Compile!

Pierre Marquis
CRIL-CNRS, Université d'Artois, France
marquis@cril.univ-artois.fr

Abstract

This paper is concerned with *knowledge compilation* (KC), a family of approaches developed in AI for more than twenty years. Knowledge compilation consists in pre-processing some pieces of the available information in order to improve the computational efficiency (especially, the time complexity) of some tasks. In this paper, the focus is laid on three KC topics which gave rise to many works: the development of knowledge compilation techniques for the clausal entailment problem in propositional logic, the concept of compilability and the notion of knowledge compilation map. The three topics, as well as an overview of the main results from the literature, are presented. Some recent research lines are also discussed.

Introduction

Pioneered more than two decades ago, knowledge compilation (KC) (see (Cadoli and Donini 1998) for a survey) appears nowadays as a very active field in AI. In KC one is basically concerned with the computation of a recursive function f, which represents the task to be solved. f takes two arguments: a "fixed" part F and a "varying part" V. The objective is to improve the time needed to compute $f(F, V)$, the image of F, V by f, when V varies. To reach this goal, the fixed part F is pre-processed during an off-line phase: F is turned into a compiled form $C(F)$ using a compilation function C. In some situations, such a pre-processing is computationally useful: for some p the (cumulated) time needed to compute $C(F)$, $f(C(F), V_1)$, ..., $f(C(F), V_p)$ can be lower (sometimes by orders of magnitude) than the (cumulated, on-line) time needed to compute $f(F, V_1)$, ..., $f(F, V_p)$. It must be noted that the assumption that some pieces of information are not very often subject to change (a "fixed" part exists) is valid in many scenarios. For instance, in a computer-aided product configuration problem (which will be considered as a "running example"), the objective is to help the customer to find out a product (e.g., a car) by making some elementary choices (e.g., the number of seats, the engine, the color of the car). What makes this task not obvious is that, on the one hand, not all the combinations of elementary choices correspond to a feasible product (due to production or marketing constraints) and on the other hand, the product catalog is so large that it cannot be represented extensionally. Here, F may represent intensionally the feasible products and each V_i represents some user choices; F is independent of the V_i; among the tasks of interest, a basic one consists in determining whether the current choices V_i of the user are compatible with F, i.e., whether there exists a product which satisfies those choices (if the answer is negative, then the user must be aware of it and has to question some of her choices).

In KC, "knowledge" must be taken in a rather broad sense: F may represent any piece of information. In AI, F typically corresponds to beliefs or to preferences, but pieces of information of other types could be considered as well. Obviously, beliefs and preferences can be modeled in many different ways; most of the existing work focus on propositional statements (in classical logic) and utility functions, respectively. Unsurprisingly, due to the very nature of the input, the tasks to be achieved typically amount to combinations of elementary inference or optimization processes. For instance, in a configuration problem, one needs to be able to provide the user with the direct consequences of the choices she makes and to maintain the minimal and maximal costs of the products compatible with the current choices. In such applications where guaranteed response times are expected, knowledge compilation appears as very challenging. Indeed, offering some guarantee on the response time is a first-class requirement for many programs which must make on-line decisions under strong time constraints. Especially, this is the case for a number of Web-based applications (the user is typically not ready to wait a long time for getting the information she is looking for).

A standard scenario for which some KC approaches have been developed is the one where F is a database (or a knowledge base), each V_i is a query, and $f(F, V_i)$ is the answer to V_i given F. More specifically, one is often interested in the case when F is a (classical) propositional formula, V_i is a propositional clause (or a propositional CNF formula), and f corresponds to the clausal entailment problem **CE**, i.e., f is the characteristic function of the language $\{(F, V_i) \mid F \models V_i\}$. This problem occurs in many AI applications. Indeed, since $F \models V_i$ holds iff $F \wedge \neg V_i$ is consistent, determining whether the user choices ($\neg V_i$) are consistent with F reduces to the clausal entailment problem. In particular, such a task is of the utmost value for product

configuration: the models of F represents the feasible products and each time the user makes some choices one must be able to determine efficiently whether such choices can be extended to a feasible product. Because the clausal entailment problem is coNP-complete in propositional logic when F is a CNF formula, there is no available polynomial-time algorithm for solving it (i.e., for computing f) in this case, and it is conjectured that no such algorithm is possible (this would imply P = NP). As a consequence, in the general case there is no polynomial bound on the time required to compute $f(F, V)$ whatever the algorithm used. Especially, while modern SAT solvers prove efficient in many cases, taking account for the partial assignment corresponding to $\neg V$ may reduce drastically the set of solutions, so that $F \wedge \neg V$ has very few models and those models are hard to be found. Thus, no guaranteed response times can be ensured in the general case when $f(F, V)$ is computed as $\mathsf{sat}(F \wedge \neg V)$.

A fundamental question is the assessment of a KC method: "when does a KC approach C achieve its goal?". This is not an easy question. Indeed, the set of all possible V_i which will be considered during the on-line phase is typically unknown and often too large for being taken into account exhaustively. Hence one needs to select a subset of "significant" V_i and check whether the cumulated time needed to compute $C(F)$ and $f(C(F), V_i)$ for each "significant" V_i is lower than the cumulated time needed to compute $f(F, V_i)$ for each "significant" V_i. Several difficulties must be overcome: how to select the V_i used for assessing the method? Which algorithm should be considered as a base line for computing $f(F, V_i)$?

Instead of looking at the assessment problem at the instance level (i.e., for some given V_i), one can also consider it at the problem level (i.e., whatever V). This leads to the concept of *compilability*, see e.g., (Cadoli et al. 2002). Under this view, provided that f is a pseudo-Boolean function (corresponding to a decision problem encoded as usual as the formal language of its positive instances), compiling F using C is considered as valuable if the problem which consists in deciding whether $f(C(F), V) = 1$ for any admissible V is computationally easier than the problem which consists in deciding whether $f(F, V) = 1$ for any admissible V. C is required to be a polynomial-size function (i.e., for some fixed polynomial, the size of $C(F)$ is polynomial in the size of F). "Computationally easier" typically means that the decision problem when F is compiled using C belongs to a complexity class C located "below" in the polynomial hierarchy than the same problem when F has not been compiled. Of special interest is the case when C = P, i.e., the problem becomes tractable provided that the fixed part of the input is $C(F)$. In such a case, the problem is said to be compilable to P. It must be noted that the polynomial-size condition on C is a mandatory requirement for avoiding an unbiased evaluation. Nevertheless, evaluating a KC approach C at the problem level also has its cons: it can prove too coarse-grained to give 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 $C(F)$ is polynomial in the size of F.

Another important issue in knowledge compilation is the choice of the language L into which the fixed part F must be compiled, i.e., the co-domain of the compilation function C. Often the domain-dependent task f to be achieved is the combination of a number of elementary subtasks, each of them modeled as a specific function f_i. Each function corresponds either to a query (i.e., an operation which does not modify the input) or to a transformation (i.e. an operation which returns another L-representation). The choice of a target language L (which is reminiscent to a well-known issue in Computer Science: the choice of a data structure for a given abstract data type, made precise by the set of functions f_i) relies on several criteria, including the expressiveness of L (i.e., its ability to represent the available information) and more precisely its spatial efficiency (i.e., its ability to do so using little space), as well as its time efficiency (i.e., the ability to compute the functions f_i efficiently when the input is an L-representation). A *knowledge compilation map* (Darwiche and Marquis 2002) is a systematic, multi-criteria comparative analysis of languages which are candidates for the representation of $C(F)$. A first map for propositional formulae / Boolean functions has been developed and enriched from then.

The rest of the paper is organized as follows. First, I review some of the pioneering works on knowledge compilation, where the problem of interest is clausal entailment in propositional logic. Given its importance (as explained before), there is still some active research on the topic and some recent results based on the notion of empowering clause are sketched. Then the concept of compilability is formally defined and again a couple of properties presented. The next section is about the notion of knowledge compilation map. The concepts of queries and transformations, as well as those of expressiveness, spatial efficiency and time efficiency w.r.t. some queries and/or transformations are recalled. Some examples pertaining to the propositional case are provided. I also briefly describe some propositional languages which have been added recently to the propositional map. Finally, I sketch some hot off the press results about the compilation of valued CSPs (and similar representations), which can be used in AI for modeling various types of information, including probability distributions and utility functions. Of course, due to space reasons, some choices had to be made so that this paper is far from an exhaustive survey of the topic. Among other things, KC for closed-world reasoning and default reasoning, KC for modal logics/description logics, KC for lifted probabilistic inference, applications of KC to diagnosis, to planning, are all interesting topics which are not covered here.

KC for Clausal Entailment

One of the main applications of KC (at least from a "historical" perspective) is to enhance the efficiency of clausal

entailment **CE** in propositional logic. 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, C 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 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 with any propositional formula F one of the logically weakest HORN−CNF formula $f_l(F)$ implying F as well as one of the logically strongest HORN−CNF formula $f_u(F)$ implied by F (Selman and Kautz 1996). While $f_u(F)$ is unique up to logical equivalence, exponentially many non-equivalent $f_l(F)$ may exist. Once $f_l(F)$ and $f_u(F)$ are computed, they can be used to answer in polynomial time every clausal query V s.t. $f_l(F) \not\models V$ (1) or $f_u(F) \models V$ (2) (due to the transitivity of $\models$); the answer is "no" in case (1), "yes" in case (2), and Horn (clausal) queries can be answered exactly using $f_l(L)$. Such approaches have been extended to other languages (see e.g., (Simon and del Val 2001)).

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), which have been used for a long time for the KC purpose. Like the conjunctive formulae $F = F_1 \wedge \ldots \wedge F_n$ which are decomposable, i.e., the conjuncts do not share common variables, the prime implicates formulae (alias Blake formulae) satisfy a (first) *separability property* (the conjunctive one) stating that:

$F = F_1 \wedge \ldots \wedge F_n$ is conjunctively separable if and only if for any clause V, $F \models V$ if and only if there exists $i \in 1 \ldots n$ s.t. $F_i \models V$.

A similar (yet distinct) second separability property (the disjunctive one) is satisfied by all disjunctive formulae F:

$F = F_1 \vee \ldots \vee F_n$ is disjunctively separable, i.e., for any formula V, $F \models V$ if and only if for every $i \in 1 \ldots n$ s.t. $F_i \models V$.

Beyond the language PI of prime implicates formulae, those two separability properties underly many target languages for KC satisfying **CE**, especially the language DNNF (Darwiche 2001) of propositional circuits in "Decomposable Negation Normal Form" (and its subsets d−DNNF, $\text{OBDD}_<$, DNF, MODS). This can be explained by considering Shannon decompositions of formulae. Indeed, the Shannon decomposition of a formula F over a variable x is a formula

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

equivalent to F exhibiting several separable subformulae:

- $\neg x \wedge (F \mid \neg x)$ and $x \wedge (F \mid x)$ are disjunctively separable;
- $\neg x$ and $F \mid \neg x$ are conjunctively separable;
- x and $F \mid x$ are conjunctively separable.

Performing in a systematic way the Shannon decomposition of a formula F over the variables occurring in it leads to formulae from languages satisfying **CE**. Using or not a fixed decomposition ordering gives rise to distinct languages. Since the DPLL procedure (Davis, Logemann, and Loveland 1962) for the satisfiability problem SAT can be used to perform such decompositions, several target languages for KC can be characterized by the traces achieved by this search procedure (Huang and Darwiche 2007). Many KC approaches to clausal entailment actually exploit the separability properties (see e.g., (Marquis 1995; Schrag 1996; Boufkhad et al. 1997)).

Finally, other compilation techniques aim at exploiting the power of unit resolution so as to make clausal entailment tractable. Saturating a CNF formula F using unit resolution, i.e., computing the CNF formula obtained by removing the literal $\sim l$ in every clause of F whenever the unit clause l belongs to F, can be achieved in time linear in the size of F. When the empty clause is generated, a unit refutation from F exists. In order to make a CNF formula F unit-refutation complete (i.e., such that for every implicate $V = l_1 \vee \ldots \vee l_k$ of it, there is a unit refutation from $F \wedge \sim l_1 \wedge \ldots \wedge \sim l_k$), an approach consists in adding some clauses (typically some prime implicates of F) to it. Computing all the prime implicates is useless; especially it is enough to focus on those which can be derived using merge resolution (del Val 1994).

There has been recently a "revival" of the KC techniques for clausal entailment based on unit resolution, see e.g., (Bordeaux and Marques-Silva 2012; Bordeaux et al. 2012). This can be explained by the design for the past few years in the Constraint Programming community of some propagation-complete CNF encodings for several specific constraints. Basically, one wants to derive CNF encodings which are such that unit propagation is enough to derive every unit clause which is implied by the formula in any conjunctive context. In more formal terms, a CNF formula F is propagation-complete iff for every term $l_1 \wedge \ldots \wedge l_k$ and every literal l, if $F \wedge l_1 \wedge \ldots \wedge l_k \models l$, then there exists a unit resolution proof of l from $F \wedge l_1 \wedge \ldots \wedge l_k$. Clearly enough, if F is propagation-complete, then it is unit-refutation complete as well. In order to make F propagation-complete, it is enough to add empowering implicates to it: an implicate $V = l_1 \vee \ldots \vee l_k \vee l$ of F is empowering when $F \wedge \sim l_1 \wedge \ldots \wedge \sim l_k \models l$ but there is no unit resolution proof of l from $F \wedge \sim l_1 \wedge \ldots \wedge \sim l_k$. It has been proved that F has no empowering implicate (i.e., it is closed under empowerment) precisely when F is propagation-complete. Interestingly, the concept of empowering clause plays an important role for characterizing clause-learning SAT solvers (Pipatsrisawat and Darwiche 2011). Approaches to the compilation of CNF formulae by adding empowering clauses have been developed. Noticeably, there are families of CNF formulae which are closed under empowerment (hence propagation-complete) but also have exponentially many prime implicates (even when considering only those produced by merge resolution).

A Glimpse at Compilability

For all the compilation functions C considered in the previous section, there exist families of formulae F_s of size s such that $C(F_s)$ is of size exponential in s.[1] While the clausal entailment problem can be solved in polynomial time from $C(F_s)$, the computational benefits of KC is dubious in such a case since it is not guaranteed that the computational effort spent for compiling F_s can be balanced; indeed, due to the size of $C(F_s)$ it can be the case that the time needed to determine whether a clause V is a logical consequence of $C(F_s)$ exceeds the time needed to determine whether V is a logical consequence of F_s. Could a clever choice of C lead to overcome the problem? This question amounts to determining whether the decision problem corresponding to **CE** is *compilable to* P. As we will see, the answer to it is negative (under standard assumptions of complexity theory).

Roughly speaking, a decision problem is said to be compilable to a given complexity class C if it is in 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 preprocessing leads to a compiled form of polynomial size 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 the polynomial hierarchy PH) and the corresponding reductions (see the excellent pieces of work (Liberatore 2001; Cadoli et al. 2002)). This enables to classify many AI problems as compilable to a class C, or as not compilable to C (usually under standard assumptions of complexity theory - the fact that PH does not collapse).

Several families of classes can be considered as candidates to represent what "compilable to C" means. One of them is the family of compC classes:

Definition 1 *Let C be a complexity class closed under polynomial many-one reductions and admitting complete problems for such reductions. A language of pairs $\mathbf{L}_1$ belongs to compC if and only if there exists a polynomial-size function C [2] and a language of pairs $\mathbf{L}_2 \in \mathsf{C}$ such that $(F, V) \in \mathbf{L}_1$ if and only if $(C(F), V) \in \mathbf{L}_2$.*

Obviously enough, for every admissible complexity class C, we have the inclusion $\mathsf{C} \subseteq \mathsf{compC}$ (choose C as the identity function). Note that no requirement is imposed on the time complexity of C (the function C can even be non-recursive!). This leads to strong non-compilability results.

In order to prove the membership to compC of a problem it is enough to follow the definition (hence, exhibiting a polynomial-size compilation function C). Things are more complex to prove that a given problem does *not* belong to compC (under the standard assumptions of complexity theory).

A notion of comp-reduction suited to the compilability classes compC has been pointed out, and the existence of complete problems for such classes proved. However, many non-compilability results from the literature cannot be rephrased as compC-completeness results. For instance, it is unlikely that **CE** is $\mathsf{compcoNP}$-complete (it would make $\mathsf{P} = \mathsf{NP}$). In order to go further, one needs to consider the compilability classes $\mathsf{nu\text{-}compC}$ (Cadoli et al. 2002):

Definition 2 *Let C be a complexity class closed under polynomial many-one reductions and admitting complete problems for such reductions. A language of pairs $\mathbf{L}_1$ belongs to $\mathsf{nu\text{-}compC}$ if and only if there exists a binary polynomial-size function C and a language of pairs $\mathbf{L}_2 \in \mathsf{C}$ such that for all $\langle F, V\rangle \in \mathbf{L}_1$, we have:*

$$\langle F, V\rangle \in \mathbf{L}_1 \text{ if and only if } \langle C(F, |V|), V\rangle \in \mathbf{L}_2.$$

Here "nu" stands for "non-uniform", which indicates that the compiled form of F may also depend on the size of the varying part V. A notion of non-uniform comp-reduction suited to the compilability classes $\mathsf{nu\text{-}compC}$ has also been pointed out (it includes the notion of (uniform) comp-reduction), as well as the existence of complete problems for such classes. For instance, the clausal entailment problem is $\mathsf{nu\text{-}compcoNP}$-complete.

Inclusion of compilability classes compC, $\mathsf{nu\text{-}compC}$ similar to those holding in the polynomial hierarchy PH exist (see (Cadoli et al. 2002)). 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 (Cadoli et al. 2002)). For instance, if the clausal entailment problem **CE** is in $\mathsf{nu\text{-}compP}$, then the polynomial hierarchy collapses. Accordingly, in order to show that a problem is not in compC, it is enough to prove that it is $\mathsf{nu\text{-}compC'}$-hard, where $\mathsf{C'}$ is located higher than C in the polynomial hierarchy. Since complete problems for any $\mathsf{nu\text{-}compC}$ class can be easily derived from complete problems for the corresponding class C of the polynomial hierarchy, $\mathsf{nu\text{-}compC}$-complete problems appear as a key tool for proving non-compilability results.

Thus, the compilability of a number of AI problems, including diagnosis, planning, abduction, belief revision, belief update, closed-world reasoning, and paraconsistent inference from belief bases has been investigated from then.

The Notion of KC Map

Another important issue to be addressed in KC is the choice of a *target language* L, i.e., the representation language of compiled forms (L is the co-domain of C). 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 introductive section, many tasks can be decomposed into domain-independent basic queries and transformations f_i, so that one can focus on such queries and transformations instead of the tasks themselves. Stepping back to the configuration problem, beyond determining whether the user choices correspond to a feasible

[1] This is known for years for prime implicates; contrastingly, though a non-constructive proof was already known, the identification of a family of CNF formulae for which the number of clauses to be added for rendering it propagation-complete is recent (Babka et al. 2013).

[2] A function C is polynomial-size if and only if there exists a polynomial p such that $|C(F)|$ is bounded by $p(|F|)$ for every F in the domain of C.

product, one can be asked to count them, and when their number is small enough, the user can be interested in enumerating such products.

Generally speaking, the choice of a target language L for KC is based on several criteria including:

- the expressiveness and the spatial efficiency (succinctness) of L;
- the time efficiency of L for the queries of interest;
- the time efficiency of L for the transformations of interest.

Expressiveness is modeled by a pre-order over representation languages. It captures the ability of encoding information. Considering a language L which is not expressive enough for representing F may lead to a loss of information (the best we can do is then to approximate F in L, see the previous discussion about approximate compilation methods for the clausal entailment problem). Succinctness is a refinement of expressiveness which considers the representation sizes.

The Propositional Case

In the propositional case, expressiveness and succinctness are defined as follows:

Definition 3 *Let L_1 and L_2 be two propositional languages.*

- *L_1 is at least as expressive as L_2, denoted $L_1 \leq_e L_2$, iff for every formula $\alpha \in L_2$, there exists an equivalent formula $\beta \in L_1$;*
- *L_1 is at least as succinct as L_2, denoted $L_1 \leq_s L_2$, iff there exists a polynomial p such that for every formula $\alpha \in L_2$, there exists an equivalent formula $\beta \in L_1$ where $|\beta| \leq p(|\alpha|)$.*

Observe that the definition of succinctness does not require that there exists a function C computing β given α in *polynomial time*; it is only asked that a *polynomial-size* function C exists. The succinctness criterion is important because the time complexity of any f_i depends on the size of the input $C(F_i)$; switching from F to an exponentially larger $C(F)$ can be a way to obtain polynomial-time algorithms for some f_i but actual benefits will be hardly reached (as a matter of example, just consider the compilation of CNF formulae into the MODS language).

The KC map presented in (Darwiche and Marquis 2002) is suited to the case of classical propositional logic. Beyond clausal entailment **CE**, queries considered in it are tests for consistency **CO** (i.e., the SAT problem), validity **VA**, implicants **IM**, equivalence **EQ**, and sentential entailment **SE**. Model counting **MC** and model enumeration **ME** have also been considered. A number of transformations have also been taken into account in (Darwiche and Marquis 2002), especially conditioning **CD**, closures w.r.t. the n-ary connectives $\wedge\mathbf{C}$ (conjunction), $\vee\mathbf{C}$ (disjunction), the unary one $\neg\mathbf{C}$ (negation), and the corresponding bounded versions $\wedge\mathbf{BC}$ and $\vee\mathbf{BC}$. Forgetting **FO** (i.e., the elimination of existentially quantified variables) is another transformation which is important for many problems. Each query and each transformation corresponds to a specific function f_i of interest. Queries and transformations are also viewed as properties satisfied (or not) by representation languages: roughly speaking, L is said to satisfy a given query / transformation f_i when there exists a polynomial-time algorithm for computing f_i provided that the input is in L; L is said not to satisfy f_i if the existence of such an algorithm is impossible or if it would imply P = NP.

The KC map reported in (Darwiche and Marquis 2002) gathers the multi-criteria evaluation of a number of propositional languages. As to the time efficiency, one can find "positive results" in it. For instance, DNNF satisfies **CO**, **CE**, **ME**, **CD**, **FO**, $\vee\mathbf{C}$. "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**, $\wedge\mathbf{BC}$, $\neg\mathbf{C}$ unless P = NP. Importantly, the KC map also indicates how languages compare one another w.r.t. expressiveness/succinctness. For instance, while DNNF, d−DNNF, $\text{OBDD}_<$ and MODS are equally (and fully) expressive (i.e., every propositional formula has a representation in any of those languages), they are not equally succinct; indeed, we have the following strict succinctness ordering: DNNF $<_s$ d−DNNF $<_s$ $\text{OBDD}_<$ $<_s$ MODS.

Here, "positive" translatability results (i.e., showing that $L_1 \leq_s L_2$ for some given languages L_1 and L_2) can be obtained by exhibiting a polynomial-size function C; "negative" results (i.e., showing that $L_2 \not\leq_s L_1$) can be obtained using combinatorial arguments or non-compilability 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 formulae is not in compP unless PH collapses.

Based on the KC map, the choice of a target language for the KC purpose can be done by listing first the queries and transformations required by the application under consideration and by considering as second class criteria the expressiveness/succinctness ones. Other selection policies are possible.

It must be noted that many additional languages (and some missing results (Bova et al. 2014)) have been added to the propositional KC map for the past twelve years. The objective was to search for languages offering other trade-offs w.r.t. spatial complexity and time complexity than the existing ones. Among others, let us mention the language DDG of decomposable decision graphs (alias decision-DNNF) (Fargier and Marquis 2006; Beame et al. 2013), the language $\text{ToOBDD}_<$ of trees of $\text{OBDD}_<$ (Subbarayan, Bordeaux, and Hamadi 2007), the language SDD of sentential decision diagrams, which is a subset of d−DNNF but a superset of $\text{OBDD}_<$(Darwiche 2011), the language EADT of extended affine decision trees (Koriche et al. 2013), the languages of disjunctive closures (Fargier and Marquis 2014), the language Sym−DDG of symmetry-driven decomposable decision graphs (Bart et al. 2014), etc.

Beyond the Propositional Case

For many applications, the language of propositional logic is not expressive enough for the representation of the available information. Considering again the configuration problem, the user choices have costs, and the cost of a full configuration is typically the sum of the costs of the elementary choices. For such a problem, it is important to provide the

user with both the minimum price and the maximum price of a feasible product corresponding to her current choices. This calls for languages enabling the representation (and the compilation) of non-Boolean functions, for instance real-valued pseudo-Boolean functions (i.e., functions from $\mathbb{B}^n$ to $\mathbb{R}$) or more generally real-valued multivariate functions from $D_1 \times \ldots \times D_n$ to $\mathbb{R}$, where each D_i is a finite domain. Obviously, the family of such functions does not reduce to cost functions as considered in product configuration but encompasses many other functions, widely used in AI, like utility functions and probability distributions.

Clearly enough, many languages have been defined so far for the representation of such multivariate functions. Among them are the language of weighted propositional formulae, Bayes nets, GAI-nets, valued CSPs, etc. However none of them qualifies as a target language for KC because those languages do not offer polynomial-time algorithms for some basic queries like the consistency one (does there exist a variable assignment leading to a valuation different from the least one?). This calls for languages for representing compiled forms of such functions.

Interestingly, the languages of the propositional KC map can be exploited to this purpose. An approach consists in associating a weight (or more generally a valuation from the carrier of a valuation structure) with each value of a variable and to compute the image by f of an assignment as a combination of the valuations of its components. Because the variables of f are often pairwise dependent (the independence of two variables of f, when it holds, is typically conditional to the assignment of other variables), new variables must be introduced, and only the corresponding weights are significant (the weights of the initial variables take the values of the neutral element of the combination operator). Thus propositional weighted bases can be compiled by associating a new (selector) variable with each formula of the base so that if the variable is set to true, the formula must be set to true as well. The weight given to the selector is the one of the associated formula. The propositional formulae associating each selector with an input formula are hard (structural) constraints. Once their conjunction has been compiled into a DNNF representation, determining an optimal model (e.g., a minimal cost one when initial weights represent costs additively aggregated) is a tractable query and minimizing it (i.e., computing a DNNF representation of all the optimal models) is a tractable transformation (Darwiche and Marquis 2004). Bayes nets can also be compiled by considering first a CNF formula representing it; this CNF formula can then be compiled as a d−DNNF representation, from which an arithmetic circuit is extracted in linear time; this circuit represents the polynomial of the input network. New variables ("parameter variables") corresponding to each entry of each conditional probability table are introduced (Darwiche 2009).

It must be noted that several families of valued decision diagrams have also been considered for a while as compiled representations of real-valued multivariate functions given by valued CSPs. In such diagrams, weights are used as labels of the nodes or the arcs and the valuation associated with an assignment is obtained by aggregating the weights encountered in the corresponding path in the diagram. Different languages result depending on the nature of the weights and the way they are aggregated. Let us mention the algebraic decision diagrams (ADD) (Bahar et al. 1993), the edge-valued decision diagrams (EVBDD) (Lai and Sastry 1992; Lai, Pedram, and Vrudhula 1996; Amilhastre, Fargier, and Marquis 2002), the semiring-labeled decision diagrams (SLDD) (Wilson 2005), the affine algebraic decision diagrams (AADD) (Tafertshofer and Pedram 1997; Sanner and McAllester 2005). However, no systematic evaluation of them w.r.t. the criteria considered in the KC map is available. Some recent papers are first steps in this direction. Thus, in (Fargier, Marquis, and Niveau 2013), the key concepts underlying the KC map have been generalized recently to the non-propositional case. In (Fargier, Marquis, and Schmidt 2013), the relative succinctness of ADD, AADD, and of two SLDD languages (one with $+$ as aggregator, the other one with $\times$) has been established. In (Fargier et al. 2014), the time efficiency of several valued decision diagrams for a number of queries (including optimization, consistency, validity) and transformations (including combinations and variable eliminations w.r.t. several operators – min, max, $+$, $\times$ – and cuts) has been identified.

Some Recent Compilers

Interestingly, in the past few years, several new compilers have been developed and applied to a number of AI problems. Let us mention among others:

- SDD is a recent package for constructing, manipulating and optimizing sentential decision diagrams (SDD) (Darwiche 2011). SDD representations are generated in a bottom-up way from CNF or DNF formulae.
- EADT is a top-down compiler which outputs an EADT representation (extended affine decision tree) (Koriche et al. 2013) equivalent to the input CNF formula.
- VDD is a bottom-up compiler which is currently developed in the context of the BR4CP project, about product configuration and granted by the French National Research Agency. The VDD compiler aims at generating (ordered) valued decision diagrams from valued CSPs; the knowledge compilation languages which can be targeted are AADD, SLDD_+, $\text{SLDD}_\times$ and ADD.

Such compilers are efficient enough for enabling the compilation of many instances of significant sizes, corresponding to real problems. For instance, a weighted CSP representing the cost function associated with cars from the same family has been compiled using VDD as a SLDD_+ representation in less than 2 minutes, despite its size which is quite big (268 variables, with a maximal domain size of 324, and 2157 constraints). Similarly, SDD makes possible to compile probabilistic graphical models with hundred of variables and thousands of parameters in a couple of minutes (see (Choi, Kisa, and Darwiche 2013)), while with EADT the compilation of some CNF instances including thousands of Boolean variables and thousands of clauses proved feasible within a few seconds. Everyone can now take advantage of such compilers (they available on the Web) to determine whether it is suited to her purpose. Thus: *compile!*

References

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs application to configuration. *Artificial Intellligence* 135:199–234.

Babka, M.; Balyo, T.; Cepek, O.; Gurský, S.; Kucera, P.; and Vlcek, V. 2013. Complexity issues related to propagation completeness. *Artificial Intelligence* 203:19–34.

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic decision diagrams and their applications. In *Proc. of ICCAD'93*, 188–191.

Bart, A.; Koriche, F.; Lagniez, J.-M.; and Marquis, P. 2014. Symmetry-driven decision diagrams for knowledge compilation. In *Proc. of ECAI'14*, 51–56.

Beame, P.; Li, J.; Roy, S.; and Suciu, D. 2013. Lower bounds for exact model counting and applications in probabilistic databases. In *Proc. of UAI'13*, 52–61.

Bordeaux, L., and Marques-Silva, J. 2012. Knowledge compilation with empowerment. In *Proc. of SOFSEM'12*, 612–624.

Bordeaux, L.; Janota, M.; Marques-Silva, J.; and Marquis, P. 2012. On unit-refutation complete formulae with existentially quantified variables. In *Proc. of KR'12*, 75–84.

Boufkhad, Y.; Grégoire, E.; Marquis, P.; Mazure, B.; and Saïs, L. 1997. Tractable cover compilations. In *Proc. of IJCAI'97*, 122–127.

Bova, S.; Capelli, F.; Mengel, S.; and Slivovsky, F. 2014. Expander CNFs have Exponential DNNF Size. Technical Report arXiv:1411.1995, Cornell University Library.

Cadoli, M., and Donini, F. 1998. A survey on knowledge compilation. *AI Communications* 10(3–4):137–150.

Cadoli, M.; Donini, F.; Liberatore, P.; and Schaerf, M. 2002. Preprocessing of intractable problems. *Information and Computation* 176(2):89–120.

Choi, A.; Kisa, D.; and Darwiche, A. 2013. Compiling probabilistic graphical models using sentential decision diagrams. In *Proc. of ECSQARU'13*, 121–132.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Darwiche, A., and Marquis, P. 2004. Compiling propositional weighted bases. *Artificial Intelligence* 157(1-2):81–113.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proc. of IJCAI'11*, 819–826.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Journal of the ACM* 5(7):394–397.

del Val, A. 1994. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proc. of KR'94*, 551–561.

Fargier, H., and Marquis, P. 2006. On the use of partially ordered decision graphs in knowledge compilation and quantified Boolean formulae. In *Proc. of AAAI'06*, 42–47.

Fargier, H., and Marquis, P. 2014. Disjunctive closures for knowledge compilation. *Artificial Intelligence* 216:129–162.

Fargier, H.; Marquis, P.; Niveau, A.; and Schmidt, N. 2014. A knowledge compilation map for ordered real-valued decision diagrams. In *Proc. of AAAI'14*, 1049–1055.

Fargier, H.; Marquis, P.; and Niveau, A. 2013. Towards a knowledge compilation map for heterogeneous representation languages. In *Proc. of IJCAI'13*, 877–883.

Fargier, H.; Marquis, P.; and Schmidt, N. 2013. Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues. In *Proc. of IJCAI'13*, 884–890.

Huang, J., and Darwiche, A. 2007. The language of search. *Journal of Artificial Intelligence Research* 29:191–219.

Koriche, F.; Lagniez, J.-M.; Marquis, P.; and Thomas, S. 2013. Knowledge compilation for model counting: Affine decision trees. In *Proc. of IJCAI'13*, 947–953.

Lai, Y.-T., and Sastry, S. 1992. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proc. of DAC'92*, 608–613.

Lai, Y.-T.; Pedram, M.; and Vrudhula, S. B. K. 1996. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers* 45(2):247–255.

Liberatore, P. 2001. Monotonic reductions, representative equivalence, and compilation of intractable problems. *Journal of the ACM* 48(6):1091–1125.

Marquis, P. 1995. Knowledge compilation using theory prime implicates. In *Proc. of IJCAI'95*, 837–843.

Pipatsrisawat, K., and Darwiche, A. 2011. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence* 175(2):512–525.

Sanner, S., and McAllester, D. A. 2005. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *Proc. of IJCAI'05*, 1384–1390.

Schrag, R. 1996. Compilation for critically constrained knowledge bases. In *Proc. of AAAI'96*, 510–515.

Selman, B., and Kautz, H. 1996. Knowledge compilation and theory approximation. *Journal of the ACM* 43:193–224.

Simon, L., and del Val, A. 2001. Efficient consequence finding. In *Proc. of IJCAI'01*, 359–370.

Subbarayan, S.; Bordeaux, L.; and Hamadi, Y. 2007. Knowledge compilation properties of tree-of-BDDs. In *Proc. of AAAI'07*, 502–507.

Tafertshofer, P., and Pedram, M. 1997. Factored edge-valued binary decision diagrams. *Formal Methods in System Design* 10(2-3).

Wilson, N. 2005. Decision diagrams for the computation of semiring valuations. In *Proc. of IJCAI'05*, 331–336.