Knowledge Compilation for Product Configuration

Carsten Sinz[1]

Abstract. In this paper we address the application of knowledge compilation techniques to product configuration problems. We argument that both the process of generating valid configurations, as well as validation of product configuration knowledge bases, can potentially be accelerated by compiling the instance independent part of the knowledge base. Besides giving transformations of both tasks into logical entailment problems, we give a short summary on knowledge compilation techniques, and present a new algorithm for computing unit-resolution complete knowledge bases.

1 Introduction

Configuration of complex products is a computation intensive task. In most formalisms proposed in the literature [5, 10, 11], generating a consistent configuration can be intractable in the worst case, and is at best an NP-hard problem. Moreover, in a typical setting, huge series of configuration runs have to be performed for the same kind of product, but different customer demands. The closely related task of checking a product configuration knowledge-base for consistency [7] exhibits similar characteristics. Here a large number of validation properties have to be checked for a given knowledge-base.

These conditions make it attractive to investigate the application of knowledge compilation techniques (see Cadoli and Donini's article for a survey on knowledge compilation [2]). For a set of common problem instances, knowledge compilation separates the computational task into an instance dependent and an instance independent part. The latter can be solved in advance during a preprocessing step, which can potentially lead to a speedup in the overall run time.

A significant advantage of pre-compiled knowledge-bases is that in the lucky case of successful compilation into a knowledge base of reasonable size, short runtimes can be guaranteed for all individual configuration processes.

2 Formalisms for Product Configuration

In this paper, we consider two formalisms for product configuration. We use them as representatives for demonstrating applicability of knowledge compilation for both generating valid configurations and checking consistency of knowledge bases.

The first formalism is a slight variant of the logical theory of configuration presented by Felfernig *et al.* [5], which complies with Mittal and Frayman's component-port representation for configuration knowledge [11]. The second is a simplified version of the formalism used for the validation of DaimlerChrysler's engineering and production configuration system [7].

[1] Symbolic Computation Group, WSI for Computer Science, University of Tübingen, Sand 13, 72076 Tübingen, Germany

In Felfernig's system, a configuration problem consists of a domain description $\mathcal{D}$ and a system requirements specification $\mathcal{S}$, from which a consistent (*valid*) configuration c has to be constructed[2]. The domain description is a set of predicate logic sentences expressing compatibility constraints on the product's parts, and additional axioms describing and restricting the constraints language; the system requirements specification states customer demands on the desired product, again in the form of a set of predicate logic sentences. Then, a conjunction c of ground literals is a solution to the configuration problem $\langle \mathcal{D}, \mathcal{S} \rangle$, or a *valid* configuration, when it is logically consistent with the domain description and the requirements specification, i.e., $\mathcal{D} \cup \mathcal{S} \cup \{c\} \not\models \bot$. For our purpose we consider the propositional variant of this formalism. We assume a finite universe, or a universe with a finite number of equivalence classes with respect to the domain description and the requirements specification. Now the propositional case can be obtained from the first-order case by replacing all sentences of $\mathcal{D}$ and $\mathcal{S}$ by a conjunction of their ground instances, similar to a Herbrand expansion.

The second formalism we consider consists of a set of propositional constraint rules that make up a knowledge-base describing valid products [7]. The semantics of the whole knowledge-base can be interpreted as a propositional formula $\mathcal{B}$ whose models are the valid configurations. To check its consistency, we generate a set D of validation properties d_i and test whether or not the knowledge-base satisfies them. Therefore, we have to determine whether $\mathcal{B} \models d_i$ for all $d_i \in D$.

3 Logical Entailment and Knowledge Compilation

Problem instances of both formalisms can be formulated as *propositional entailment* problems, where the question is to find the deducible consequences a of a theory $\mathcal{T}$ (a set of propositional sentences). For the purpose of knowledge compilation, we partition the theory $\mathcal{T}$ into a constant part $\mathcal{T}_c$ and a varying part $\mathcal{T}_v$. The constant part $\mathcal{T}_c$ is then replaced by an equivalent, but computationally preferable, compiled theory $\mathcal{T}_c'$, and the varying part of the theory is moved to the consequence by means of the deduction theorem. Thus, the general entailment problem

$$\mathcal{T}_c \cup \mathcal{T}_v \models a$$

is restated in the compiled theory as

$$\mathcal{T}_c' \models a \vee \bigvee_{s \in \mathcal{T}_v} \neg s \quad . \tag{1}$$

[2] Ferfernig *et al.* [5] distinguish between consistent and valid configurations, where valid configurations have to fulfill additional completeness axioms. In this article we always refer to the augmented version with completeness axioms added when talking about valid or consistent configurations.

This transformation especially offers advantages when (a) the constant part of the theory is large compared to the variable part of the theory and the consequence to be checked, (b) the compilation of theory $\mathcal{T}_c$ into $\mathcal{T}_c'$ is efficient, and (c) there is a large number of entailment checks to be performed.

In transforming the first formalism, the fixed part $\mathcal{T}_c$ is the domain description $\mathcal{D}$, and the varying part is the systems requirement specification $\mathcal{S}$. The task now is to find a valid configuration consistent with $\mathcal{D}$ and $\mathcal{S}$ by a series of entailment checks. When we consider the inverted requirements specification $I = \bigvee_{s \in \mathcal{S}} \neg s$ being represented in conjunctive normal form (CNF), i.e. $I = i_1 \wedge \cdots \wedge i_k$, we can perform k entailment checks $\mathcal{D}' \models i_j$ for $1 \leq j \leq k$ within the compiled theory $\mathcal{D}'$ to find configurations conforming with the domain description: For each i_j with $\mathcal{D}' \not\models i_j$ the conjunction of literals $\neg i_j = l_1 \wedge \cdots \wedge l_m$ is a minimal valid configuration. This can be verified by observing that the configuration $c = \neg i_j = l_1 \wedge \cdots \wedge l_m$ is consistent with $\mathcal{D} \cup \mathcal{S}$. As $\mathcal{D}' \not\models i_j$, and $\mathcal{D}'$ is logically equivalent to $\mathcal{D}$, we have $\mathcal{D} \cup \{\neg i_j\} \not\models \bot$, hence a fortiori $\mathcal{D} \cup \{\neg i_1 \vee \cdots \vee \neg i_k\} \not\models \bot$. Now as S is equivalent to $\neg I = \neg i_1 \vee \cdots \vee \neg i_k$ and $c = \neg i_j$ for some j, we obtain $\mathcal{D} \cup \mathcal{S} \cup \{c\} \not\models \bot$.

Further non-minimal valid configurations can be generated using the following scheme: for each literal l not occurring in i_j we test whether $\mathcal{D}' \not\models i_j \vee \neg l$. If this is the case, we can safely add l to the configuration without violating validity.

In summary, to treat the transformation of the first formalism, we set $a = \bot$ in Formula 1, and consider a special representation (DNF) of the variable part of the theory T_v.

For transformation of the second formalism (checking validation properties) we assume the validation properties d_i to be in conjunctive normal form, i.e. $d_i = d_{i,1} \wedge \cdots \wedge d_{i,k}$. Then, after setting the constant part $\mathcal{T}_c$ of the theory to $\mathcal{T}_c = \{\mathcal{B}\}$, and noting that there is no variable part, i.e. $\mathcal{T}_v = \emptyset$, the test for property d_i decomposes into k entailment checks $\mathcal{T}_c' \models d_{i,j}$ in the compiled theory $\mathcal{T}_c'$.

In both formalisms we only have a restricted form of entailment checks, namely only tests $\mathcal{T}' \models a$, where a is a clause. Therefore we restrict our attention to propositional clausal entailment in the following. Knowledge compilation aims at generating theories for which the entailment problem is tractable—decidable in polynomial time—whereas the general propositional clausal entailment problem is coNP-complete [2].

4 Knowledge Compilation Techniques

Knowledge compilation and concequence finding are active areas of research [2, 9, 13]. The methods proposed in the literature are usually separated into two main categories: *approximate* and *exact* compilations. Approximate compilation mostly appears in the form of theory approximation, where a theory $\mathcal{T}$ is approximated by a computationally more tractable theory $\mathcal{T}'$. Selman and Kautz [13] use two approximating Horn theories, one approximating from above ($\mathcal{T} \models \mathcal{T}_{\mathrm{ub}}$) and the other from below ($\mathcal{T}_{\mathrm{lb}} \models \mathcal{T}$).[3] To decide entailment for a clause c, algorithm THEORY-APPROX, as shown in Figure 1, is employed. Entailment for Horn theories, i.e. for $\mathcal{T}_{\mathrm{ub}}$ and $\mathcal{T}_{\mathrm{lb}}$, can be decided in linear time, so algorithm THEORY-APPROX is supposed to decide many cases efficiently: the better the approximation, the more cases are computationally tractable. However, computation of good Horn approximations can be quite hard (computation of best approximations is NP-hard). Selman and Kautz [13] present different algorithms for their computation.

```
ALGORITHM THEORY-APPROX
INPUT:    $\mathcal{T}_{\mathrm{lb}}, \mathcal{T}, \mathcal{T}_{\mathrm{ub}}, c$ with $\mathcal{T}_{\mathrm{lb}} \models \mathcal{T} \models \mathcal{T}_{\mathrm{ub}}$
OUTPUT:   $true$ if $\mathcal{T} \models c$, $false$ otherwise
BEGIN
    IF $\mathcal{T}_{\mathrm{ub}} \models c$ THEN return $true$
    ELSE IF $\mathcal{T}_{\mathrm{lb}} \not\models c$ THEN return $false$
    ELSE return $\mathcal{T} \models c$
END
```

Figure 1. Theory Approximation Algorithm.

Exact compilation methods, as opposed to approximations, try to find a theory $\mathcal{T}'$ that is equivalent to $\mathcal{T}$, for which the entailment problem is tractable, i.e., decidable in polynomial time. The predominant method for computing such compilations is by generating prime implicants or prime implicates of the original theory.

In the following, we consider a compilation by the computation of prime implicates. The source theory then usually also is in conjunctive normal form (CNF)—a common incidence in practice. An *implicate c* of a theory $\mathcal{T}$ is a non-trivial clause (without complementary literals) such that $\mathcal{T} \models c$; moreover, c is a *prime implicate* if no proper sub-clause[4] of c is also an implicate of $\mathcal{T}$.

Computing the set $\mathrm{PI}(\mathcal{T})$ of all prime implicates of a theory $\mathcal{T}$ yields a theory $\mathcal{T}'$ that is equivalent to $\mathcal{T}$ and has the following important property: a clause c is a consequence of $\mathcal{T}$, i.e. $\mathcal{T} \models c$, iff there is a clause $p \in \mathrm{PI}(\mathcal{T})$ that is a sub-clause of c. Thus, using $\mathcal{T}' = \mathrm{PI}(\mathcal{T})$ as a compiled version of the theory $\mathcal{T}$, clausal entailment for a clause a can be decided in linear time in the size of $\mathcal{T}'$ and the query a.

Different algorithms have been proposed to compute the set of prime implicates of a theory [3, 12, 15]. However, the number of prime implicates may be exponential in the size of the theory $\mathcal{T}$, and therefore different strategies have been developed to compute more compact exact compilations. Among the extensions are computations of prime implicates from no-merge resolvents [4], theory prime implicates [8], and tractable cover compilations [1]. In the following, we will describe del Val's work [4] in more detail.

The prime implicate computation from no-merge resolvents is sometimes also referred to as unit-resolution (UR) complete compilation, and the idea is to delete those prime implicates from $\mathrm{PI}(\mathcal{T})$ that can be derived by a UR refutation proof from the remaining theory. As UR refutations can be computed in linear time, this means a shift from precompiled knowledge to deduction by a calculus that is still tractable. In the following we denote UR derivations by $\vdash_{\mathrm{u}}$, and use the convention that for a clause $c = l_1 \vee \cdots \vee l_k$ the notation $\bar{c}$ stands for the set of units $\{\neg l_1, \ldots, \neg l_k\}$ obtained by negating c. Our goal now is to compute a set $\mathcal{T}^*$ of clauses that is equivalent to $\mathcal{T}$, and for which

$$\mathcal{T}^* \cup \bar{c} \vdash_{\mathrm{u}} \bot$$

holds for all clauses c with $\mathcal{T} \models c$. Then all consequences of $\mathcal{T}$ can be derived by UR refutations from $\mathcal{T}^*$. Of course, setting $\mathcal{T}^* = \mathrm{PI}(\mathcal{T})$ would be a solution, but this is often not practical and does not

[3] The bounds are defined in terms of models: the lower bound has fewer, the upper bound more models than the given theory.

[4] Clause c is a (proper) sub-clause of d if the set of literals of c is a (proper) subset of the literals of d.

deliver the best, i.e. minimal, theory. Del Val [4] suggests different candidates for theory $\mathcal{T}^*$.

We now want to derive a precise characterization of an optimal $\mathcal{T}^*$, expressed by means of a fixed-point equation. Therefore, we define $\mathcal{T}^*_{\mathrm{opt}}$, an optimal solution for UR complete compilation, as a smallest (regarding set inclusion) solution $\hat{\mathcal{T}}$ of the formula

$$\forall c \in \mathrm{PI}(\mathcal{T})\big(c \in \hat{\mathcal{T}} \iff \bar{c} \cup \hat{\mathcal{T}} \setminus \{c\} \not\vdash_{\mathrm{u}} \bot\big) \ . \qquad (2)$$

Then a clause c from the set of prime implicates of theory $\mathcal{T}$ is in the solution theory $\mathcal{T}^*$ iff it cannot be derived by a UR refutation from $\mathcal{T}^*$ without c.

5 An Alternative Algorithm to Compute UR Complete Knowledge-Bases

Based on Formula 2 we now give an algorithm for UR complete knowledge compilation. Our algorithm, as well as all of del Val's algorithms for computation of UR complete compilations, is based on prime implicate generation. This has the advantage that advances in prime implicate computation can also improve knowledge compilation, but suffers from the drawback that in the worst case an exponential number of clauses has to be generated, even if the final result does not show this exponential blow-up. Our alternative algorithm, as shown in Figure 2, computes a different, in some cases smaller, compiled knowledge-base than del Vals algorithms.

```
(1)  ALGORITHM UR-COMPILATION
(2)  INPUT: $\mathcal{T}$
(3)  OUTPUT: $\mathcal{T}^*$, which is a solution to Formula 2
(4)  BEGIN
(5)    $\mathcal{T}^*$ := PI($\mathcal{T}$);
(6)    FOR EACH $c \in \mathcal{T}^*$ DO
(7)      IF $\bar{c} \cup \mathcal{T}^* \setminus \{c\} \vdash_{\mathrm{u}} \bot$ THEN
(8)        $\mathcal{T}^*$ := $\mathcal{T}^* \setminus \{c\}$;
(9)        compute minimal $D(c) \subseteq \mathcal{T}^*$ with
(10)         $\bar{c} \cup D(c) \vdash_{\mathrm{u}} \bot$;
(11)       FOR EACH $c' \in \mathrm{PI}(\mathcal{T}) \setminus \mathcal{T}^*$ with $c \in D(c')$ DO
(12)         IF $\bar{c}' \cup \mathcal{T}^* \vdash_{\mathrm{u}} \bot$ THEN
(13)           update $D(c')$
(14)         ELSE
(15)           $\mathcal{T}^*$ := $\mathcal{T}^* \cup \{c'\}$;
(16) END
```

Figure 2. Algorithm for UR Complete Knowledge Compilation.

Starting with the whole set of prime implicates, clauses are successively temporarily removed (line 8) from the result set $\mathcal{T}^*$, if the entailment of a clause c can also be obtained from $\mathcal{T}^*$ without c by a UR refutation (line 7). Then a justification $D(c)$ of why the deletion of c was possible is computed (lines 9/10). This justification contains the clauses involved in a shortest UR refutation of c. By removing clause c, UR refutation proofs of other, already removed clauses, may break. So for each previously removed clause c' it is checked whether a UR refutation proof of c' is still possible (lines 11/12). If this is the case, the proof, i.e. the justification $D(c')$, is updated (line 13) analogous to the computation in lines 9/10. Otherwise the formerly removed clause c' is re-added to the working theory $\mathcal{T}^*$ (line 15), and the whole process is repeated until no further changes result, and thus a fixed point is reached.

The main loop of the algorithm may be interrupted after each round, and still returns a UR complete theory equivalent to the input theory, yet not necessarily minimal.

To further illustrate the effects of our algorithm let us consider an example[5]. Let

$$\mathcal{T} = \{\ pqs, p\bar{q}r, \bar{p}qt, \bar{p}\bar{q}u, \bar{t}vw, \bar{v}x\ \}\ .$$

Computation of the set of prime implicates in line 5 of the algorithm then yields

$$\begin{aligned}\mathcal{T}^* = \{\ & pqs, p\bar{q}r, \bar{p}qt, \bar{p}\bar{q}u, \bar{t}vw, \bar{v}x, qst, \bar{q}ru, prs,\\ & \bar{p}tu, rstu, \bar{p}qvw, qsvw, \bar{p}uvw, rsuvw, \bar{t}wx,\\ & \bar{p}qwx, qswx, \bar{p}uwx, rsuwx\ \}\ .\end{aligned}$$

Suppose we first choose $c = rsuwx$ in line 6. Then we have

$$\{\bar{r}, \bar{s}, \bar{u}, \bar{w}, \bar{x}\} \cup \mathcal{T}^* \setminus \{rsuwx\} \vdash_{\mathrm{u}} \bot,$$

and we thus remove c from $\mathcal{T}^*$. A minimal justification $D(c)$ for deleting c is

$$D(c) = \{\ pqs, p\bar{q}r, \bar{p}uwx\ \}\ .$$

Selecting $c = \bar{p}uwx$ next, we obtain $\{p, \bar{u}, \bar{w}, \bar{x}\} \cup \mathcal{T}^* \setminus \{\bar{p}uwx\} \vdash_{\mathrm{u}} \bot$, and therefore we also remove this clause and compute $D(c)$ for it. But now we have for $c' = \{rsuwx\}$ that $c' \in \mathrm{PI}(\mathcal{T}) \setminus \mathcal{T}^*$ and $c \in D(c')$. As $\{\bar{r}, \bar{s}, \bar{u}, \bar{w}, \bar{x}\} \cup \mathcal{T}^* \vdash_{\mathrm{u}} \bot$ still holds, we just have to update the justification, e.g. $D(c') = \{rsuvw, \bar{v}x\}$. Repeating the algorithm's steps over and over we reach the fix-point

$$\mathcal{T}^* = \{\ pqs, p\bar{q}r, \bar{p}qt, \bar{p}\bar{q}u, \bar{t}vw, \bar{v}x, \bar{q}ru, prs, \bar{t}wx\ \}\ .$$

The complexity of our algorithm is dominated by the prime implicate computation step in line 5, which—as is well known—may in the worst case require exponential space (and thus time) (see, e.g., [9]). It remains an interesting task to find an algorithm for UR complete knowledge-base compilation that is not based on prime implicate computation.

6 Preliminary Experimental Results

We are currently starting experiments with our algorithm on databases from automotive product configuration. First results obtained from a prototypical implementation of our algorithm are shown in Table 1.

problem	$\lvert\mathrm{PS}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathrm{PI}(\mathcal{T})\rvert$	$\lvert\mathcal{T}^*\rvert$
Adder	21	50	9,700	1,183
C250FV	1,465	2,356	2,492	1,837
C210FVF	1,934	3,985	496,050,800	–

Table 1. Experimental Results

The first example is taken from Forbus and de Kleer's book [6], which contains databases that are often used as benchmarks in the knowledge compilation community. The following two examples are knowledge bases describing valid model lines of DaimlerChrysler's

[5] This is Example 1 from del Val [4]. We also use his abbreviated notation for clauses, e.g. writing $pq\bar{r}$ instead of $p \vee q \vee \neg r$.

Mercedes cars. These databases are used by Küchlin and Sinz [7] for validation checks.

The columns of Table 1 show in turn the number of propositional variables, the size of the database in number of clauses, the number of prime implicates of the theory, and the number of prime implicates that remain after application of our algorithm.

problem	$t_{\rm PI}$	$t_{\rm reduce}$	$t_{\rm UR}$
Adder	0.38	2683.10	2683.48
C250FV	93.46	220.42	313.88
C210FVF	426.55	–	–

Table 2. Compilation Times

In Table 2 we present runtimes for our UR complete knowledge compilation algorithm. The last column shows the total runtime $t_{\rm UR}$, which is split into the time for prime implicate computation ($t_{\rm PI}$) and reduction of the prime implicate set in algorithm lines 6-14 ($t_{\rm reduce}$). We used Simon and del Val's BDD-based implementation *zres* as a prime implicate generator [14], and ran it on a Pentium III running at 733 MHz. For the reduction part we used an experimental implementation written in Haskell, compiled with the Glasgow Haskell Compiler, version 4.04. Our implementation failed on reducing the prime implicates for the C210FVF data set, which is indicated by a dash in the tables. We are currently working on an implementation in C++ using more efficient data structures.

7 Conclusion and Future Work

We presented a method to compile the fixed part of product configuration databases, and proposed a new algorithm for the computation of UR complete compilations. First experiments indicate that, at least for validation of configuration database properties, our method is applicable.

Besides conducting further experiments, we consider improvement of knowledge compilation algorithms, e.g. by developing exact algorithms that do not require a prior computation of all prime implicates, as a promising area of future research. Moreover, it could be of interest to evaluate the performance of knowledge compilation on other product documentation formalisms and for other practical application areas of configuration.

ACKNOWLEDGEMENTS

I am indebted to Laurent Simon for providing me his *zres* implementation for prime implicate generation.

REFERENCES

[1] Y. Boufkhad, G. Éric, P. Marquis, B. Mazure, and S. Lakhdar, 'Tractable cover compilations', in *Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pp. 122–127, Nagoya, Japan, (August 1997).

[2] M. Cadoli and F.M. Donini, 'A survey on knowledge compilation', *AI Communications*, **10**(3–4), 137–150, (1997).

[3] O. Coudert and J.C. Madre, 'Implicit and incremental computation of primes and essential primes of boolean functions', in *Proc. of the 29th Design Automation Conf. (DAC 1992)*, pp. 36–39, Anaheim, CA, (June 1992).

[4] A. del Val, 'Tractable databases: How to make propositional unit propagation complete through compilation', in *Proc. of the 4th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR'94)*, pp. 551–561, Bonn, Germany, (May 1994).

[5] A. Felfernig, G.E. Friedrich, D. Jannach, and M. Stumptner, 'Consistency-based diagnosis of configuration knowledge bases', in *Proc. of the 14th European Conf. on Artificial Intelligence (ECAI 2000)*, pp. 146–150, Berlin, Germany, (August 2000).

[6] K.D. Forbus and J. de Kleer, *Building Problem Solvers*, MIT Press, 1993.

[7] W. Küchlin and C. Sinz, 'Proving consistency assertions for automotive product data management', *J. Automated Reasoning*, **24**(1–2), 145–163, (February 2000).

[8] P. Marquis, 'Knowledge compilation using theory prime implicates', in *Proc. of the 14th Intl. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pp. 837–845, Montréal, Canada, (August 1995).

[9] P. Marquis, 'Consequence finding algorithms', in *Handbook of Defeasable Reasoning and Uncertainty Management Systems*, eds., D.M. Gabbay and Ph. Smets, volume 5, 41–145, Kluwer, (2000).

[10] D.L. McGuinness and J.R. Wright, 'Conceptual modelling for configuration: A description logic-based approach', *AIEDAM*, **12**(4), 333–344, (1998).

[11] S. Mittal and F. Frayman, 'Towards a generic model of configuration tasks', in *Proc. of the 11th Intl. Joint Conf. on Artificial Intelligence*, pp. 1395–1401, Detroit, MI, (August 1989).

[12] I. Rish and R. Dechter, 'Resolution versus search: Two strategies for SAT', *J. Automated Reasoning*, **24**(1–2), 225–275, (February 2000).

[13] B. Selman and H. Kautz, 'Knowledge compilation and theory approximation', *JACM*, **43**(2), 193–224, (1994).

[14] L. Simon and A. del Val, 'Efficient consequence finding', in *Proc. of the 17th Intl. Joint Conf. on Artificial Intelligence (IJCAI'01)*, pp. 359–365, Seattle, WA, (August 2001).

[15] P. Tison, 'Generalized consensus theory and application to the minimization of boolean functions', *IEEE Transactions on Electronic Computers*, **EC-16**(4), (August 1967).