Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Incremental Algorithms for Approximate Compilation

Alberto Venturini and **Gregory Provan***
Department of Computer Science, University College Cork, Cork, Ireland
Tel: +353 21 4901816, Fax: +353 21 4274390, e-mail: aventurini@gmail.com, g.provan@cs.ucc.ie

Abstract

Compilation is an important approach to a range of inference problems, since it enables linear-time inference in the size S of the compiled representation. However, the main drawback is that S can be exponentially larger than the size of the original function. To address this issue, we propose an incremental, approximate compilation technique that guarantees a sound and space-bounded compilation for weighted boolean functions, at the expense of query completeness. In particular, our approach selectively compiles all solutions exceeding a particular threshold, given a range of weighting functions, without having to perform inference over the full solution-space. We describe incremental, approximate algorithms for the prime implicant and DNNF compilation languages, and provide empirical evidence that these algorithms enable space reductions of several orders-of-magnitude over the full compilation, while losing relatively little query completeness.

Introduction

A broad range of compilation approaches have been proposed in the literature, such as prime implicants (de Kleer 1986), DNNF (Darwiche 2001), and Ordered Binary Decision Diagrams (OBDDs) (Bryant 1992). The benefit of these approaches is that they enable inference that is linear in the size of the compiled representation; the drawback is that the size of the compiled representation can be exponentially larger than that of the original function. For example, the number of prime implicants of a set m of arbitrary clauses is $O(3^m)$ (Chandra and Markowsky 1978), and the size-complexity of compiled representations (e.g., DNNF, OBDD) for problems in propositional logic, Bayesian networks and constraint satisfaction is exponential in the treewidth of their interaction graph (Darwiche 2001).

In this article we propose a sound but incomplete incremental approximation technique that enables us to trade off the size of the compilation for coverage of the most-preferred solutions. We assume that we have a valuation function that identifies the most likely solutions (or satisfying assignments) of f. Such functions are well-known in the literature for a range of applications; for example, in diagnosis we may choose the probabilistically most-likely diagnoses, and in configuration we may choose a cardinality-minimal configuration. Given a preference function ϕ over f, we calculate an approximate compilation $\zeta(f)$ by pruning solutions whose preference value is below a threshold ϱ.

We empirically validate this approach for two important compilation targets, prime implicants and DNNF. For each target language, we propose incremental algorithms that generate approximate (sound but incomplete) representations that trade off query completeness (solution coverage) for space and inference efficiency, without having to compute the full compilation (which may be of size exponential in certain parameters of f). We provide empirical evidence that incremental compilation achieves space reductions of several orders-of-magnitude over the full compilation, while losing relatively little query completeness.

Related Work

This section reviews prior work in related areas. It is important to note that we focus on the *size* of the compilation, which few previous papers have addressed. Most previous studies of compilation have focused on a variety of other questions, such as the existence of solutions, or of relevance and necessity of hypotheses (Eiter and Gottlob 1995). Further, to our knowledge, this is the first proposal of *incremental, approximate* compilation algorithms based on solution preference functions.

The complexity of compilation has been addressed in many papers, including (Cadoli et al. 2002). Ferrara et al. (2007) have proven that for temporal logic model checking, preprocessing cannot reduce complexity, given polynomial space bounds (in the size of the input) on the output. In the case of compiling using BDDs, one cannot guarantee that a BDD will not be of size exponential in the number of variables, since the problem of finding the best variable ordering is NP-hard (Bollig and Wegener 1996). Further, the approximation problem is also hard, as for any constant $c > 1$ it is NP-hard to compute a variable ordering resulting in an OBDD with a size that is at most c times larger than optimal (Sieling 2002). In the case of DNNF, the size of the DNNF generated for problems in propositional logic, Bayesian networks and constraint satisfaction is exponential in the *treewidth of their interaction graph* (Darwiche 2001).[1]

*Supported by SFI grant 04/IN3/I524.

[1] It turns out that many real-world problems, e.g., the ISCAS

Notation and Preliminaries

Propositional Logic

We assume a standard propositional logic in this paper. We use a finite alphabet of propositional symbols, $\Sigma = \{z_1, z_2, ..., z_n\}$, using the usual boolean connectives $\wedge, \vee, \neg$, and $\Rightarrow$ for constructing well-formed formulae from Σ. A literal is a propositional symbol or its negation. A clause $z_1 \vee z_2 \vee \cdots \vee z_k \vee \neg z_{k+1} \vee \cdots \vee \neg z_n$ is a disjunction of literals. A clause is Horn (definite Horn) if $k \leq 1$ ($k = 1$). A function (or formula) f is a conjunction of clauses; in this article we assume that a formula is defined over n symbols, unless stated otherwise. The size of a formula f is $|f|$.

A (partial) interpretation γ for Σ, a mapping from (a subset of) Σ to $\{\mathsf{true}, \mathsf{false}\}$, can be extended to boolean formulae in the usual recursive way. Γ is the set of all interpretations. A *partial solution* is a partial interpretation consistent with f. A *model* of a formula f is an interpretation γ that maps f to true. $\mathcal{B}$ is the set of all boolean formulae over Σ. The function $atoms : \mathcal{B} \rightarrow 2^{\Sigma}$ maps a formula f to the set of propositional symbols occurring in f.

Prime implicants An *implicant* I of a formula f is a conjunction of literals such that $I \Rightarrow f$. An implicant I is a *prime implicant* (PI) if, for every conjunct J obtained by removing one or more literals from I, $J \not\Rightarrow f$. In other words, a prime implicant is a minimal implicant of f. The disjunction Δ of all prime implicants of a formula f is equivalent to f, i.e. Δ preserves the models of f. The size of Δ is the sum of the size of all prime implicants.

Decomposable Negation Normal Form A formula f is in *Negation Normal Form* (NNF) if its literals are joined using only the operators $\vee$ and $\wedge$. The *Decomposable Negation Normal Form* (DNNF) is a subclass of NNF satisfying the decomposability property, i.e., for every conjunction $\alpha = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$ in a DNNF, it holds that $atoms(\alpha_i) \cap atoms(\alpha_j) = \emptyset$ for $i \neq j$. A DNNF can be represented by a rooted directed acyclic graph, where each leaf node is associated with a literal or truth value, and each intermediate node corresponds to either $\vee$ or $\wedge$. Given this representation, the size of a DNNF is the number of edges in the graph.

Compilability

We address abduction problems whose instances can be expressed as the pair $\langle f, \sigma \rangle$, where f is the fixed part (instance-independent) and σ is the varying part (instance-dependent). Given a problem P, an instance $\langle f, \sigma \rangle$ of P with $f \in \mathcal{B}$ and $\sigma \in \Sigma^*$ and a query function $Q_P : \mathcal{B} \times \Sigma^* \rightarrow \{\mathsf{yes}, \mathsf{no}\}$, we are interested in finding the value $Q_P(f, \sigma)$.

Definition 1 (Compilation)**.** *Given a problem P, an instance $\langle f, \sigma \rangle$ of P and a query function Q_P, we define a* compilation function of f, $\zeta(f)$*, such that there exists a query function $Q'_P : \zeta(\mathcal{B}) \times \Sigma^* \rightarrow \{\mathsf{yes}, \mathsf{no}\}$ and $\forall \langle f, \sigma \rangle \in \mathcal{B} \times \Sigma^*$ it holds that $Q'_P(\zeta(f), \sigma) = Q_P(f, \sigma)$.*[2]

It is clear from the definition that a compilation $\zeta(f)$ preserves the models of f. In order to find the value $Q(f, \sigma)$, we process the fixed part f off-line, thus obtaining $\zeta(f)$, and then we find the value $Q'_P(\zeta(f), \sigma)$. Compilation is worthwhile if answering the query $Q'_P(\zeta(f), \sigma)$ is easier than answering $Q(f, \sigma)$. Note that we do not restrict the time needed to compute the function ζ.

This definition of compilation captures all of the compilation approaches that have been proposed in the literature, such as PIs (de Kleer 1986), DNNF (Darwiche 2001), and OBDDs (Bryant 1992). These approaches all make space/time tradeoffs, i.e., they typically generate compilations from which a variety of classes of inference can be done in time linear in the size of the compiled representation, although the compilation may require significant space.

Preference-Based Compilation

This section introduces the notion of a preference function ϕ over f, and of compiling f with respect to ϕ.

Assume that we have a full compilation $\zeta(f)$ such that we can specify the space of solutions, Λ. We can rank-order the solutions of Λ according to a given preference function ϕ into a set of equivalence classes, in which each equivalence class is characterised by the same ϕ value.

We now consider the case where we use a preference function ϕ to guide the choice of compilation target, i.e., we aim to compile the *most-preferred* solutions.

Definition 2 (Preference Function)**.** *A preference function ϕ defined over the solutions Λ of f defines a partial ordering $\succ$ over Λ. We say that solution $\lambda_1 \in \Lambda$ is preferred to solution $\lambda_2 \in \Lambda$, written $\lambda_1 \succ \lambda_2$, if $\phi(\lambda_1) > \phi(\lambda_2)$.*

We now restrict ourselves to the domain of propositional abduction. A propositional abduction problem (PAP) can be defined using a triple $\langle H, \mu, f \rangle$ where H and μ are sets of variables, while f is a propositional formula. H is typically referred to as the hypotheses, and μ as the manifestations. Some important preference-criteria that are widely used in the literature for PAP compilation include Subset-inclusion ($\phi_{\subseteq}$), Cardinality (ϑ) and Probability ($\boldsymbol{Pr}$).

One key criterion is that the compilation preserves preference functions, which is guaranteed for the standard compilation languages, which preserve the models of f.

Definition 3 (Preference Preservation)**.** *A compilation $\zeta(f)$ preserves a preference function ϕ if, given $\langle f, \sigma \rangle$, for any pair of solutions λ_1, λ_2 such that $\lambda_1, \lambda_2 \in \Lambda_f$ and $\lambda_1, \lambda_2 \in \Lambda_{\zeta(f)}$, $\lambda_1 \succ \lambda_2$ is valid in $\zeta(f)$ iff $\lambda_1 \succ \lambda_2$ is valid in f.*

Since we are interested in compiling only the most-preferred solutions, we define *preferred approximate compilations* as compilations including only a subset of most-preferred solutions:

Definition 4 (Preferred Approximate Compilation)**.** *Given a compilation $\zeta(f)$ with space of solutions $\Lambda_{\zeta(f)}$, $\zeta(f)$ denotes a* preferred approximate compilation of f *if: (1)*

benchmark circuits (Brglez, Bryan, and Kozminski 1989), do not have treewidths bounded by some relatively small integer, in order to ensure compact DNNF compilations.

[2]Note that this definition differs from the original definition of (Cadoli et al. 2002), where a compilation is assumed to be of size polynomial in $|f|$ and answering the query Q'_P is assumed to require a time polynomial in $|\sigma| + |\zeta(f)|$.

$\Lambda_{\zeta(f)} \subseteq \Lambda_f$*; and (2)* $\forall \lambda_1 \in \Lambda_{\zeta(f)}, \forall \lambda_2 \in \Lambda_f \setminus \Lambda_{\zeta(f)}$*, it holds that* $\lambda_1 \succ \lambda_2$.

To focus on the most-preferred solutions, we assign a valuation threshold ϱ. We aim to compile all solutions with valuations at least as preferred as ϱ, denoted Λ_ϱ; solutions with valuations less-preferred than ϱ are denoted $\Lambda_{\bar{\varrho}}$.

Definition 5 (ϱ-sound compilation)**.** *Given a preference function ϕ over f and a threshold ϱ, the preferred approximate compilation $\zeta(f)$ denotes a* ϱ-sound compilation of *f if: (1) $\zeta(f)$ preserves the preference function ϕ; and (2) $\zeta(f)$ contains every solution $\lambda \in \Lambda_\varrho$, i.e. $\Lambda_{\zeta(f)} \supseteq \Lambda_\varrho$.*

We have designed our algorithms to incrementally generate partial solutions of increasing size only if the partial solution at each step is more preferred than ϱ. We call this approach partial-solution extension (PSE), which we can prove holds for the set ϕ^* of preference functions $\boldsymbol{\phi_\subseteq}$, $\boldsymbol{\vartheta}$, $\boldsymbol{Pr}$.

Theorem 1. *Given a preference function $\phi \in \phi^*$ over f and a threshold ϱ, partial-solution extension (PSE) is guaranteed to generate a* ϱ-sound compilation of f.

We have developed PSE algorithms for PI and DNNF compilations; we omit full pseudocode descriptions and proofs of the correctness of these algorithms due to space limitations, but note that they are special cases of the above theorem. Our notion of preference-based compilation is different to the use of cost functions for DNNF minimal-diagnosis extraction (Darwiche 1998). The cost-function approach aims to compute the most-preferred diagnosis given a complete DNNF and an observation; in our case we are incrementally compiling all solutions which are more preferred than a given threshold ϱ. Hence, whereas using probabilistic cost-functions can prune a complete DNNF such that valid solutions may be lost (Darwiche 1998), the threshold-based incremental compilation guarantees that no solutions $\lambda \in \Lambda_\varrho$ will fail to be included in the approximate compilation, as shown in Theorem 1.

Incremental Compilation Algorithms

Prime Implicants Our PI algorithm is based on the PRIME algorithm (Shiny and Pujari 2002). The original procedure generates full compilations in a *divide-et-impera* fashion, by (1) dividing a formula f into two sub-formulae, (2) recursively calculating the PIs of each sub-formula, and (3) merging the results in order to obtain the PIs of f. In order to calculate approximate compilations, we modify the third step of the original algorithm by merging only those PIs that are more preferred than the threshold ϱ. In other words, we discard partial solutions that are less preferred than ϱ, in accordance with the PSE approach.

DNNF Our DNNF algorithm is based on the algorithm proposed by Darwiche (Darwiche 2001). The original procedure calculates full DNNF compilations of a formula f by executing the following steps: (1) a specific subset A of variables of f is created; (2) for each instantiation of A-variables, f is divided into two sub-formulae; (3) the DNNF compilation of each sub-formula is calculated recursively; (4) the DNNF compilation of f is calculated by merging the DNNF representations of the sub-formulae.

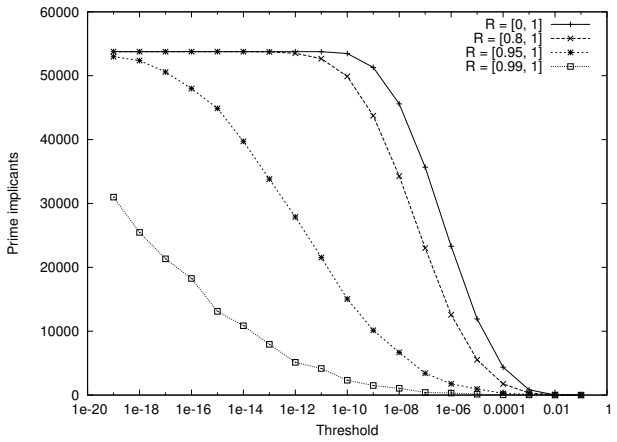

Figure 1: Number of prime impicants of a formula f encoding a 16-gate circuit; each graph refers to a different probability range.

In order to calculate approximate compilations, we modify the second step of the original algorithm so that instantiations less preferred than the threshold ϱ are not considered. Thus, we take into account only those DNNF representations that encode partial solutions more preferred than ϱ.

Empirical Results

We have implemented the above algorithms and carried out experiments that compare the PI and DNNF approaches, measuring the size and coverage trade-off of compilations. All our experiments were run with a set of formulae representing a suite of digital circuits. The digital circuits were generated randomly by a circuit generator program (Provan and Wang 2007), such that the circuits have properties similar to those of the ISCAS circuit benchmarks (Brglez, Bryan, and Kozminski 1989). Each circuit is represented by a formula f defined over a set of boolean variables $V = H \cup K$, where a variable $h_i \in H$ denotes the health of gate i, and variables in K denote input/output signals. To each variable $h_i \in H, i = 1, ..., m$ we assign a probability valuation by randomly sampling the value for $Pr(h_i = 1) = p$ from a uniform distribution over ranges $R = [i, 1]$, with $0 \leq i < 1$. We compute $Pr(H) = \prod_{i=1}^{m} Pr(h_i)$ by assuming that all components fail independently.

Figure 1 shows how the number of solutions varies with different thresholds and weight ranges. Depending on the range R we obtain distributions that decrease more or less gradually to 0. The number of preferred PIs decreases as $i \rightarrow 1$ for all threshold values $\varrho > 0$. We can control the number of solutions (and thus the size and coverage of the approximate compilation) by selecting appropriate (i, ϱ) combinations. In the following experiments we used a range $R = [0.99, 1]$; however, our experimental data indicate that we obtain similar results using different ranges.

Figure 2 compares the size of a full PI compilation with full and approximate DNNF compilations. The size of compilations grows exponentially in the size of the input; however, a full DNNF compilation is more succint than a full PI compilation, achieving in some cases a memory saving of 50%. Moreover, approximate compilations can save a

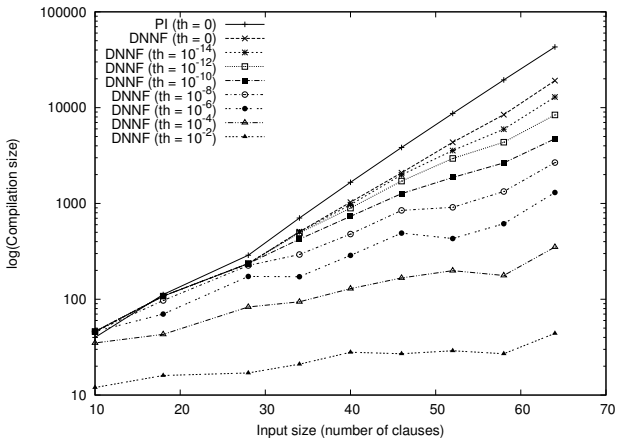

Figure 2: Comparing the size of full PI compilations with full and approximate DNNF compilations.

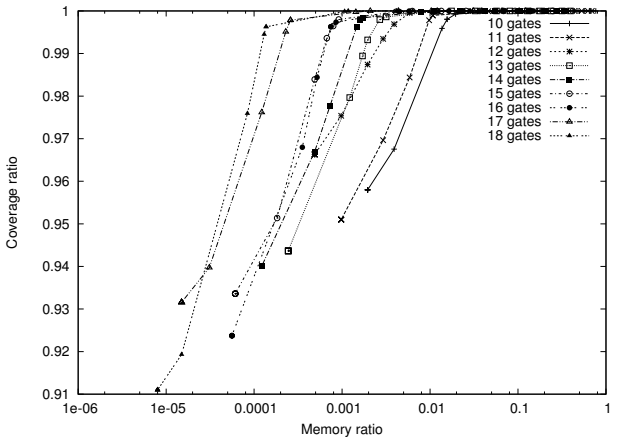

Figure 3: CDFs of solution distributions for PI compilations. Given a full compilation ζ_f and an approximate compilation ζ_a, Memory Ratio is defined as $mr = |\zeta_a|/|\zeta_f|$ and Coverage Ratio is defined as $cr = (\sum_i Pr(\lambda_i))/(\sum_j Pr(\lambda_j))$, with $\lambda_i \in \zeta_a$ and $\lambda_j \in \zeta_f$.

significant amount of space when compared to full compilations. We obtain similar results with approximate PI compilations.

Figure 3 shows the Cumulative Distribution Functions (CDFs) of solution distributions for circuits with 10 to 18 gates, encoded in formulae with 58 to 108 clauses. These graphs refer to PI compilations; we obtain similar results using DNNF. Here we see that, using approximate compilations, we obtain significant solution coverage yet require just a fraction of the memory of the full compilation. In particular, we obtain up to 5 orders-of-magnitude space savings, while maintaining $> 90\%$ query-coverage; moreover, for all circuits very high coverage ratio ($> 99\%$) is possible with 3-4 orders-of-magnitude space savings. Note that the space savings increase with circuit (or formula) size, meaning that this approach scales well with the size of f.

Summary and Conclusions

We have proposed a Partial-Solution Extension approach for incrementally compiling the most-preferred solutions of boolean formulae, which focuses on the size and solution coverage of approximate compilations, as well as on the memory required to compute them. We have provided a theoretical analysis and presented PSE algorithms for PIs and DNNF target languages. All the algorithms are ϱ-sound, i.e. they compute approximate compilations that include all solutions more preferred than the valuation threshold ϱ. Experimental results have been reported that empirically demonstrate the space efficiency of approximate compilations, showing that we can achieve orders-of-magnitude space savings while covering the majority of solutions.

References

Bollig, B., and Wegener, I. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers* 45(9):993–1002.

Brglez, F.; Bryan, D.; and Kozminski, K. 1989. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems*, volume 3, 1929–1934.

Bryant, R. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. volume 24. 293–318.

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

Chandra, A., and Markowsky, G. 1978. On the number of prime implicants. *Discrete Mathematics* 24:7–11.

Darwiche, A. 1998. Model-Based Diagnosis using Structured System Descriptions. *Journal of Artificial Intelligence Research* 8:165–222.

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

de Kleer, J. 1986. An assumption-based TMS. *Artif. Intell.* 28(2):127–162.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM (JACM)* 42(1):3–42.

Ferrara, A.; Liberatore, P.; and Schaerf, M. 2007. Model Checking and Preprocessing. In *Proc. AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer.

Provan, G., and Wang, J. 2007. Evaluating the adequacy of automated benchmark model generators for model-based diagnostic inference. In *Proceedings of IJCAI-07*.

Shiny, A., and Pujari, A. 2002. Computation of Prime Implicants using Matrix and Paths. *Journal of Logic and Computation* 8(2):135–145.

Sieling, D. 2002. The nonapproximability of obdd minimization. *Inf. Comput.* 172(2):103–138.