Knowledge Compilation with Empowerment

Lucas Bordeaux[1] and Joao Marques-Silva[1,2,3]

[1] Microsoft Research, Cambridge, UK
[2] University College Dublin, Ireland
[3] IST/INESC-ID, Lisbon, Portugal

Abstract. When we encode constraints as Boolean formulas, a natural question is whether the encoding ensures a ”propagation completeness” property: is the basic unit propagation mechanism able to deduce all the literals that are logically valid? We consider the problem of automatically finding encodings with this property. Our goal is to compile a naïve definition of a constraint into a good, propagation-complete encoding. Well-known Knowledge Compilation techniques from AI can be used for this purpose, but the constraints for which they can produce a polynomial size encoding are few. We show that the notion of *empowerment* recently introduced in the SAT literature allows producing encodings that are shorter than with previous techniques, sometimes exponentially.

1 Introduction

Modeling problems with constraints is as much an art as it is a science. Very often the same relations between variables can be encoded in a wide range of possible ways. Although semantically equivalent these various encodings may present dramatic differences in terms of complexity: some encodings are ”well-posed” in some sense, which guarantees a tractable reasoning on the constraints, while some others are formulated in a way that hinders the deduction mechanisms of constraint solvers.

The Question we investigate is how to automatize the search for such ”well-posed” encodings: given a naïve definition of a constraint in propositional logic, how do we produce an equivalent, but ”well-posed”, encoding of the constraint? Both of the words *naïve* and *well-posed* require clarification. The specific notion of ”well-posed” we focus on is defined precisely later in the paper by the property of *propagation completeness*, which states that the simple unit propagation rule that is at the core of SAT solvers effectively deduces all the literals that are logically entailed. The notion of ”naïve” encoding is unavoidably more subjective: by this we essentially mean a concise encoding that logically captures the semantics of the constraint, but ignores any of the redundancies or ”modeling tricks” that experts will usually add for performance reasons.

Producing propagation-complete encodings is an important question on which significant prior work can be found. In particular the Knowledge Compilation literature proposes a rich set of techniques for the preprocessing of logical formulas into a ”compiled” form that allows certain types of reasoning, including literal and clausal entailment, to be done in polynomial time [16, 8]. In parallel, recent Constraint Programming (CP) literature proposes propagation-complete SAT encodings of specific constraints [2, 6, 15, 11, 4, 9, 4]. Bacchus [2] makes the connection between the two areas and was the first to explicitly suggest the use of Knowledge Compilation techniques to automatically obtain propagation-complete encodings of complex constraints.

Our Contribution in this paper is to relate clausal Knowledge compilation to the notion of *empowerment*, recently introduced by [1, 14]. We show that generating empowering redundant clauses is a key rewriting technique that can be used to generate "useful" redundant constraints that ultimately produce a propagation-complete formula. This observation is simple and, in retrospect, natural, but we prove that restricting the clause generation to empowering clauses can in some cases reduce exponentially the size of compiled formulas, compared to previous CNF knowledge compilation techniques.

Overview of the Paper. We start by reviewing the notation and required material in Section 2. Section 3 presents the notion of empowerment and studies the question of how to find empowering clauses. Section 4 introduces and studies the technique of compilation using empowerment, which is compared to prior CNF compilation techniques in Section 5. Finally, Section 6 outlines a number of research directions.

2 Preliminaries

This section overviews the needed preliminary material and notation.

Resolution. The paper will mostly consider logical formulas in CNF (Conjunctive Normal Form). Recall that these formulas are conjunctions of clauses, each of which is a disjunction of literals (variable or its negation). We write $\varphi \models c$ to indicate that clause c is a logical consequence of φ, i.e. all models of φ also satisfy c. We say that c is an *implicate* of φ. We denote by $\perp$ the empty clause; by $\mathit{vars}(\varphi)$ the set of variables that have an occurrence in a formula φ; by $|c|$ the length (number of literals) of a clause c.

Propositional *resolution* is the well-known deduction rule that deduces from two clauses of the form $A \vee x$ and $\neg x \vee B$ the consequence $A \vee B$. Additionally to this rule we implicitly use the rewriting rules of *simplification* ($x \vee x$ and $x \wedge x$ rewrite to x) and *exchange* ($x \vee y$, resp. $x \wedge y$, rewrite to $y \vee x$, resp. $y \wedge x$), which mean that conjunctions and disjunctions are effectively treated as sets. *Unit resolution*, aka unit propagation, is the special case where A or B is empty. We use the symbol $\vdash$ for deduction using resolution, with subscripts corresponding to restricted cases of resolution: in particular given a CNF formula φ we write $\varphi \vdash_1 l$ if the clause c can be deduced from φ using unit propagation, as used in SAT solvers.

Propagation-Completeness is defined as follows:

Definition 1 (Propagation-Completeness). *A formula φ is propagation-complete if for any set of literals $\{l_1, \cdots, l_k\}$ any literal d that is logically entailed can be obtained by unit propagation, i.e.,*

$$\textit{if} \quad \varphi \wedge l_1 \wedge \cdots \wedge l_k \models d \quad \textit{then} \quad \varphi \wedge l_1 \wedge \cdots \wedge l_k \vdash_1 d$$

In particular, if $\varphi \wedge l_1 \wedge \cdots \wedge l_k$ is inconsistent, and φ is propagation-complete, unit propagation deduces $\perp$. Propagation completeness has been implicitly considered in recent CP papers [10, 2, 6, 15, 11, 4] because of its connection to *Domain Consistency* (aka *Generalized Arc-Consistency*): when we encode in SAT a constraint over some

Finite-Domain variables, if the encoding of the variables is a natural encoding with one Boolean variable per value in the variable's domain (as opposed to "logarithmic encodings" where b Booleans encode 2^b possible values), and if the encoding of the constraint is propagation-complete, then unit propagation on the SAT encoding effectively finds the same implications as Domain Consistency.

Knowledge Compilation. We can now define more formally the problem we are considering throughout the paper:

Problem 1. Given a CNF formula φ, produce a "compiled formula" that is equivalent to φ and propagation-complete.

This question has been extensively studied in the *Knowledge Compilation* literature; more precisely what is studied is usually a variant called *clausal entailment* (see, e.g. [8]): given a clause $c \equiv (l_1 \vee \cdots \vee l_k)$, do we have $\varphi \models c$? It is easy to see that if a formula is propagation-complete then clausal entailment on the formula can be done in polynomial time as we can simply check whether $\varphi \wedge \neg l_1 \wedge \cdots \wedge \neg l_k \vdash_1 \bot$. Conversely if a polynomial-time algorithm (not necessarily based on unit propagation) exists for clausal entailment then we can check efficiently whether $\varphi \wedge l_1 \wedge \cdots \wedge l_k \models d$ for any literal d by checking whether $(\neg l_1 \vee \cdots \vee \neg l_k \vee d)$ is entailed. Clausal entailment and propagation-completeness are therefore essentially equivalent, but tractable clausal entailment can resort to any type of polynomial-time algorithm, while propagation-completeness specifically focusses on unit propagation.

In general, if the formula φ to be compiled is arbitrary, the compilation process hits fundamental complexity limits: clausal entailment, like other intractable problems, is "non-compilable" in general, unless *NP* $\subseteq$ *P/poly* [7]; hence reaching propagation-completeness requires in the worst case an exponentially large encoding. There are nevertheless many specific constraints whose satisfiability and unit implication problems are polynomial-time solvable, and for many of them (but *not all*, as shown in [5]!), concise propagation-complete CNF encodings have been proposed. [6, 15, 11, 4, 9, 4]. Problem 1 asks how we automatically find such encodings.

3 Empowering Implicates

In this section we review the notion of *empowering clause* and relate it to knowledge compilation. We next address the question: how do we compute empowering clauses for a formula?

3.1 Empowerment

It is well-known that adding logically redundant constraints (in SAT: implicates) to a problem can *in some cases* improve propagation but that not all redundant clauses are useful. Consider for instance $\varphi = \{(\neg x \vee y), (\neg y \vee z)\}$. The clause $(\neg x \vee z)$ is an implicate, however this clause does not add anything that really benefits propagation: from x we can deduce z and from $\neg z$ we can deduce $\neg x$, with or without this clause. The property that this clause is missing is *empowerment* [14]. This notion has been introduced in a different setting, but we argue that it is the *exact* characterization of "useful" clause. Implicates such as $(\neg x \vee z)$ that are useless are said to be *absorbed* by the CNF φ.

Definition 2 (Empowerment [14]; Absorption [1]). *Let φ be a CNF formula, and $c = (l_1 \vee \cdots \vee l_k)$ be an implicate of φ. The clause c is empowering*[4] *w.r.t. φ if one of its literals l_i, called* empowered literal*, is such that:*

$$\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\vdash_1 \quad l_i.$$

c called absorbed *by φ if it has no empowered literal.*

Our first observation is that empowerment is intimately related to compilation:

Definition 3 (Closure under Empowerment; Completion). *A formula ψ is said to be* closed under empowerment *if it absorbs any implicate. Given a formula φ, let ψ be a set of implicates of φ; if the formula $\varphi \cup \psi$ is closed under empowerment, then we say that it is a* completion *of φ.*

Proposition 1. *A formula is closed under empowerment iff it is propagation-complete.*

Given a CNF formula φ of size s (size here means the sum of clause lengths), and a candidate clause $c = (l_1 \vee \cdots \vee l_k)$, there are two contexts in which it may be useful to check whether c is an empowering implicate. If we do not know whether c is an implicate, then checking whether it is is coNP-complete. In some other cases, c may be *known to be an implicate*, for instance if it is obtained by application of steps of propositional resolution. Checking whether it is empowering is in this case easier: we simply have to check whether any of the l_is $(i \in 1..k)$ is obtained by propagation when we assert the conjunction $\bigwedge_{j \in 1..k, j \neq i} \neg l_j$. This can be done in time $\mathcal{O}(k \cdot s)$ since it is sufficient to run (and undo) k propagations.

3.2 Finding Empowering Implicates using QBF

Empowerment was introduced in the context of SAT solvers, and it was noted that learnt clauses in DPLL solvers are in fact empowering [14, 1]. However the literature has not, to our knowledge, proposed any *complete* method for finding empowering clauses. Such a method should fail to return an empowering implicate only when the formula is proved to be propagation-complete. We propose such a method based on an encoding to Quantified Boolean Formulae (QBF). The generated Quantified Boolean Formula asks whether there exists a clause that is a valid implicate and that is empowering. This QBF encoding is described below.

A set of variables X is assumed, with $|X| = n$, $X = \{x_1, \ldots, x_n\}$. Literals are represented as x_i^p, where $x_i \in X$, $p \in \{0, 1\}$, $x_i \equiv x_i^1$ and $\neg x_i \equiv x_i^0$. Essentially, p

[4] Reference [14] uses the term 1-empowering, but we drop the 1- prefix for simplicitiy. Also we use in fact the original formulation from a AAAI conference paper that precedes [14]: for technical reasons [14] adds the extra condition $\varphi \wedge \bigwedge_{j \neq i} \neg l_j \not\vdash_1 \bot$, which we do not need.

denotes the truth value of x_i that satisfies the literal associated with x_i^p. Furthermore, let $T(x_i^p)$ denote the clauses satisfied by x_i when assigned value $p \in \{0, 1\}$.

Consider a CNF formula φ and a clause c. Define $\varphi^a = \varphi \cup \{\neg l_c : l_c \in c\}$ and $\varphi^c = \varphi^a \setminus \{\neg l\}$, for a given literal $l \in c$. A clause c is 1-empowering for φ if there exists $l \in c$ such that: (i) $\varphi \models c$; (ii) $\varphi^c \nvdash_1 \perp$; and (iii) $\varphi^c \nvdash_1 l$.

The identification of a 1-empowering clause c consists of the following main steps: (i) select the literals of clause c, among all possible sets of literals; and (ii) validate the conditions of Definition 2 for clause c. The configuration of clause c is done through a set of auxiliary variables $S = \{s_i^p \,|\, x_i \in X \wedge p \in \{0, 1\}\}$. Clause c is defined as follows: $c = \{l_1^0, l_1^1, l_2^0, l_2^1, \ldots, l_n^0, l_n^1\}$, where $l_i^p \leftrightarrow x_i^p \wedge s_i^p$. Moreover, $(\neg s_i^0 \vee \neg s_i^1)$ holds for $i = 1, \ldots, n$. A complete assignment to the variables in set S decides which literals actualy integrate clause c.

Given the definition of set S, the model outlined above can be refined as follows:

$$\exists S.(\varphi \models c) \wedge (\varphi^c \nvdash_1 \perp) \wedge (\varphi^c \nvdash_1 l) \tag{1}$$

which captures the conditions of Definition 2. In the remainder of this section, the complete QBF is derived by specifying the following predicates (each associated with one of the conditions of 1-empowering clause):

$$\exists S.\text{Unsat}(\varphi^a) \wedge \text{ProperUPConfig}(\varphi^c) \wedge \bigvee_{l \in c} \text{NonUPImplied}(\varphi^c, l) \tag{2}$$

where Unsat(φ^a) holds if φ^a is unsatisfiable, ProperUPConfig(φ^c) holds if unit propagation (UP) on φ^c is non-inconsistent, and NonUPImplied(φ^c, l) holds if l is not implied by unit propagation. Predicate Unsat(φ^a) is represented by $\forall X.\neg\varphi^a(X)$. The other two predicates require modeling the proper outcomes of unit propagation (UP) as a propositional formula.

The selection of the set of variables to be declared assigned by unit propagation is achieved through a few sets of auxiliary variables. The first set U, is defined as follows: $U = \{u_i^p \,|\, x_i \in X \wedge p \in \{0, 1\}\}$, where u_i^p is true iff $x_i \in X$ takes value $p \in \{0, 1\}$ by unit propagation. Clearly, $(\neg u_i^0 \vee \neg u_i^1)$ holds for all $1 \leq i \leq n$. Also, if $u_i^0 = u_i^1 = 0$ then x_i is unassigned by unit propagation. Moreover, $u_{i,d}^p$ denotes whether x_i takes value p in no more than d unit propagation steps. The consistent assignments to $u_{i,d}^p$ are defined recursively as follows: (i) $u_{i,0}^p = 1$ iff (x_i^p) is a unit clause; and (ii) for $d > 0$, $u_{i,d}^p = 1$ if x_i^p was already set to 1 at earlier propagation stages than d, or if there exists a clause $c_t \in \varphi^c$ with all literals but x_i^p assigned value 0 in no more than $d - 1$ unit propagation steps. Formally,

$$\begin{array}{ll} u_{i,0}^p = 1 & \text{if } (x_i^p) \in \varphi^c \\ u_{i,0}^p = 0 & \text{if } (x_i^p) \notin \varphi^c \\ u_{i,d}^p = u_{i,d-1}^p \vee \bigvee_{c_t \in T(x_i^p)} \bigwedge_{\substack{x_j^q \in c_t \\ x_j^q \neq x_i^p}} u_{j,d-1}^{1-q} & \text{for } d > 0 \end{array} \tag{3}$$

For simplicity of notation, in the rest of the section we simply denote by u_i^p the variables $u_{i,n}^p$, that represent the state of each x_i at the end of propagation, i.e. when $d = n$ (last ”propagation level”).

Let $\varphi^{u,d}(u_i^p)$ denote the set of clauses from (3). Then, the predicate ProperUP(u_i^p) is given by the conjunction of $\varphi^{u,d}(u_i^p)$, $(\neg u_i^0 \vee \neg u_i^1)$, $(u_{i,d-1}^p \rightarrow u_{i,d}^p)$, with $1 \leq d \leq n$, and $(u_i^p \leftrightarrow u_{i,n}^p)$.

For a given clause c_t, the predicate SAT(c_t) is given by $(\vee_{x_i^p \in c_t} u_i^p)$. After consistent unit propagation, all clauses *must* either be satisfied or non-unit. Define v_i to be true iff x_i is unassigned, i.e. $v_i \leftrightarrow (\neg u_i^0 \wedge \neg u_i^1)$. Then the NonUnit$(c_t)$ predicate is defined by $(\sum_{x_i^p \in c_t} v_i \geq 2)$.

The predicate ProperUPConfig(φ^c, U) can be defined as follows:

$$\bigwedge_{\substack{1 \leq i \leq n \\ p \in \{0,1\}}} \text{ProperUP}(u_i^p) \wedge \bigwedge_{c_t \in \varphi^c} (\text{SAT}(c_t) \vee \text{NonUnit}(c_t)) \tag{4}$$

The existence of a proper UP configuration is then given by: $\exists U$.ProperUPConfig(φ^c, U). Finally, assuming literal l corresponds to x_k^p, predicate NonUPImplied(φ^c, x_k^p, U) is defined as follows: ProperUPConfig$(\varphi^c, U) \rightarrow \neg u_k^p$, i.e. for any non-inconsistent unit propagation x_k^p remains unassigned. Thus, the condition that x_k is not implied, is given by $\forall U$.NonUPImplied(φ^c, l, U).

Proposition 2. *Given a formula φ, the question:* is there a clause c that is an empowering implicate of φ? *is polynomial-time reducible to a QBF formula with quantifier alternation $\exists\forall$.*

Proof. See QBF derivation above.

We also note that the QBF encoding can easily be tuned to express the existence of empowering clauses *of bounded length*: this amounts to constraining the length of the desired clause in the encoding.

4 Compilation by Iterative Empowerment

The most natural approach to knowledge compilation using empowering clauses is to iteratively add empowering clauses to the formula until it ultimately becomes propagation-complete. We call this approach *compilation by iterative empowerment*. Here we study this approach and focus in particular on a disciplined approach where empowering clauses are introduced by increasing length.

4.1 Compilation Sequences

Knowledge compilation by empowering implicate generation is highly dependent on the order in which empowering clauses are generated. The notion of *compilation sequence* captures this ordering:

Definition 4. *Let φ be a CNF formula. A* compilation sequence *is a sequence of clauses $[\varphi; c_1; \cdots; c_k]$ such that:*

- *Each clause c_i for $1 \leq i \leq k$ is an implicate of φ that is empowering wr.t. the partially compiled formula $\varphi \wedge c_1 \wedge \cdots \wedge c_{i-1}$;*

– *$\varphi \wedge c_1 \wedge \cdots \wedge c_k$ is ultimately propagation-complete.*

Example 1. Consider the formula: $\varphi \equiv \{(a \vee b), (\neg a \vee x), (\neg a \vee y), (\neg b \vee x), (\neg b \vee y), (\neg x \vee y), (\neg y \vee x)\}$. There are two possible compilation sequences for this formula: $[\varphi; (x)]$ and $[\varphi; (y)]$. In other words we may start by adding the empowering implicate (x), in which case (y) becomes deducible by unit propagation and therefore is not an empowering clause; or we may start by adding (y) in which case (x) is not empowering.

4.2 Clause Deprecation

Being empowering w.r.t. $\varphi \wedge c_1 \wedge \cdots \wedge c_{i-1}$ does not, in general, guarantee that c_i will remain empowering w.r.t. to the fully compiled formula. One issue is that as we generate empowering clauses sequentially, some clauses created at an earlier stage may become non-empowering later as more clauses are added. We call this *clause deprecation*:

Definition 5 (Clause Deprecation). *In a compilation sequence $[\varphi; c_1; \cdots ; c_k]$ we say that a clause c_i is* deprecated *by the addition of clause c_j $(j > i)$ if c_i is empowering w.r.t $\varphi \wedge \bigwedge_{h \in 1..j-1, h \neq i} c_h$ and non-empowering w.r.t. $\varphi \wedge \bigwedge_{h \in 1..j, h \neq i} c_h$.*

Example 2. Consider the formula: $\varphi \equiv \{(a \vee b), (\neg a \vee x), (\neg a \vee y), (\neg b \vee x), (\neg b \vee y), (\neg x \vee y)\}$. The two possible compilation sequences are: $[\varphi; (y); (x)]$ and $[\varphi; (x)]$. In the first sequence, (y) is empowering w.r.t. φ but is not empowering w.r.t. $\varphi \wedge (x)$, i.e. becomes deprecated by the addition of clause (x).

In extreme examples, some poorly selected sequences can ultimately contain exponentially many deprecated clauses, as shown in the following example.

Example 3. Consider the following formulas parameterized by a size m:

$$\bigwedge_{p} \left(y \vee \bigvee_{h} x_{ph} \right) \quad \wedge \quad \bigwedge_{h} \bigwedge_{p} \bigwedge_{p' > p} (y \vee \neg x_{ph} \vee \neg x_{p'h})$$

where p ranges over $1..m$ and h ranges over $1..m-1$. (These formulas are a variant of the well-known Pigeon-Hole Principle (PHP) formulas encoding $y \vee \mathit{PHP}_m$.) It is clear that for this formula we can generate exponentially many clauses, and in particular many empowering ones, whereas the unit clause (y) is indeed the only meaningful implicate. Specifically, a possible compilation sequence starts by φ, then adds all clauses of the form $\left(y \vee \bigvee_{p \in 1..m-1} \neg x_{p,\delta(p)}\right)$, for all bijections δ from $1 \cdots m-1$ onto itself (i.e. permutations); then adds (y). All clauses are empowering at the time where they are added. Yet adding the final clause (y) deprecates all the previous clauses.

Example 3 shows that generating a short (here, unit) empowering clause can sometimes prevent the creation of many more empowering clauses. This suggests a strategy of *length-increasing* iterative empowerment, detailed next.

```
**function** *length_increasing_empower*($\varphi$):
    **let** $\psi := \emptyset$
    **for** $L$ **from** $1$ **to** *width*$(\varphi)$
        **while** there exists a clause $c$ of length $L$ that is empowering w.r.t. $\varphi \cup \psi$
            $\psi := \psi \cup \{c\}$
        % minimization code optionally goes here
    **return** $\varphi \cup \psi$

**function** *minimize*($\varphi$):
    **let** $\psi := \varphi$
    **foreach** clause $c$ **in** $\psi$        % arbitrary order
        **if** $c$ is absorbed by $\psi \setminus \{c\}$
            $\psi := \psi \setminus \{c\}$
    **return** $\psi$
```

Fig. 1. Algorithms: *length_increasing_empower* takes a CNF φ and returns a completion of it; *minimize* takes a propagation-complete CNF φ and returns a subset of it that is equivalent, propagation-complete, and minimal.

4.3 Length-Increasing Iterative Empowerment

We now consider what happens if we generate clauses *by increasing length*: we first saturate the formula under empowering clauses of length 1; only then do we consider length 2; and so forth. This is shown as Algorithm *length_increasing_empower* in Fig. 1. The algorithm is similar to the simple width-increasing algorithm resolution proposed in e.g. [3], but (1) only generates clauses that are empowering, and (2) exhaustively checks that no implicate of length L exists before incrementing L. The latter test can be done by a width-bounded QBF encoding as suggested in the previous section. This algorithm is non-deterministic in that there are many possible choices in the selection of the empowering clause at any stage. The sequences of implicates it generates can be characterized as follows:

Definition 6 (Length-Increasing Compilation Sequence). *A compilation sequence* $[\varphi; c_1; \cdots; c_k]$ *is* length-increasing *if for every* $i \in 1..k$ *we have that all implicates of length strictly less than* $|c_i|$ *are absorbed by* $\varphi \wedge c_1 \wedge \cdots \wedge c_{i-1}$.

A key property of compilation by length-increasing iterative empowerment is that it limits the effects of deprecation, in the following sense:

Proposition 3. *In a length-increasing sequence, a clause of length* b *never deprecates a clause of length* $a < b$.

4.4 Minimality

It may be desirable in some cases to compute propagation-complete formulas that are *minimal*, where removing any clause would cause the formula not to be propagation-complete anymore.

Definition 7 (Minimal propagation-complete formula). *A propagation-complete CNF formula* $\{c_1 \cdots c_m\}$ *is* minimal *if no* c_i *is absorbed by the set of remaining clauses, i.e.* $\{c_j \; : \; j \neq i\}$.

Example 2 shows that length-increasing iterative empowerment does not necessarily lead to formulas that are minimal: deprecation can happen *between generated implicates of the same length*. Minimizing a propagation-complete formula can be done by simply checking one by one, in some arbitrary order, whether any clause is absorbed by the rest of the formula, as shown in Algorithm *minimize* of Fig. 1. If a clause c is verified to be empowering during the execution of the algorithm, it is clear that it will remain empowering at the end of the execution, where more clauses have been removed; therefore considering each clause once is enough. Minimizing a formula of length s (counted in sum of clause lengths) takes time $\mathcal{O}(s^2)$ since we need to do/undo one propagation for each literal of every clause.

To construct a minimal propagation-complete formula using the length-increasing compilation approach, it is possible to interleave the generation of empowering clauses of every length with minimization steps. In Algorithm *length_increasing_empower* of Fig. 1 the minimization code can be added in the commented area. Because of Proposition 3, it is sufficient, once the generation of empowering implicates of a certain length L has completed, to verify the empowerment of clauses of length L, i.e. the **foreach** loop of Algorithm *minimize* can be restricted to clauses whose length is L.

5 Iterative Empowerment versus Prime Implicate Saturation

We now study compilation by iterative empowerment and compare it against prior implicate-based approaches to knowledge compilation. We focus for most of our results on the length-increasing compilation scheme, but do not assume minimality. Our main point is that it provides a strictly more "succinct" compilation language than prior compilation methods by prime implicates, where succinctness is defined as in [8].

5.1 Previous Compilation Schemes

A well-known way to obtain a propagation-complete formula is to saturate it by prime implicates, as was proposed by, e.g. [16, 13] and suggested for the encoding of constraints by [2]. An implicate of a formula φ is a clause c that is a valid consequence, i.e. $\varphi \models c$. An implicate is *prime* if it is not subsumed by another implicate, i.e., there is no implicate c' that contains a strict subset of the literals of c. We denote by $\mathit{prime}(\varphi)$ the set of prime implicates of a formula φ. (This set is uniquely defined.)

It is known that prime implicate generation can generate clauses that are useless for propagation-completeness; for instance from the formula $\varphi = (\neg x \vee y), (\neg y \vee z)$ the absorbed clause $(\neg x \vee z)$ is a prime implicate. Heuristics have been proposed to restrict the number of absorbed clauses, in particular based on the notion of *merge* resolution. When resolving two clauses $A \vee x$ and $\neg x \vee B$, the resolution step is called *non-merge* if $\mathit{vars}(A) \cap \mathit{vars}(B) = \emptyset$, and *merge* otherwise. Several optimizations to prime implicate generation have been proposed in [16]; the central idea is to avoid adding to the formula some implicates that are generated by non-merge resolution.

We compare iterative empowerment to this approach. Later approaches to prime implicate generation have been proposed, for instance [12], but these approaches depart from explicit CNF generation, while it is our aim to produce CNF encodings amenable to SAT solving. ([12] uses, specifically, a compact clause representation with BDDs).

5.2 Iterative Empowerment versus Prime Implicates

We first note that length-increasing compilation sequences can never generate more clauses than prime implicate generation, as they only contain prime implicates. We then show that compilation by saturation under empowering clauses can in some cases be exponentially more compact than approaches based on prime implicate generation.

Proposition 4. *All implicates generated in a length-increasing compilation sequence are prime.*

Proof. Consider a sequence $[\varphi; c_1; \cdots ; c_k]$, and clauses c_i and c_j. We assume that c_i subsumes c_j and show a contradition. Since the literals of c_i are a strict subset of those of c_j, we have $|c_i| < |c_j|$ and $i < j$, i.e. clause c_j is generated after c_i in the sequence. But c_j is not empowering w.r.t. the formula that already includes c_i: whenever c_j becomes unit, c_i either also becomes unit, or becomes inconsistent; in both cases no useful new literal can be inferred from c_j.

A class of formulas that exhibit an exponential separation between prime implicate saturation and iterative empowerment is the so-called EVEN formulas, introduced next. (Comments on the right-hand side of the formula explain the meaning of each block of clauses in this formula.) These are CNF encodings of the formula $x_1 \oplus \cdots \oplus x_n = 0$, true when the number of 1s is even.

Definition 8 (EVEN Formulas). *We denote by* EVEN_n *the following formula over the sets of variables* $X = \{x_1 \cdots x_n\}$ *and* $X = \{y_1 \cdots y_n\}$*:*

$$\begin{array}{ll}
(\neg x_1 \vee y_1) \wedge (\neg y_1 \vee x_1) & \text{"}y_1 = x_1\text{"} \\
\wedge \bigwedge_{i \in 2..n} \begin{pmatrix} (\neg y_i \vee y_{i-1} \vee x_i) \wedge \\ (\neg y_i \vee \neg y_{i-1} \vee \neg x_i) \wedge \\ (y_i \vee \neg y_{i-1} \vee x_i) \wedge \\ (y_i \vee y_{i-1} \vee \neg x_i) \end{pmatrix} & \text{"}y_i = y_{i-1} \oplus x_i\text{"} \\
\wedge (y_n) & \text{"output gate } y_n \text{ is true"}
\end{array}$$

Proposition 5. *The* EVEN_n *formulas are closed under empowerment yet have exponentially many prime implicates.*

Proof. We first note that the constraint hyper-graph of these formulas is Berge-acyclic. It is well-known that for acyclic constraint networks achieving arc consistency for each constraint of the hyper-graph is enough to achieve arc-consistency for the whole network. It follows that the formula is propagation-complete; In other words any implicate is absorbed. Now there exist an exponential number of prime implicates: (1) any clause over the variables $\{x_1 \cdots x_n\}$ that has an odd number of negative literals is an implicate. (2) these implicates are prime: if we remove any of their literals we obtain an invalid clause. (3) there are 2^{n-1} such clauses.

5.3 Iterative Empowerment versus Merge Resolution

We next consider the optimizations to prime implicate generation based on merge resolution [16]. The goal of these methods was to find a subset of the prime implicates that remains propagation-complete. However these previous approaches did not formulate the notion of empowerment, and the question is to compare them with iterative empowerment. We show that in some cases, those compilation schemes, that discard clauses that are produced by *non-merge* resolution steps, can still generate exponentially many non-empowering clauses. This is exhibited by the following class of formulas.

Definition 9 (MERGE-EVEN formulas). *We denote by* MERGE-EVEN$_n$ *the variant of* EVEN *in which every clause receives an additional positive literal f, such that $f \notin X \cup Y$; i.e.* MERGE-EVEN$_n$ *is the set of clauses* $\{(f \vee c) \ : \ c \in \text{EVEN}_n\}$.

The set of solutions of MERGE-EVEN$_n$ is exactly the union of: (1) all assignments where f is true (with any value assigned elsewhere); (2) all assignments where f is false and EVEN$_n$ holds.

Proposition 6. *The set of prime implicates of* MERGE-EVEN$_n$ *is:* $\{(f \vee A) \ : \ A \in$ *prime(*EVEN$_n$*)*$\}$

Proposition 7. *The formula* EVEN$_n$ *is closed under empowerment, but has exponentially many prime implicates; furthermore all of these implicates are produced by merge-resolution.*

Proof. Assume the existence of a non-empowering prime implicate of MERGE-EVEN. It is a clause of the form $(f \vee A \vee l)$, not subsumed by any clause of MERGE-EVEN, and where literal l is not obtained from MERGE-EVEN by unit propagation when f is false and all literals in A are false. When f is set to false the formula simplifies to EVEN, therefore it is also the case that l is not obtained from EVEN when all literals in A are false. This implies that $(A \vee l)$ is an empowering clause of EVEN not subsumed by any clause in it, which contradicts Proposition 5.

We now show that all the resolvents applied in the compilation are Merge. All clauses of EVEN include a positive occurrence of f. Any resolution on two such clauses is a merge resolution operation and produces in a clause that also includes a positive occurrence of f. Therefore all clauses in any resolution proof will include only clauses with positive occurrence of f, and make us of merge resolution exclusively.

Note also that in formula MERGE-EVEN the merge literal is always f. This means that the optimization of algorithm *FPI*$_1$ of [16] do not apply and that this algorithm also generates exponentially many absorbed implicates.

6 Conclusion and Perspectives

Clausal (CNF) formalisms played an important role in early Knowledge Compilation work, but more recent work has favoured non-clausal formalisms that are in many respects more expressive [8]. In our view clausal Knowledge Compilation remains important because of its connections to propagation-complete encodings, and to our initial Question (formalized as Problem 1). Our main findings in this paper are the following:

– The notion of *empowerment* of [1, 14] sheds a new light on clausal Knowledge Compilation: several heuristics had previously been proposed to limit the ”useless” redundant constraints generated by classical prime implicate methods; but they did not, to our knowledge, explicitly define ”useless” — it is now clear that empowerment is *the desired property* of implicates used in compilation.
– In particular we showed that length-increasing empowerment is a knowledge compilation scheme that has many appealing features in terms of limited deprecation and easier minimization, connection to treewidth, and comparison with other prime implicate generation schemes.

Acknowledgement. This work is partially supported by SFI PI grant BEACON (09/ IN.1/I2618), by FCT through grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/-123051/2010), and by INESC-ID multiannual funding from the PIDDAC program funds.

References

1. Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. J. of Artif. Intel. Research (JAIR) 40, 353–373 (2011), preliminary version in SAT 2009
2. Bacchus, F.: GAC via unit propagation. In: Princ. and Practice of Constraint Programming (CP). pp. 133–147 (2007)
3. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow - resolution made simple. J. of the ACM 48(2), 149–169 (2001)
4. Bessiere, C., Katsirelos, G., Narodytska, N., Quimper, C.G., Walsh, T.: Decompositions of all different, global cardinality and related constraints. In: Int. Joint. Conf. on Artif. Intel. (IJCAI). pp. 419–424 (2009)
5. Bessiere, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit complexity and decompositions of global constraints. In: Int. Joint. Conf. on Artif. Intel. (IJCAI). pp. 412–418 (2009)
6. Brand, S., Narodytska, N., Quimper, C.G., Stuckey, P.J., Walsh, T.: Encodings of the sequence constraint. In: Princ. and Practice of Constraint Programming (CP) (2007)
7. Cadoli, M., Donini, F., Liberatore, P., Schaerf, M.: Preprocessing of intractable problems. Information and Computation 176, 89–120 (2002)
8. Darwiche, D., Marquis, P.: A knowledge compilation map. J. of Artif. Intel. Research (JAIR) 17, 229–264 (2002)
9. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Princ. and Practice of Constraint Programming (CP). pp. 352–366 (2009)
10. Gent, I.P.: Arc consistency in SAT. In: Euro. Conf. on Artif. Intel. (ECAI). pp. 121–125 (2002)
11. Huang, J.: Universal Booleanization of constraint models. In: Princ. and Practice of Constraint Programming (CP). pp. 144–158 (2008)
12. Marquis, P., Sadaoui, S.: A new algorithm for computing theory prime implicates compilations. In: Conf. on Artif. Intel. (AAAI). pp. 504–509 (1996)
13. Marquis, P.: Knowledge compilation using theory prime implicates. In: IJCAI. pp. 837–845 (1995)
14. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artif. Intel. 175(2), 512–525 (2011), preliminary works in AAAI 2008 (notion of *empowerment*) and CP 2009.
15. Quimper, C.G., Walsh, T.: Decompositions of grammar constraints. In: Conf. on Artif. Intel. (AAAI). pp. 1567–1570 (2008)
16. del Val, A.: Tractable databases: How to make propositional unit resolution complete through compilation. In: Knowledge Representation and Reasoning (KR). pp. 551–561 (1994)