$  otherwise.
- The set  $\Sigma_{\subseteq}$  of all maximal (w.r.t.  $\subseteq$ ) consistent subbases of  $\Sigma$  containing  $\Delta_1$  is  $\{S \subseteq \Delta \mid S \text{ is consistent, } \Delta_1 \subseteq S, \text{ and } \forall \phi \in \Delta \setminus S, S \cup \{\phi\} \text{ is inconsistent}\}$ . Two valuable subsets of it are:
  - The set  $\Sigma_{\mathcal{IP}}$  of all the preferred subbases of  $\Sigma$  w.r.t. the *inclusion-preference* policy is  $\{S \subseteq \Delta \mid S \text{ is consistent and } \forall St \subseteq \Delta \text{ s.t. } St \neq S \text{ and } St \text{ is consistent, } \forall i \in 1 \dots k ((\forall j < i (St_j = S_j)) \Rightarrow S_i \not\subseteq St_i)\}$ .
  - The set  $\Sigma_{\mathcal{LE}}$  of all the preferred subbases of  $\Sigma$  w.r.t. the *lexicographic* policy is  $\{S \subseteq \Delta \mid S \text{ is consistent and } \forall St \subseteq \Delta \text{ s.t. } St \neq S \text{ and } St \text{ is consistent, } \forall i \in 1 \dots k ((\forall j < i (card(St_j) = card(S_j))) \Rightarrow card(S_i) \not\leq card(St_i))\}$ .

All these selection policies have their own motivations and lead to inference relations that are more or less satisfying from a logical point of view and from the computational complexity point of view (see [4, 5]).

Now, given a selection policy, several entailment principles can be considered, especially credulous inference, argumentative inference, skeptical inference. Among them, skeptical inference leads to inference relations which are at least preferential ones, i.e., relations satisfying “reflexivity”, “left logical equivalence”, “right weakening”, “cut”, “cautious monotony” and “or” (see [4]):

<sup>5</sup> By convention,  $\bigcup_{j=1}^0 \Delta_j = \emptyset$ .

**Definition 27.** Let  $\Sigma = (\Delta_1, \dots, \Delta_k)$  be an SBB,  $\mathcal{P}$  a policy for the generation of preferred subbases, and  $\alpha$  a formula from  $PROP_{PS}$ .  $\alpha$  is a (skeptical) consequence of  $\Sigma$  w.r.t.  $\mathcal{P}$ , noted  $\Sigma \vdash_{\mathcal{P}} \alpha$ , if and only if  $\forall S \in \Sigma_{\mathcal{P}}, S \models \alpha$ .

Unfortunately, whatever the selection policy among  $\mathcal{PO}, \mathcal{LO}, \mathcal{IP}, \mathcal{LE}$ , skeptical inference is not tractable (under the standard assumptions of complexity theory) [63, 17]:

**Definition 28.** Let  $\vdash_{\mathcal{P}}^{\mathcal{P}}$  be any inference relation from  $\{\vdash_{\mathcal{P}}^{\mathcal{PO}}, \vdash_{\mathcal{P}}^{\mathcal{LO}}, \vdash_{\mathcal{P}}^{\mathcal{IP}}, \vdash_{\mathcal{P}}^{\mathcal{LE}}\}$ .  $\vdash_{\mathcal{P}}^{\mathcal{P}}$  is the following decision problem:

- **Input:** An SBB  $\Sigma = (\Delta_1, \dots, \Delta_k)$  and a formula  $\alpha$  from  $PROP_{PS}$ .
- **Query:** Does  $\Sigma \vdash_{\mathcal{P}}^{\mathcal{P}} \alpha$  hold?

**Proposition 2.** The complexity of  $\vdash_{\mathcal{P}}^{\mathcal{P}}$  from an SBB for  $\mathcal{P} \in \{\mathcal{PO}, \mathcal{LO}, \mathcal{IP}, \mathcal{LE}\}$  is as reported in Table 1.

**Table 1.** Complexity of skeptical inference from SBBs (general case).

$\mathcal{P}$	$\vdash_{\mathcal{P}}^{\mathcal{P}}$
$\mathcal{PO}$	$\Theta_2^P$ -complete
$\mathcal{LO}$	$\Delta_2^P$ -complete
$\mathcal{IP}$	$\Pi_2^P$ -complete
$\mathcal{LE}$	$\Delta_2^P$ -complete

All the hardness results given in this proposition still hold when the query  $\alpha$  is restricted to a literal.

The following compilability results concerning inference from SBBs have been obtained [22]:

**Proposition 3.**

- $\vdash_{\mathcal{P}}^{\mathcal{PO}}, \vdash_{\mathcal{P}}^{\mathcal{LO}}$  and  $\vdash_{\mathcal{P}}^{\mathcal{LE}}$  with fixed  $\Delta$  and  $\leq$  and varying  $\alpha$  are in **compcoNP** but not in **compP** unless the polynomial hierarchy collapses at the second level.
- $\vdash_{\mathcal{P}}^{\mathcal{IP}}$  with fixed  $\Delta$  and  $\leq$  and varying  $\alpha$  is in **comp** $\Pi_2^P$  but not in **compcoNP** unless the polynomial hierarchy collapses at the third level.

The fact that  $\vdash_{\mathcal{P}}^{\mathcal{PO}}, \vdash_{\mathcal{P}}^{\mathcal{LO}}$  with fixed  $\Delta$  and  $\leq$  and varying  $\alpha$  are in **compcoNP** comes easily from the fact that for those two policies, there is only one preferred subbase of  $\Sigma$ ; the off-line phase amounts to computing it as the compiled form of  $\Sigma$  (then inference reduces to classical entailment). Furthermore, since those two inference problems include the problem of classical entailment (from a consistent formula), none of them can be in **compP** unless PH collapses. The case of  $\vdash_{\mathcal{P}}^{\mathcal{LE}}$  is a bit more elaborate.

Obviously enough, considering the stratification not fixed changes the picture; for instance:

**Proposition 4.**

- $\vdash_{\mathcal{P}}^{\mathcal{PO}}$  with fixed  $\Delta$  and varying  $\leq$  and  $\alpha$  is in **comp** $\Theta_2^P$  but not in **compcoNP** unless the polynomial hierarchy collapses at the third level.
- $\vdash_{\mathcal{P}}^{\mathcal{LO}}$  and  $\vdash_{\mathcal{P}}^{\mathcal{LE}}$  with fixed  $\Delta$  and varying  $\leq$  and  $\alpha$  are in **comp** $\Delta_2^P$  but not in **compcoNP** unless the polynomial hierarchy collapses at the third level.

Intuitively, as to  $\vdash_{\mathcal{V}}^{\mathcal{PO}}, \vdash_{\mathcal{V}}^{\mathcal{LO}}$ , there is no way to compute the preferred subbase of  $\Sigma$  during the off-line phase when  $\leq$  is not fixed; this explains the compilability shift.

From a more practical side, relaxing the size requirement on compiled forms imposed by the compilability setting, there exist compilation functions for turning SBBs into compiled ones from which skeptical inference (for some selection principles) is computationally easier than in the general case when queries are limited to clauses or CNF formulas. Of course, this does not show KC as helpful at the problem level (the previous non-compilability results showing that this is impossible) but renders it possible at the instance level (the situation is similar to what happens with the clausal entailment problem, which looks not compilable to  $\mathcal{P}$ , but for which compilation can nevertheless prove valuable for some instances).

Obviously enough, it is not possible to compile directly the conjunction of formulas of  $\Delta$  using one of the compilation functions presented in Section 3 since this conjunction typically is inconsistent (hence entailment from it would trivialize). In order to take advantage of the compilation functions *comp* considered so far for improving clausal entailment in the objective of compiling an SBB  $\Sigma = (\Delta_1, \dots, \Delta_k)$ , an approach consists in turning  $\Sigma$  into a so-called *comp*-compiled SBB  $comp(\Sigma) = (\Delta'_1, \dots, \Delta'_k)$  such that  $\Delta_1$  belongs to the target language associated to *comp* and  $\bigcup_{i=2}^k \Delta'_i$  is a consistent set of literals:

**Definition 29.** Let  $\Sigma = (\Delta_1, \dots, \Delta_k)$  be an SBB (with  $\Delta = \bigcup_{i=1}^k \Delta_i$ ) and let *comp* be any (equivalence-preserving) compilation function targetting a fragment satisfying **CE**. Without loss of generality, let us assume that every stratum  $\Delta_i$  ( $i \in 1 \dots k$ ) of  $\Sigma$  is totally ordered (w.r.t. any fixed order) and let us note  $\phi_{i,j}$  the  $j^{\text{th}}$  formula of  $\Delta_i$  w.r.t. this order. The SBB

$$comp(\Sigma) = (\chi_1, \dots, \chi_k)$$

where  $\chi_i = \{\text{holds}_{i,1}, \dots, \text{holds}_{i, \text{card}(\Delta_i)}\}$  for  $i \in 2 \dots k$ , each  $\text{holds}_{i,j} \in L_{PS} \setminus L_{Var(\Delta)}$ , and

$$\chi_1 = \{comp(\Delta_1 \cup (\bigcup_{i=2}^k \{ \bigwedge_{j=1}^{\text{card}(\Delta_i)} (-\text{holds}_{i,j} \vee \phi_{i,j}) \}))\}$$

is the *comp*-compilation of  $\Sigma$ .

This transformation basically consists in giving a name (under the form of a new literal  $\text{holds}_{i,j}$ ) to each assumption of  $\Delta$  and in storing the correspondance assumption/name with the certain beliefs before compiling them using *comp*. A similar transformation is given in [17, 27]. Interestingly, whatever the selection policy  $\mathcal{P} \in \{\mathcal{PO}, \mathcal{LO}, \mathcal{IP}, \mathcal{LE}\}$ ,  $\Sigma$  and  $comp(\Sigma)$  are equivalent on  $Var(\Sigma)$  w.r.t.  $\mathcal{P}$ : there exists a bijection  $f$  from  $\Sigma_{\mathcal{P}}$  to  $comp(\Sigma)_{\mathcal{P}}$  such that for every  $S \in \Sigma_{\mathcal{P}}$  and every formula  $\alpha$  from  $PROP_{Var(\Sigma)}$ ,  $S \models \alpha$  if and only if  $f(S) \models \alpha$ .

**Proposition 5.** The complexity of  $\vdash_{\mathcal{V}}^{\mathcal{P}}$  and of its restrictions to clause and literal inference for  $\mathcal{P} \in \{\mathcal{PO}, \mathcal{LO}, \mathcal{IP}, \mathcal{LE}\}$  from a *comp*-compiled SBB is as reported in Table 2.

Proposition 5 shows that considering compiled SBBs can actually make inference computationally easier, even if this is not the case for every policy. In this proposition, no specific assumption on the nature of the *comp* function is made. In order to obtain complexity lower bounds (hardness results) and some tractability results in the clausal case w.r.t. the  $\mathcal{IP}$  policy and the  $\mathcal{LE}$  policy, restricted

**Table 2.** Complexity of skeptical inference from *comp*-compiled SBBs (upper bounds).

$\mathcal{P}$	$\vdash_{\mathcal{V}}^{\mathcal{P}}$	CLAUSE / LITERAL $\vdash_{\mathcal{V}}^{\mathcal{P}}$
$\mathcal{PO}$	in coNP	in $\mathcal{P}$
$\mathcal{LO}$	in coNP	in $\mathcal{P}$
$\mathcal{IP}$	in coNP	in coNP
$\mathcal{LE}$	in $\Delta_2^{\mathcal{P}}$	in $\Delta_2^{\mathcal{P}}$

compiled SBBs must be considered. In the following, we focus on *comp*-compiled SBBs where *comp* is a compilation function which maps any propositional formula into  $\mathcal{PI}$  formula, a DNF formula or a HORN-CNF[ $\vee$ ] formula.

**Proposition 6.** The complexity of  $\vdash_{\mathcal{V}}^{\mathcal{IP}}$  and of its restrictions to clause and literal inference from *comp*-compiled SBBs is as reported in Table 3.

**Table 3.** Complexity of skeptical inference w.r.t.  $\mathcal{IP}$  from *comp*-compiled SBBs.

<i>comp</i>	$\vdash_{\mathcal{V}}^{\mathcal{IP}}$	CLAUSE / LITERAL $\vdash_{\mathcal{V}}^{\mathcal{IP}}$
$\mathcal{PI}$	coNP-complete	coNP-complete
DNF	coNP-complete	in $\mathcal{P}$
HORN-CNF[ $\vee$ ]	coNP-complete	coNP-complete

**Proposition 7.** The complexity of  $\vdash_{\mathcal{V}}^{\mathcal{LE}}$  and of its restrictions to clause and literal inference from *comp*-compiled SBBs is as reported in Table 4.

**Table 4.** Complexity of skeptical inference w.r.t.  $\mathcal{LE}$  from *comp*-compiled SBBs.

<i>comp</i>	$\vdash_{\mathcal{V}}^{\mathcal{LE}}$	CLAUSE / LITERAL $\vdash_{\mathcal{V}}^{\mathcal{LE}}$
$\mathcal{PI}$	$\Delta_2^{\mathcal{P}}$ -complete	$\Delta_2^{\mathcal{P}}$ -complete
DNF	coNP-complete	in $\mathcal{P}$
HORN-CNF[ $\vee$ ]	$\Delta_2^{\mathcal{P}}$ -complete	$\Delta_2^{\mathcal{P}}$ -complete

Tractability is only achieved for *comp*-compiled SBBs for which  $\Delta_1$  is a DNF formula and queries are restricted to CNF formulas. Intractability results w.r.t. both  $\vdash_{\mathcal{V}}^{\mathcal{IP}}$  and  $\vdash_{\mathcal{V}}^{\mathcal{LE}}$  still hold when  $\Delta_1$  is a consistent Krom formula, or when  $\Delta_1$  is a HORN-CNF formula. All the hardness results presented in Table 3 still hold in the specific case when the number  $k$  of strata under consideration satisfies  $k \geq 2$ . Similarly, LITERAL  $\vdash_{\mathcal{V}}^{\mathcal{LE}}$  remains coNP-hard in the restricted case when  $k \geq 2$ .

Interestingly, imposing some restrictions on the literals used to name assumptions allows to derive tractable restrictions for both CLAUSE  $\vdash_{\mathcal{V}}^{\mathcal{IP}}$  and CLAUSE  $\vdash_{\mathcal{V}}^{\mathcal{LE}}$  from the *comp*-compilation of an SBB where  $\Delta_1$  is a HORN-CNF[ $\vee$ ] formula. Indeed, we have:

**Proposition 8.** CLAUSE  $\vdash_{\mathcal{V}}^{\mathcal{IP}}$  and CLAUSE  $\vdash_{\mathcal{V}}^{\mathcal{LE}}$  from a *comp*-compiled SBB  $\Sigma = (\Delta_1, \dots, \Delta_k)$  where  $\Delta_1$  is a HORN-CNF[ $\vee$ ] formula and  $\bigcup_{i=2}^k \Delta_i$  contains only negative literals are in  $\mathcal{P}$ .

For other works on the compilation of belief bases, see e.g. [17, 6, 27, 7].

## 7 CONCLUSION

For the last two decades, KC has become an important topic in AI. Focusing on the propositional case, I have sketched in the previous

paragraphs the main research directions which have been followed for this period.

As explained in the introduction, these notes must be viewed as an introduction to KC, nothing more. In particular, there are many important pieces of work relevant to KC, yet not mentioned here. Among others: KC for closed-world reasoning and default reasoning [65, 3, 21], KC languages based on other formal settings, like CSPs, Bayesian networks, valued CSPs, description logics, etc. [2, 1, 25, 42, 39, 73, 19, 18, 38, 40], applications of KC to diagnosis [23, 45, 34, 66], to planning [41, 67, 8, 68].

Much work remains to be done. For instance:

- The conception of a refined compilability framework, obtained by imposing some computational restrictions on *comp* (in the current framework, *comp* can even be a non-recursive function).
- The study of the benefits which can be obtained by taking advantage of KC within many other AI problems (e.g., merging and other forms of paraconsistent inference). This evaluation can be done both at the problem level and at the instance level.
- The inclusions of additional languages, queries and transformations into the KC map.
- The decomposition of other AI problems into such queries and transformations.
- The development of KC maps for more expressive settings than propositional logic.
- The design of additional compilers and their evaluation on benchmarks.
- The successful exploitation of KC into other applications.

## ACKNOWLEDGEMENTS

The author would like to thank the ECAI'08 committees for their kind invitation. This work has been partly supported by the Région Nord/Pas-de-Calais.

## REFERENCES

- [1] J. Amilhastre, H. Fargier, and P. Marquis, 'Consistency restoration and explanations in dynamic CSPs application to configuration', *Artificial Intelligence*, **135**(1-2), 199–234, (2002).
- [2] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, 'Algebraic Decision Diagrams and their applications', in *Proc. of ICAD'93*, pp. 188–191, (1993).
- [3] R. Ben-Eliyahu and R. Dechter, 'Default reasoning using classical logic', *Artificial Intelligence*, **84**, 113–150, (1996).
- [4] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade, 'Inconsistency management and prioritized syntax-based entailment', in *Proc. of IJCAI'93*, pp. 640–645, (1993).
- [5] S. Benferhat, D. Dubois, and H. Prade, 'How to infer from inconsistent beliefs without revising', in *Proc. of IJCAI'95*, pp. 1449–1455, (1995).
- [6] S. Benferhat, S. Kaci, D. Le Berre, and M.-A. Williams, 'Weakening conflicting information for iterated revision and knowledge integration', in *Proc. of IJCAI'01*, pp. 109–115, (2001).
- [7] S. Benferhat, S. Yahi, and H. Drias, 'On the compilation of stratified belief bases under linear and possibilistic logic policies', in *Proc. of IJCAI'07*, pp. 2425–2430, (2007). (poster).
- [8] B. Bonet and H. Geffner, 'Heuristics for planning with penalties and rewards using compiled knowledge', in *Proc. of KR'06*, pp. 452–462, (2006).
- [9] Y. Boufkhad, 'Algorithms for propositional KB approximation', in *Proc. of AAAI'98*, pp. 280–285, Madison (WI), (1998).
- [10] Y. Boufkhad, E. Grégoire, P. Marquis, B. Mazure, and L. Sais, 'Tractable cover compilations', in *Proc. of IJCAI'97*, pp. 122–127, (1997).
- [11] R.E. Bryant, 'Graph-based algorithms for Boolean function manipulation', *IEEE Transactions on Computers*, **8**, 677–692, (C-35).
- [12] M. Cadoli and F.M. Donini, 'A survey on knowledge compilation', *AI Communications*, **10**(3–4), 137–150, (1998).
- [13] M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf, 'The size of a revised knowledge base', *Artificial Intelligence*, **115**(1), 25–64, (1999).
- [14] M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf, 'Space efficiency of propositional knowledge representation formalisms', *Journal of Artificial Intelligence Research*, **13**, 1–31, (2000).
- [15] M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf, 'Preprocessing of intractable problems', *Information and Computation*, **176**(2), 89–120, (2002).
- [16] M. Cadoli, F.M. Donini, and M. Schaerf, 'Is intractability of non-monotonic reasoning a real drawback?', *Artificial Intelligence*, **88**, 215–251, (1996).
- [17] C. Cayrol, M.-C. Lagasquie-Schiex, and Th. Schiex, 'Nonmonotonic reasoning: from complexity to algorithms', *Annals of Mathematics and Artificial Intelligence*, **22**(3–4), 207–236, (1998).
- [18] M. Chavira and A. Darwiche, 'Compiling Bayesian networks using variable elimination', in *Proc. of IJCAI'07*, pp. 2443–2449, (2007).
- [19] M. Chavira, A. Darwiche, and M. Jaeger, 'Compiling relational Bayesian networks for exact inference', *International Journal of Approximate Reasoning*, **42**(1–2), 4–20, (2006).
- [20] H. Chen, 'Parameterized compilability', in *Proc. of IJCAI'05*, pp. 412–417, (2005).
- [21] S. Coste-Marquis and P. Marquis, 'Knowledge compilation for closed world reasoning and circumscription', *Journal of Logic and Computation*, **11**(4), 579–607, (2001).
- [22] S. Coste-Marquis and P. Marquis, 'On stratified belief base compilation', *Annals of Mathematics and Artificial Intelligence*, **42**(4), 399–442, (2004).
- [23] A. Darwiche, 'Model-based diagnosis using structured system descriptions', *Journal of Artificial Intelligence Research*, **8**, 165–222, (1998).
- [24] A. Darwiche, 'Decomposable negation normal form', *Journal of the ACM*, **48**(4), 608–647, (2001).
- [25] A. Darwiche, 'A logical approach to factoring belief networks', in *Proc. of KR'02*, pp. 409–420, (2002).
- [26] A. Darwiche and P. Marquis, 'A knowledge compilation map', *Journal of Artificial Intelligence Research*, **17**, 229–264, (2002).
- [27] A. Darwiche and P. Marquis, 'Compiling propositional weighted bases', *Artificial Intelligence*, **157**(1-2), 81–113, (2004).
- [28] M. Davis, G. Logemann, and D. Loveland, 'A machine program for theorem-proving', *Journal of the ACM*, **5**(7), 394–397, (July 1962).
- [29] A. del Val, 'Tractable databases: How to make propositional unit resolution complete through compilation', in *Proc. of KR'94*, pp. 551–561, (1994).
- [30] A. del Val, 'An analysis of approximate knowledge compilation', in *Proc. of IJCAI'95*, pp. 830–836, (1995).
- [31] A. del Val, 'Approximate knowledge compilation: The first order case', in *Proc. of AAAI'96*, pp. 498–503, (1996).
- [32] A. del Val, 'A new method for consequence finding and compilation in restricted languages', in *Proc. of AAAI'99*, pp. 259–264, (1999).
- [33] A. del Val, 'First order LUB approximations: characterization and algorithms', *Artificial Intelligence*, **162**(1-2), 7–48, (2005).
- [34] P. Elliott and B.C. Williams, 'DNNF-based belief state estimation', in *Proc. of AAAI'06*, pp. 36–41, (2006).
- [35] H. Fargier and P. Marquis, 'On the use of partially ordered decision graphs in knowledge compilation and quantified Boolean formulae', in *Proc. of AAAI'06*, pp. 42–47, (2006).
- [36] H. Fargier and P. Marquis, 'Extending the knowledge compilation map: Closure principles', in *Proc. of ECAI'08*, (2008). (to appear).
- [37] H. Fargier and P. Marquis, 'Extending the knowledge compilation map: Krom, Horn, affine and beyond', in *Proc. of AAAI'08*, (2008). (to appear).
- [38] H. Fargier and P. Marquis, 'On valued negation normal form formulas', in *Proc. of IJCAI'07*, pp. 360–365, (2007). (poster).
- [39] H. Fargier and M.-C. Vilarem, 'Compiling CSPs into tree-driven automata for interactive solving', *Constraints*, **9**, 263–287, (2004).
- [40] Ulrich Furbach and Claudia Obermaier, 'Knowledge compilation for description logics', in *Proc. of KESE'07*, (2007).
- [41] H. Geffner, 'Planning graphs and knowledge compilation', in *Proc. of KR'04*, pp. 662–672, (2004).
- [42] F. Goasdoué and M.-C. Rousset, 'Compilation and approximation of conjunctive queries by concept descriptions', in *Proc. of ECAI'02*, pp. 267–271, (2002).
- [43] G. Gogic, H.A. Kautz, Ch.H. Papadimitriou, and B. Selman, 'The com-

- parative linguistics of knowledge representation', in *Proc. of IJCAI'95*, pp. 862–869, (1995).
- [44] L. Hai and S. Jigui, 'Knowledge compilation using the extension rule', *Journal of Automated Reasoning*, **32**, 93–102, (2004).
- [45] J. Huang and A. Darwiche, 'On compiling system models for faster and more scalable diagnosis', in *Proc. of AAAI'05*, pp. 300–306, (2005).
- [46] J. Huang and A. Darwiche, 'The language of search', *Journal of Artificial Intelligence Research*, **29**, 191–219, (2007).
- [47] R.M. Karp and R.J. Lipton, 'Some connections between non-uniform and uniform complexity classes', in *Proc. of STOC'80*, pp. 302–309, (1980).
- [48] H.A. Kautz and B. Selman, 'Forming concepts for fast inference', in *Proc. of AAAI'92*, pp. 786–793, San Jose (CA), (1992).
- [49] J. Lang, P. Liberatore, and P. Marquis, 'Propositional independence - formula-variable independence and forgetting', *Journal of Artificial Intelligence Research*, **18**, 391–443, (2003).
- [50] P. Liberatore, *Compilation of intractable problems and its application to artificial intelligence*, Ph.D. dissertation, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1998.
- [51] P. Liberatore, 'On the compilability of diagnosis, planning, reasoning about actions, belief revision, etc.', in *Proc. of KR'98*, pp. 144–155, (1998).
- [52] P. Liberatore, 'Compilability and compact representations of revision of Horn knowledge bases', *ACM Transactions on Computational Logic*, **1**(1), 131–161, (2000).
- [53] P. Liberatore, 'Monotonic reductions, representative equivalence, and compilation of intractable problems', *Journal of the ACM*, **48**(6), 1091–1125, (2001).
- [54] P. Liberatore, 'Complexity and compilability of diagnosis and recovery of graph-based systems', *International Journal of Intelligent Systems*, **20**(10), 1053–1076, (2005).
- [55] P. Liberatore and M. Schaerf, 'Compilability of propositional abduction', *ACM Transactions on Computational Logic*, **8**(1), 2, (2007).
- [56] F. Lin and R. Reiter, 'Forget it!', in *Proc. of the AAAI Fall Symposium on Relevance*, pp. 154–159, (1994).
- [57] P. Marquis, 'Knowledge compilation using theory prime implicates', in *Proc. of IJCAI'95*, pp. 837–843, (1995).
- [58] P. Marquis, *Consequence Finding Algorithms*, volume vol. 5 of "Algorithms for Defeasible and Uncertain Reasoning" of *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, chapter 2, 41–145, Kluwer Academic Publisher, 2000.
- [59] P. Marquis and S. Sadaoui, 'A new algorithm for computing theory prime implicates compilations', in *Proc. of AAAI'96*, pp. 504–509, (1996).
- [60] Ph. Mathieu and J.-P. Delahaye, 'The logical compilation of knowledge bases', in *Proc. of JELIA'90*, pp. 386–398, (1990).
- [61] Ph. Mathieu and J.-P. Delahaye, 'A kind of logical compilation for knowledge bases', *Theoretical Computer Science*, **131**(1), 197–218, (1994).
- [62] Y. Moses and M. Tennenholtz, 'Off-line reasoning for on-line efficiency: Knowledge bases', *Artificial Intelligence*, **83**, 229–239, (1996).
- [63] B. Nebel, *Belief Revision*, volume 3 of *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, chapter How hard is it to revise a belief base?, 77–145, Kluwer Academic, 1998.
- [64] B. Nebel, 'On the compilability and expressive power of propositional planning formalisms', *Journal of Artificial Intelligence Research*, **12**, 271–315, (2000).
- [65] A. Nerode, R.T. Ng, and V.S. Subrahmanian, 'Computing circumscriptive databases. I: Theory and algorithms', *Information and Computation*, **116**, 58–80, (1995).
- [66] B. O'Sullivan and G.M. Provan, 'Approximate compilation for embedded model-based reasoning', in *Proc. of AAAI'06*, pp. 894–899, (2006).
- [67] H. Palacios, B. Bonet, A. Darwiche, and H. Geffner, 'Pruning conformant plans by counting models on compiled d-DNNF representations', in *Proc. of ICAPS'05*, pp. 141–150, (2005).
- [68] H. Palacios and H. Geffner, 'Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes)', in *Proc. of AAAI'06*, pp. 900–905, (2006).
- [69] Ch. H. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994.
- [70] K. Pipatsrisawat and A. Darwiche, 'New compilation languages based on structured decomposability', in *Proc. of AAAI'08*, (2008). (to appear).
- [71] R. Reiter and J. de Kleer, 'Foundations of assumption-based truth maintenance systems: Preliminary report', in *Proc. of AAAI'87*, pp. 183–188, (1987).
- [72] O. Roussel and Ph. Mathieu, 'The achievement of knowledge bases by cycle search', *Information and Computation*, **162**(1-2), 43–58, (2000).
- [73] S. Sanner and D. A. McAllester, 'Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference.', in *Proc. of IJCAI'05*, pp. 1384–1390, (2005).
- [74] R. Schrag, 'Compilation for critically constrained knowledge bases', in *Proc. of AAAI'96*, pp. 510–515, (1996).
- [75] B. Selman and H.A. Kautz, 'Knowledge compilation using Horn approximations', in *Proc. of AAAI'91*, pp. 904–909, (1991).
- [76] B. Selman and H.A. Kautz, 'Knowledge compilation and theory approximation', *Journal of the ACM*, **43**, 193–224, (1996).
- [77] L. Simon and A. del Val, 'Efficient consequence finding', in *Proc. of IJCAI'01*, pp. 359–370, (2001).
- [78] S. Subbarayan, L. Bordeaux, and Y. Hamadi, 'Knowledge compilation properties of tree-of-BDDs', in *Proc. of AAAI'07*, pp. 502–507, (2007).
- [79] M. Wachter and R. Haenni, 'Propositional DAGs: A new graph-based language for representing Boolean functions', in *Proc. of KR'06*, pp. 277–285, (2006).
- [80] C.K. Yap, 'Some consequences of non-uniform conditions on uniform classes', *Theoretical Computer Science*, **26**, 287–300, (1983).