Incremental Compilation-to-SAT Procedures

Marco Benedetti[§] and Sara Bernardini

Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Via Sommarive 18, 38055 Povo, Trento, Italy
{benedetti,bernardini}@itc.it

Abstract. We focus on *incremental compilation-to-SAT procedures* (iCTS), a promising way to push the standard CTS approaches beyond their limits. We propose the first comprehensive framework that encompasses all the aspects of an *incremental decision procedure*, from the encoding to the incremental solver. We apply our guidelines to a real-world CTS approach (*Bounded Model Checking*) and show how to modify both the generation mechanism of a real BMC tool (*NuSMV*) and the solving engine of a public-domain SAT solver (*SIM*). Related approaches and experimental results are discussed as well.

1 Introduction

Many decision and search problems may be successfully tackled by generating and solving a chain of increasingly complex SAT instances. Well known examples of *compilation-to-SAT* (CTS) procedures exist: computer-aided design of integrated circuits [19,17], planning [15], model checking for dynamic systems [6], scheduling [8], operations research and cryptography, just to name a few. These techniques share an underlying working schema. They first establish an ordering among classes of potential solutions. Small and short solutions come first. More and more complex candidates follow. Each class is then mapped onto a SAT instance solved by a general purpose solver [20,13].

One remarkable strength of this family of techniques is *modularity*: state-of-the-art SAT solvers can be picked off-the-shelf and applied to the solution step. Thus, every advance from the SAT community is possibly transferred to the above procedures with a minimum effort. Also, advancements proceed the other way around: a great part of the renewed interest in propositional decision procedures (and of the boost of performances of SAT solvers during the last ten years) is due to the relevance and generality of the above family of techniques.

As usual, modularity shows an unpleasant side: solvers have to be treated as almost completely black boxes. This choice limits the amount of information exchanged between the generating and the solving side during *one single round* of the procedure, besides preventing information exchange among *subsequent rounds*. The solver thus misses the key point that it is presented with a *chain* of strictly related instances. A further underestimated duty one pays for easily plug standard solvers in, is the flattening of highly structured instances down to a conjunctive normal form (the standard input format for general SAT solvers).

Some approaches have recently emerged to exploit the crucial observation that neither a given instance in a chain is unrelated to the previous ones, nor the solver is approaching a completely different search problem every time [17,24,23,5]. These approaches aim both to increase the efficiency of the overall decision procedure and to allow reasoning that don't fit well within the usual CTS framework.

In this paper, we present the first comprehensive framework that encompasses all the aspects of an *incremental decision procedure* based on propositional satisfiability. After a few notation and preliminaries (Section 2 and 3), we characterize a large family of CTS approaches that are eligible for *incrementalisation*, and stress the often overlooked issue of turning a standard encoding machinery into an incremental one (Section 4). Also, issues arising on the solving side are addressed, and two detailed examples are developed during the presentation. We apply our guidelines to the incrementalisation of a specific CTS approach (*Bounded Model Checking*, or BMC for short [6]) and show how to modify both the generation mechanism employed by a real BMC tool (*NuSMV* [7]) and a public-domain SAT solver (*SIM* [13]) (Section 5).

We carefully review the related literature in Section 6, and then present our conclusions and future work in Section 7. A more thorough presentation of our technique and the proofs of all the results are given in [2].

[§] This work is funded by PAT (*Provincia Autonoma di Trento*, Italy) under grant n. 3248/2003.

2 Notation

Given a CNF formula f and a set of literals Δ on the variables $var(f)$ of f, we denote by $f * \Delta$ the propositional formula obtained from f after the assignment Δ is made, i.e. the clause set obtained after unit subsumption ad unit resolution have been performed against each literal in Δ considered as a unit clause. Given two propositional formulas f_1 and f_2 and a set of propositional variables $V \subseteq var(f_1) \cap var(f_2)$, we write $f_1 \equiv_V f_2$ when the set of models of f_1 projected onto V is equal to the set of models of f_2 projected onto V. We graphically represent such formulas by means of direct acyclic graphs that avoid sub-formula replications (known as RBC and extensions thereof, see [1]). Yet, propositional solvers often require a conjunctive normal form (CNF) to work. We denote by $cnf(f)$ the set of clauses such that $cnf(f) \equiv_{var(f)} f$ which is obtained along the guidelines described in [10,21]. The cnf function is omitted whenever the context suffices to understand that a CNF formula is required.

3 CTS approaches

Most CTS frameworks tacitly exploit the deduction theorem over a language $\mathcal{L}$ (more expressive than propositional logic), by stating that $\mathcal{T} \models P \Leftrightarrow \not\models \mathcal{T} \wedge \neg P$, where $\mathcal{T}$ is a consistent theory that models a relevant phenomenon or system or protocol, while P expresses an (un)desired property of that phenomenon/system/protocol. The problem is to decide the consistency of $W = \mathcal{T} \wedge \neg P$.

A mechanism purposely designed to get rid of the excess of expressive power of $\mathcal{L}$ stays at the very heart of every CTS framework. It allows moving to propositional logic by considering chains of *bounded* versions of the original problem obtained through a function $[\![.]\!]_. : \mathcal{L} \times \mathbb{N} \rightarrow Prop$ - called *encoding function* - that maps a formula $W \in \mathcal{L}$ and a bound k onto a propositional formula $[\![W]\!]_k$ on variables $V_k = var([\![W]\!]_k)$.

From the point of view of a state-space search, things work as follows: (1) The space of possible solutions to the problem is partitioned according to a bound k identifying finite classes C_k of possible models for W (the larger the bound k the more complex the solutions in C_k); (2) an encoding $[\![W]\!]_k$ - satisfiable iff a solution for W happens to lay in C_k - is computed together with a decoding function mapping propositional models of satisfiable encodings onto solutions to W in C_k; (3) $[\![W]\!]_k$ is solved; whenever it comes out to be satisfiable, the decoding function plays its role in reconstructing a solution to W. Step 2 is selected for another round with a higher bound when no solution exists in C_k. The loop is exited when either some resource limit is exhausted or it is possible to prove that none of the remaining $C_i,\ i > k$ contains solutions. So, the problem of finding out whether or not W has models is answered by deciding a sequence of SAT problems on $\{[\![W]\!]_i, i = 0, 1, 2, ...\}$.

The peculiar structure of W - due to application of the deduction theorem within $\mathcal{L}$ - is maintained after the propositional translation, provided the encoding function is commutative w.r.t. negation and distributive w.r.t. conjunction. So, we have a *structured* CTS sequence $[\![W]\!]_k = [\![\mathcal{T} \wedge \neg P]\!]_k = [\![\mathcal{T}]\!]_k \wedge \neg [\![P]\!]_k$, where $[\![P]\!]_k$ is usually by far smaller than $[\![\mathcal{T}]\!]_k$ and nonetheless responsible for potential inconsistencies in $[\![W]\!]_k$.

As an example of a CTS approach, let us consider SAT-based classical planning [15]. It works by encoding into $[\![W]\!]_k$ two components: (1) an instance $[\![\mathcal{T}]\!]_k$ of the theory describing the planning domain in terms of the interconnected preconditions and effects *of at most k layers of actions* (together with other constraints such as mutual exclusion conditions between pairs of actions in the same layer), and (2) the condition or goal $[\![P]\!]_k$ to be reached after the last layer of actions has been executed.

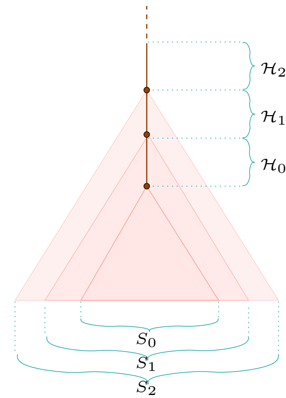
Concepts out of reach for raw propositional logic here are the universal quantifiers in front of the action schemata, and the existence of fluent predicates along the infinite timeline of the modeled world. Both of them are dealt with by propositionally *instantiating* state variables and action schemata as many times as needed.

In the basic encoding, each operator is instantiated with all the possible combinations of arguments to obtain several parameterless (boolean) actions. As the number of objects in classical planning domains doesn't change over time, this *groundization* can be done once for all and doesn't require incrementality. Conversely, the unrolling of plans over the time line (in terms of the number of action layers) is potentially infinite. Indeed, the bound for this CTS approach represents the maximal number of action layers in the solution plan we are currently looking for, and C_i is the set of feasible plans with exactly i layers of actions.

As classical planning domains have a finite state space in spite of the infinite number of feasible plans, it is also possible to check the set of reached states for saturation, thus ensuring that no solution exists for unfeasible goals. In case a satisfiable instance is encountered, the resulting plan immediately grows out of the given model as filtered by the decoding function, that remembers (1) which layer of action and status are associated with each propositional state variables, and (2) which layer and parameters are associated with propositional action instantiations.

4 Incremental Compilation-to-SAT (iCTS)

Every iCTS approach is made up of an incremental solver, an incremental generation mechanism and an architecture that explains how these components interact. As opposite to classical SAT solvers, an incremental solver is a persistent object partly aware of its surroundings that addresses the problem of deciding a chain of related satisfiability instances as a whole, thus re-using information gathered from past search.



Let us consider a SAT solver as a search engine in the space of truth assignments over $V = var(f)$ attempting to make f evaluate to true. Then, an iSAT solver is a search engine that explores a search space S defined only *once per chain*, not once per instance. Each instance f_i in a chain $\{f_i, i = 0, 1, ...\}$ specifies which portion S_i of the whole search space has to be searched for a solution. When a subspace is proved empty, a larger subspace (monotonically containing the previous ones) is considered.

As depicted in the picture aside, subspaces are connected to one another by means of some special sets of propositional hypotheses $\mathcal{H}_i$ that mark the boundary between S_i and S_{i+1}, in so as S_i is just the subspace of S_{i+1} rooted at the branch $\mathcal{H}_i$.

Definition 1 (iSAT problem). *An iSAT instance is a sequence of couples $\{\langle f_i, \mathcal{H}_i\rangle, i = 0, 1, ...\}$ where f_i is a CNF formula, $\mathcal{H}_i \in var(f_i)$ is a set of propositional hypotheses, and $\forall i.f_i \subseteq f_{i+1}$. The iSAT problem consists of deciding whether $\exists i.SAT(f_i * \mathcal{H}_i)$.*

An iSAT instance is passed to an incremental solver step by step by repeatedly invoking the primitive "`enlargeSearchSpace(`Δf_i`,`$\mathcal{H}_i$`)`" to notify the dimension $|var(f_i) \setminus var(f_{i-1})|$ and the "shape" f_i of the new subspace to be explored, together with the position $\mathcal{H}_i$ where it is attached as a subspace of S_{i+1} (with $\Delta f_i = f_i \setminus f_{i-1}$) .

When the time for implementation comes, it is by far convenient to modify an existing DPLL solver (thus retaining state-of-the-art technology) at the expense of performing some modifications.

Each instance f_i is considered under the hypotheses $\mathcal{H}_i$, placed at the very bottom of the search stack. Standard solvers are allowed to withdraw every stacked hypothesis as soon as it comes out to be responsible for inconsistencies. An incremental solver behaves in the same way in all the cases but when the hypothesis to be removed is within $\mathcal{H}_i$. By removing such hypothesis it would indeed escape from S_i. Rather, it stops working and waits for the next enlargement of the search space. Search is then restarted across the newly added subspace by removing the selected source of inconsistency. The hypotheses $\mathcal{H}_i$ loose their inviolability, which is inherited by $\mathcal{H}_{i+1}$.

We modified the SIM solver [13] to obtain i-SIM [2] by (1) slightly revising the standard LIFO policy employed by the stack of hypotheses to allow the insertion of $\mathcal{H}_i$, (2) substituting "*the stack only contains (a subset of) $\mathcal{H}_i$, wait!*" for "*the stack is empty, quit!*" as a stop condition and (3) making eligible for dynamic enlargement all the internal data structures whose size depends on the number of variables and/or clauses in the formula (taking care to keep consistency between all mutual references).

We now briefly show which kind of connections among adjacent instances can be leveraged during the solving process. For a more thorough description of our technique we refer the reader to [2].

Let us consider a structured CTS problem on $W = \langle \mathcal{T}, P\rangle$ that generates a sequence of SAT instances $[\![W]\!]_i = [\![\mathcal{T}]\!]_i \wedge \neg[\![P]\!]_i$ with a *monotone encoding* for the background theory ($\forall i.[\![\mathcal{T}]\!]_i \subseteq [\![\mathcal{T}]\!]_{i+1}$).

Definition 2 (Incremental Encoding). *An* incremental encoding *for* $\{[\![W]\!]_i, i = 0, 1, ...\}$ *is a sequence of couples* $\{\langle [\![\mathcal{T}, P]\!]_i^+, \mathcal{H}_i \rangle, i = 0, 1, ...\}$, *with* $[\![\mathcal{T}, P]\!]_0^+ = I_0 \wedge P_0$, $[\![\mathcal{T}, P]\!]_i^+ = [\![\mathcal{T}, P]\!]_{i-1}^+ \wedge \Delta[\![\mathcal{T}, P]\!]_{i-1}^i$, $\Delta[\![\mathcal{T}, P]\!]_i^{i+1} \doteq \Delta[\![\mathcal{T}]\!]_i^{i+1} \wedge \Delta[\![P]\!]_i^{i+1}$ *and*

$$\begin{cases} [\![\mathcal{T}]\!]_0^+ \doteq I_0 \\ [\![\mathcal{T}]\!]_i^+ \doteq [\![\mathcal{T}]\!]_{i-1}^+ \wedge \Delta[\![\mathcal{T}]\!]_{i-1}^i \ i > 0 \end{cases} \qquad \begin{cases} [\![P]\!]_0^+ \doteq P_0 \\ [\![P]\!]_i^+ \doteq [\![P]\!]_{i-1}^+ \wedge \Delta[\![P]\!]_{i-1}^i \ i > 0 \end{cases}$$

where $[\![\mathcal{T}]\!]_i^+$ *(*$\Delta[\![\mathcal{T}]\!]_{i-1}^i$*) is the* incremental *(*differential*)* encoding *for a monotone background theory* $\mathcal{T}$ *and is such that* $\forall i. [\![\mathcal{T}]\!]_i^+ \equiv [\![\mathcal{T}]\!]_i$, *while* $[\![P]\!]_i^+$ *(*$\Delta[\![P]\!]_{i-1}^i$*) is the* incremental *(*differential*)* encoding *for the property* $P \in \mathcal{L}$ *and is such that* $\forall i. [\![P]\!]_i^+ * \mathcal{H}_i \equiv_{V_i} [\![P]\!]_i$ *for a sequence* $\{\mathcal{H}_i, i = 0, 1, ...\}$ *of sets of literals called* closing set of hypotheses over $[\![P]\!]_i^+$. *The incremental property encoding is built after an* open encoding *over* $\mathcal{L}$, *which is a function mapping a formula* $Q \in \mathcal{L}$ *and a couple of indexes* k, k' *(*$k \leq k'$*) onto a propositional formula* $]\!]Q]\!]_k^j$ *in such a way that* $[\![P]\!]_0^+ = P_0 =]\!]P]\!]_0^0$, $\mathcal{H}_k = \{\neg\phi_k^Q | \phi_k^Q \in [\![P]\!]_k^+\}$, *and* $\Delta[\![P]\!]_k^{k'} = \bigwedge_{\phi_k^Q \in [\![P]\!]_k^+} \phi_k^Q \rightarrow]\!]Q]\!]_{k'}^{k+1}$.

The key property of an incremental encoding is that *it defines an iSAT instance satisfiability equivalent to the corresponding sequence of standard encodings*, provided a valid open encoding over $\mathcal{L}$ is defined. Open encodings mimic the usual encodings but introduce a number of additional literals ϕ_k^Q (associated to a bound k and to a formula $Q \in \mathcal{L}$) used by the subsequent open encodings as coupling points to have the overall formula grow up: after $]\!]Q]\!]_k^k$ inserts a closing literals $\phi_k^{Q'}$, $\Delta[\![P]\!]_k^{k+1}$ expands its meaning $]\!]Q']\!]_{k+1}^{k+1}$ and this in turn creates coupling points for the next round.

At the clause level, a valid open encoding for adjacent bounds may be obtained by posing $]\!]P]\!]_k^k := (\Delta_k^+ \setminus \Delta_{k+1}^-) \cup \{\phi_k^P \vee \Gamma | \Gamma \in \Delta_{k+1}^-\}$, given the sequences of sets of clauses such that $cnf([\![P]\!]_{i+1}) = cnf([\![P]\!]_i) \setminus \Delta_i^- \cup \Delta_i^+$.

Alternatively, we may *incrementally CNF-ize* any encoding exhibiting the two properties $]\!]P]\!]_{k'}^0 * \mathcal{H}_{k'} \equiv [\![P]\!]_{k'}$ and $]\!]P]\!]_{k'}^k \cdot \sigma_{k'} =]\!]P]\!]_{k'+1}^k$ for the substitution $\sigma_{k'} = \{]\!]f]\!]_{k'+1}^{k'+1} / \phi_k^f \mid \phi_k^f \in pure(]\!]P]\!]_{k'}^k)\}$. These conditions are met by open encodings built over the family of *disjunctive* property chains. For these chains it is $\forall i \geq 0. [\![P]\!]_i \rightsquigarrow [\![P]\!]_{i+1}$, where $g \rightsquigarrow f$ (f is *disjunctively expanded* from g) iff a formula h with pure literals on $\{v_1, ..., v_n\}$ and two substitutions $\sigma_g = \{g_1/v_1, ..., g_n/v_n\}$ and $\sigma_f = \{(g_1 \vee f_1)/v_1, ..., (g_n \vee f_n)/v_n\}$ exist such that $g = h \cdot \sigma_g$ and $f = h \cdot \sigma_f$. Thereafter, incremental CNF-ization amounts to ensure consistency across the clause versions of all the formulas undergoing CNF-ization by maintaining the same meaning for variables shared among differential encodings.

The intuition behind the incremental encoding is that to connect subsequent meshes of the chain we just need to focus on the property encoding, whose open encoding indeed exhibits the forethought of spreading place-holders across the formula as coupling points between adjacent instances. When the solver stops searching and a resolution tree rooted at the empty clause is found, either it is independent from stacked hypotheses (in this case, not only the current SAT instance but the whole iSAT instance is inconsistent), or some closing hypothesis lies among its leaves. In the latter case, the empty clause fails to survive the backtrack step over the closing hypothesis, so the search can restart over the enlarged problem defined by gathering new constraints.

A simple example that captures many relevant aspects of the iCTS framework is the following. We incrementally test a *shift register* with n bits and the entry bit always equal to 0 against the (false) property "*the register never becomes empty if the two most significant bits are initially set*".

Formally, if $b_j(t)$ is the value of the j-th bit after t shifts, the system is described by $\mathcal{T} = Init \wedge \forall t > 0. \neg b_0(t) \wedge \forall j > 0. b_j(t) \leftrightarrow b_{j-1}(t-1)$, where $Init = b_{n-1}(0) \wedge b_{n-2}(0)$, and the property is $\overline{P} = \neg\exists t. \forall j. \neg b_j(t)$. By the deduction theorem, we assert that the property holds iff no model exists for $\mathcal{T} \wedge \neg\overline{P}$, i.e.: for $W = \mathcal{T} \wedge P$ where $P = \neg\overline{P}$.

Even though the above theory is too expressive to be directly translated into propositional logic (t ranges over an infinite domain) conjunctions/disjunctions over a finite set of propositional variables $V_k^i = \{b_j(t), j = 0, ..., n-1, t = 0, ..., k\}$ can be substituted for the quantifiers given any finite time horizon k. The incremental version of the resulting propositional theory (up to step 3) is depicted in the bottom half of Figure 1.

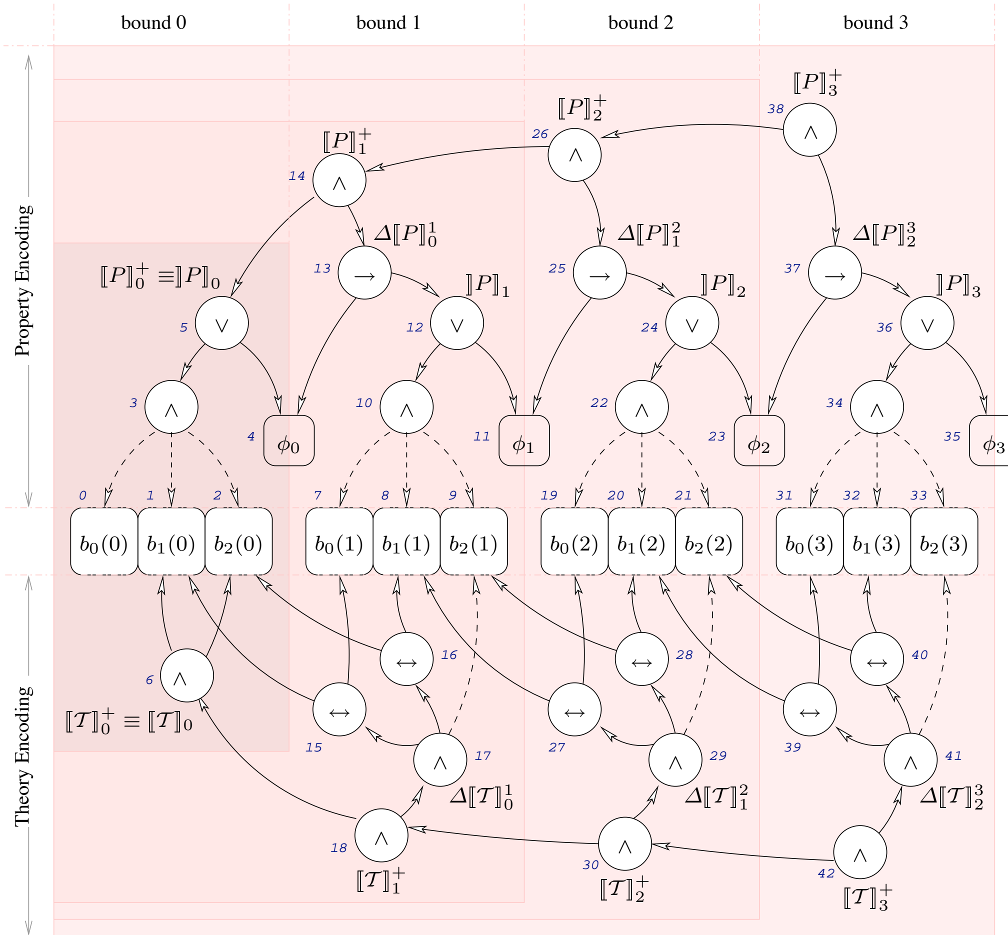


Fig. 1. An example of incremental encoding for a BMC problem.

An incremental encoding for the property $[\![P]\!]_k \equiv \bigvee_{t=0}^{k} \bigwedge_{j=0}^{n-1} \neg b_j(t)$ is obtained by choosing $\rrbracket P]\!]_t^t = \wedge_{j=0}^{n-1} \neg b_j(t) \vee \phi_i$ (upper half in the figure), and it is easy to check that $[\![P]\!]_t^+ * \neg\phi_t \equiv_{V_t} [\![P]\!]_t$. As the instance is built incrementally, argument-arrows never end in a region lighter than the one they originate from, and the only link between adjacent instances is provided by the open variables $\{\phi_i\}$. The small numeric labels near each node represent an incremental labelling for the CNF-ization procedure that generates adjacent and consistent ranges of propositional variables across subsequent encodings.

The iSAT solver is initially provided with the problem $cnf([\![\mathcal{T}, P]\!]_0^+)$ to be solved under the hypothesis $\neg\phi_0$. An inconsistency is detected soon (the property does not hold in the initial state). By traversing the dependency graph, the solver discovers that the hypothesis $\neg\phi_0$ is responsible for such contradiction. It tells the generation machinery, which in turn generates the differential part of the encoding from step 0 to step 1 and produces the clauses arising from the incremental CNF-ization of the subgraph rooted at $[\![\mathcal{T}, P]\!]_1^+$. The solver is notified of the new size of the problem, and is then given both the clause-set $cnf(\Delta[\![\mathcal{T}, P]\!]_0^1)$ and the new working hypothesis $\neg\phi_1$. Then, it dismisses $\neg\phi_0$ (it also notices that $\neg\phi_0$ was at the very bottom of the stack, so the unit clause ϕ_0 is learned, and this amounts to learn that $[\![\mathcal{T}, P]\!]_0$ has no model).

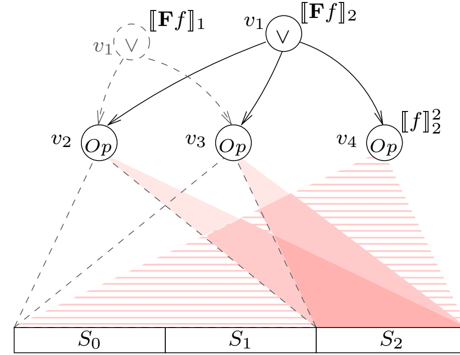
This incremental generate-and-solve loop goes on encountering other contradictions until step 3, when $[\![\mathcal{T}, P]\!]_3^+$ is considered under the hypothesis $\neg\phi_3$. A model for $[\![\mathcal{T}, P]\!]_3^+ * \neg\phi_3$ is found and used to reconstruct a 3-step witness falsifying the property. The solver maintains its internal state through the whole process and also retains all the consequences of the already performed search. In this simple example, the necessary truth value of many variables and some unit clauses are inherited from previous runs.

5 Bounded Model Checking as a testbed

BMC [6] is a SAT-based automatic technique to verify a reactive system modelled as a finite state automaton M against a property f expressed in *linear temporal logic* (LTL). The semantic entailment $M \models_k \mathbf{E}\neg f$ to be checked is dealt with by solving $[\![M, f]\!]_k := [\![M]\!]_k \wedge [\![\neg f]\!]_k$, where $[\![M]\!]_k$ is a k-step long boolean encoding of the transition relation associated with M, while $[\![\neg f]\!]_k$ unrolls the semantics of $\neg f$ over a path of length k by representing all the possible behaviours violating f on such path.

We refer the reader to [6,2] for a detailed description of this technique and to [3] for the PLTL (an extension of LTL) standard encoding we start from.

To incrementalise this technique we first generate a differential encoding for the monotone background theory: $I_0 \doteq I(s_0)$ and $\Delta[\![M]\!]_k \doteq T(s_{k-1}, s_k)$ (see [6,2,5]). According to [3], two adjacent property encodings only differ as each future time operator unrolls its semantics to the newly added time step.



From a CNF point of view, the clauses to be added/removed are obtained by recursively considering such operators, and are obtained by traversing the syntactic tree of the PLTL formula and conjuncting the set of clauses to be added/removed due to each node labeled by a future time operator.
The picture aside shows that shifting from $[\![\mathbf{F}f]\!]_1$ to $[\![\mathbf{F}f]\!]_2$ the CNF translation of the node v_1 changes from $cnf_a = cnf(v_1 \leftrightarrow (v_2 \vee v_3)) = \{\{\neg v_1, v_2, v_3\}, \{\neg v_2, v_1\}, \{\neg v_3, v_1\}\}$ to $cnf_b = cnf(v_1 \leftrightarrow (v_2 \vee v_3 \vee v_4)) = \{\{\neg v_1, v_2, v_3, v_4\}, \{\neg v_2, v_1\}, \{\neg v_3, v_1\}, \{\neg v_4, v_1\}\}$ and that the clauses $cnf_c = cnf([\![f]\!]_2^2)$ appear, so $\Delta_2^+ = \{\{\neg v_1, v_2, v_3, v_4\}, \{\neg v_4, v_1\}\} \cup cnf_c \cup \Gamma_2^+$ and $\Delta_2^- = \{\{\neg v_1, v_2, v_3\}\} \cup \Gamma_2^-$, where Γ_2^+ and Γ_2^- are computed by recursively looking for nested future time operators within the subtrees rooted at v_2 and v_3. These expressions can be easily generalized to shift from k to $k+1$.

Rather than working at the clause level, we may construct a higher-level incremental procedure that exploits the semantics of time operators. This procedure acts *before* the CNF converter is presented with

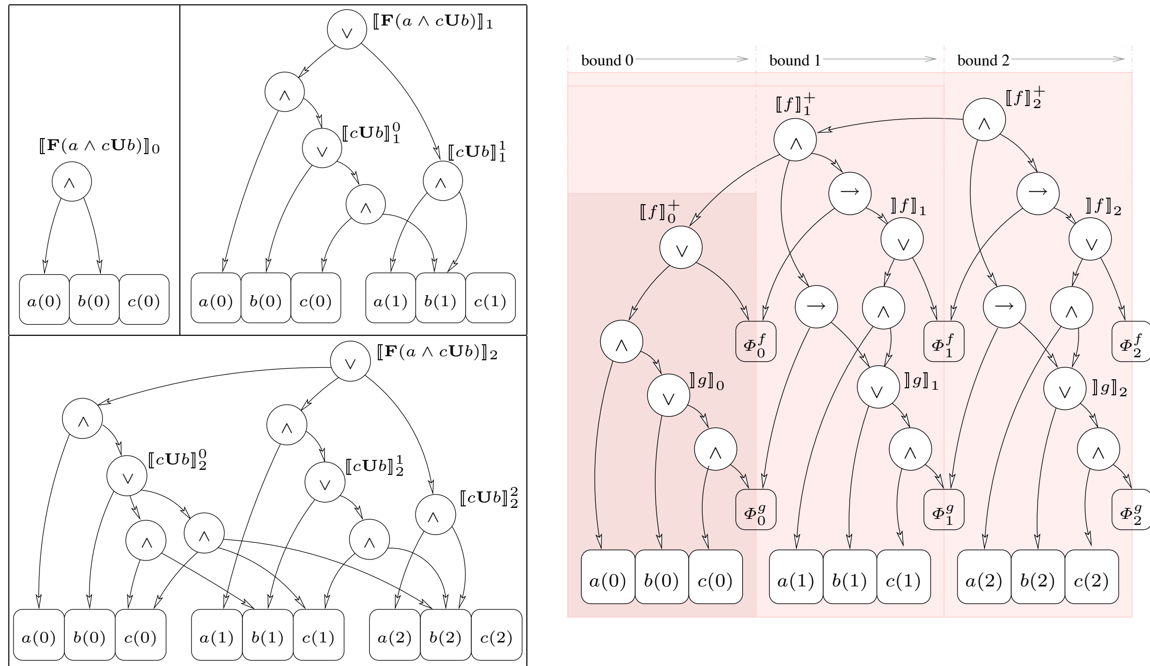


Fig. 2. Standard and incremental encoding for $f = \mathbf{F}(a \wedge g) = \mathbf{F}(a \wedge c\mathbf{U}b)$ with $g = c\mathbf{U}b$

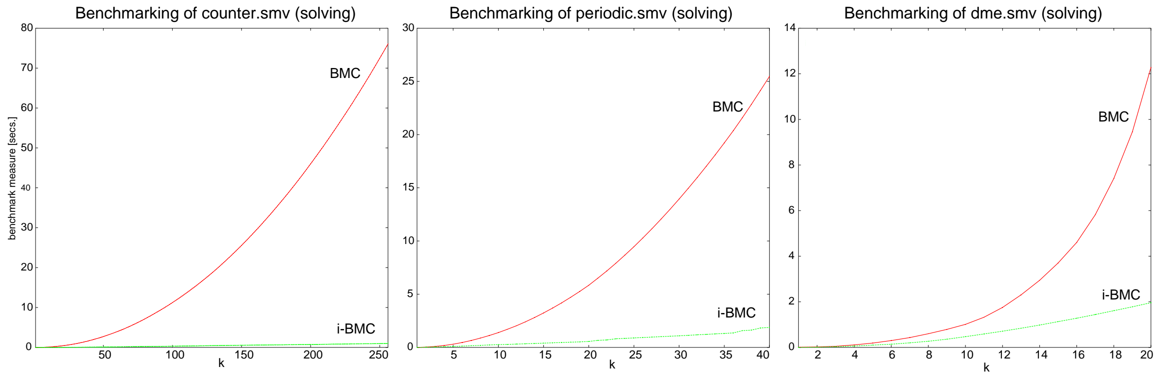


Fig. 3. Solving time compared on 3 BMC chains (310 instances)

the formula, thus yielding a more intuitive encoding. It follows the guidelines given in the previous section and consists of defining a valid open encoding (see [3] for details) for PLTL formulas.

Definition 3 (Open PLTL Encoding). *The open translation of a PLTL formula from bound i to bound k ($i \leq k$) is a propositional formula inductively defined as follows.*

$$
\begin{array}{llll}
]\!]q]\!]^i_k \doteq q^i \quad]\!]\neg q]\!]^i_k \doteq \neg q^i &]\!]f \wedge g]\!]^i_k \doteq]\!]f]\!]^i_k \wedge]\!]g]\!]^i_k &]\!]f \vee g]\!]^i_k \doteq]\!]f]\!]^i_k \vee]\!]g]\!]^i_k \\
]\!]\mathbf{X}f]\!]^i_k \doteq \begin{cases} \phi^f_k & i = k \\ ]\!]f]\!]^{i+1}_k & i < k \end{cases} &]\!]\mathbf{F}f]\!]^i_k \doteq \bigvee_{j \in [i,k]}]\!]f]\!]^j_k \vee \phi^{\mathbf{F}f}_k &]\!]\mathbf{G}f]\!]^i_k \doteq \bot
\end{array}
$$

$$
]\!]f\mathbf{U}g]\!]^i_k \doteq \bigvee_{j \in [i,k]} \left( ]\!]g]\!]^j_k \wedge \bigwedge_{h \in [i,j)}]\!]f]\!]^h_k \right) \vee \left(\phi^{f\mathbf{U}g}_k \wedge \bigwedge_{h \in [i,k]}]\!]f]\!]^h_k \right)
$$

$$
]\!]f\mathbf{R}g]\!]^i_k \doteq \bigwedge_{j \in [i,k]} \left( ]\!]g]\!]^j_k \vee \bigvee_{h \in [i,j)}]\!]f]\!]^h_k \right) \wedge \left(\neg\phi^{f\mathbf{R}g}_k \vee \bigvee_{h \in [i,j)}]\!]f]\!]^h_k \right)
$$

$$
\begin{array}{ll}
]\!]\mathbf{Y}f]\!]^i_k \doteq \begin{cases} \bot & i = 0 \\ ]\!]f]\!]^{i-1}_k & i > 0 \end{cases} &]\!]\mathbf{Z}f]\!]^i_k \doteq \begin{cases} \top & i = 0 \\ ]\!]f]\!]^{i-1}_k & i > 0 \end{cases} \\
]\!]\mathbf{O}f]\!]^i_k \doteq \bigvee_{j \in [0,i]}]\!]f]\!]^j_k &]\!]f\mathbf{S}g]\!]^i_k \doteq \bigvee_{j \in [0,i]} \left( ]\!]g]\!]^j_k \wedge \bigwedge_{h \in (j,i]}]\!]f]\!]^h_k \right) \\
]\!]\mathbf{H}f]\!]^i_k \doteq \bigwedge_{j \in [0,i]}]\!]f]\!]^j_k &]\!]f\mathbf{T}g]\!]^i_k \doteq \bigwedge_{j \in [0,i]} \left( ]\!]g]\!]^j_k \vee \bigvee_{h \in (j,i]}]\!]f]\!]^h_k \right)
\end{array}
$$

Figure 2 compares this incremental encoding with the standard one for a sample PLTL formula. The former, instead of growing form right to left in a definitely non-incremental manner as the latter does, is such that $[\![f]\!]^+_0 \subseteq [\![f]\!]^+_1$ and $[\![f]\!]^+_1 \subseteq [\![f]\!]^+_2$. The semantics of the original encoding is nonetheless preserved according to $[\![f]\!]_k \equiv_{V_k} [\![f]\!]^+_k * \mathcal{H}_k$. As an example, we easily check this relation with $k = 1$, $V_1 = \{a(0), b(0), c(0), a(1), b(1), c(1)\}$ and $\mathcal{H}_1 = \{\neg\phi^g_1, \neg\phi^f_1\}$ by existentially quantifying over ϕ^g_0 and ϕ^f_0.

Our technique has been implemented within NuSMV [7], a state-of-the-art symbolic model checker used both to verify industrial designs of hardware and software systems and to test new formal verification techniques. NuSMV that integrates BDD-based and SAT-based model checking techniques. We modified the encoder/decoder modules according to Definition 3 and the CNF converter. Then, we experimented with several problems from the standard distribution of NuMSV using iSIM, experiencing a remarkable improvement in carrying out both the encoding and the solving task.

Figure 3 presents some results. The "counter" instance (deep counterexample at step 256 over a rather simple model) is particularly interesting as it isolates the contribute of the incremental machinery from the complexity of the underlying theory. The "periodic" and "dme" instances tests true specifications against more complex models (an asynchronous pipeline and a sequential logic network).

6 Related works

Several slightly different notions of incrementality have been proposed for the SAT problem during the last ten years. The first one was introduced by Hooker [14] in 1993. He addressed the problem of deciding whether $g \cup \{\Gamma\}$ is still satisfiable, given a satisfiable clause set g and an additional clause Γ. This basic mechanism is then exploited to decide a formula f, by adding one clause at a time and solving $|f|$ incremental sub-problems. The proposed algorithm is an adaptation of the basic DPLL procedure [9] that

retains the position in the search tree (the path form the root to the last node examined) when a model for f is encountered. Then, if the assignment also satisfies $f \cup \{\Gamma\}$, nothing has to be done. Otherwise, the algorithm adds Γ to the clause set (and the possibly nonempty set $var(\Gamma) \setminus var(f)$ to the variable set), backtracks until Γ stops generating inconsistencies, and finally restarts to visit the search tree. This method was later extended to deal with the addition of multiple clauses at one time [4].

The Hooker's approach was conceived to solve problems arising from logic circuit verification, and, more generally, from problems related to the Electronic Design Automation (EDA). Significant improvements over the original proposal have been recently reported for applications on the same domain [17,18]. These contributions describe a method to simultaneously solve a series of closely related SAT instances which is similar to the Hooker's one but also allows for the removal of sets of clauses. Formally, the proposed technique tackles the following problem [18]. Given a tree $G = (V, E)$ where each node $v \in V$ denotes a set of clauses $C(v)$ and each path from the root to a particular node v_i is associated with the SAT instance $\varphi(v_i) = \bigcup_{0 \leq j \leq i} C(v_j)$, decide the satisfiability of all the instances on the leaves. A former version of this formalization exists [16,17] where only trees of depth one are considered, in so as each formula $f_i := f_C \cup \Delta f_i$ on a leaf just adds some specific set of clauses Δf_i to a shared root subformula f_C. A DPLL-like algorithm is first applied to check the satisfiability of the root. If it is unsatisfiable then all the SAT problems at the leaves are unsatisfiable. Otherwise the algorithm recursively traverses the problem tree and checks the satisfiability of each node. The model of each satisfiable node is used as a starting point for all the child problems. Whenever the algorithm bumps into an unsatisfiable node it concludes that all the instances in the sub-tree rooted at that node are inconsistent and backtracks.

The same authors later proposed SATIRE [24], once more in the framework of EDA verification and optimization problems. SATIRE is a DPLL-like SAT solver that uses an incremental reasoning engine to decide n related SAT instances $\varphi_1, \ldots, \varphi_n$ where $\varphi_{i+1} = (\varphi_i \setminus \rho_i) \cup \alpha_{i+1}$ (ρ_i is the clauses to be removed and α_{i+1} the clauses to be added in order to transform φ_i in φ_{i+1}). As a major contribution this work enlightens the importance of learned clauses in the incremental solving process. SATIRE indeed tries to take advantage of the conflict clauses learned during the solution of the instances $\varphi_1, \ldots, \varphi_i$ while tackling φ_{i+1}. The main issue in reusing learned clauses is that the removal of clauses may clash with the validity of recorded conflict clauses. Whenever a clause belonging to the clause set generating a given conflict clause is removed the conflict clause does't hold any more and has to be removed. The problem is overcome in SATIRE by means of a detailed determination of the relationships between learned clauses and existing constraints performed during the conflict analysis. This mechanism requires an extra computation that can be very time consuming.

SATIRE's authors first enlightened that the reuse of learned clauses could be very effective in the context of the BMC procedure. They indeed experimented with some SAT formulas coming from BMC encodings and showed significant improvements as to solving time. The big potential of sharing learned clauses between similar BMC instances was independently investigated by Shtrichman [23], who already has worked on tuning generic SAT solvers for BMC instances by means of pre-computation of the variable ordering and some form of internal constraints replication to reduce the dimension of the search space [22]. He observed that the sets of learned clauses obtained for consecutive bounds are quite similar, though the additional problem of deciding which conflict clauses maintain validity still arises. The author proposes a DPLL algorithm augmented with a procedure to isolate the reusable conflict clauses. This procedure is based on a careful exploration of the implication graph used to perform the conflict analysis similar to the one introduced in [24] and it suffers from the same disadvantages coming from the additional book-keeping required. Experimental results show that constraints sharing generally has a positive effect on performances, but sometime its overhead overcomes the benefits.

Recently, one SAT solver (SATZOO [11]) has been implemented to incorporate the concept of incremental resolution for highly related SAT instances. It is based on a traditional DPLL-style procedure augmented with an interface which, given two subsequent related SAT instances, allows the second to be specified incrementally from the first by means of adding and removing constraints. The interface lets only unit clauses be removed from the clause database. This way, all the clauses learned during one run may be reused by the search procedure during the subsequent runs because the unit clauses can be considered as assumptions and learned clauses are independent of the assumptions under which they are deduced.

The potential of an incremental resolution of many related instances is so evident that also the state of the art SAT solver Chaff [20] has been integrated in his last release with a module that shows an incremental solving capability. A more detailed description of the used technique can be found in [12].

7 Conclusions and future work

We proposed an integrated approach to incremental satisfiability that allows to *incrementalise* existing CTS procedures. We presented an incremental machinery that mainly retains the simplicity and strength of the original non-incremental one. In particular, we showed how to connect subsequent instances in a chain by relying on a tighter integration with the solver. We unveiled the importance of the deduction theorem and of the incremental property encoding to completely solve the learned-clause problem. We discussed the modifications needed to obtain an incremental solver and pointed out some details of the incremental generation step. We also presented an example of a complete iCTS implementation on top of real-world tools and gave a summary of the research on incremental decision procedures reported so far in the literature.

Our future work goes towards 1) the application of our framework to other domains, 2) a further enlargement of the iSAT solver interface, 3) the integration of *inductive learning* methods within our approach, and 4) some form of *validity checking* obtained by inductively reasoning on the structure of refutations rather than by explicit induction.

References

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic Reachability Analysis Based on SAT-Solvers. In *Proc. of TACAS 2000*, volume 1785, pages 411–425, 2000.
2. M. Benedetti and S. Bernardini. Incremental Compilation-to-SAT Procedures. Technical Report T03-12-13, `sra.itc.it/people/benedetti/TR031213.pdf`, ITC-Irst, 2003.
3. M. Benedetti and A. Cimatti. Bounded Model Checking for Past LTL. In *Proc. of TACAS 2003*, number 2619 in LNCS, pages 18–33, 2003.
4. H. Bennaceur, I. Gouachi, and G. Plateau. An Incremental Branch-and-Bound Method for Satisfiability Problem. *INFORMS Journal on Computing*, 10:301–308, 1998.
5. S. Bernardini. Structure and Satisfiability in Propositional Formulae. *AI*IA Notizie*, 4:46–51, 2003.
6. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proc. of Design Automation Conference*, volume 1579, pages 193–207, 1999.
7. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proc. of CAV 2002*, volume 2404 of *LNCS*, 2002.
8. J. M. Crawford and A. D.Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proc. of 12th AAAI '94*, pages 1092–1097, 1994.
9. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5:394–397, 1962.
10. T. Boy de la Tour. Minimizing the Number of Clauses by Renaming. In *Proc. of the 10th Conference on Automated Deduction*, pages 558–572, 1990.
11. N. Eén and N. Sörensson. Temporal Induction by Incremental SAT Solving. In *Proc. of the First International Workshop on Bounded Model Checking*, 2003.
12. Z. Fu. zChaff. http://ee.princeton.edu/ chaff/zchaff.php, 2003.
13. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating Search Heuristics and Optimization Techniques in Propositional Satisfiability. In *Proc. of IJCAR 2001*, 2001.
14. J.N. Hooker. Solving the Incremental Satisfiability Problem. *Journal of Logic Programming*, 15:177–186, 1993.
15. H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of ECAI 1992*, pages 359–363, 1992.
16. J. Kim, J. Whittemore, J. P. M. Silva, and K. A. Sakallah. Incremental Boolean Satisfiability and its Applications to Delay Faults Testing. In *IEEE/ACM International Workshop on Logic Synthesis*, 1999.
17. J. Kim, J. Whittemore, J. P. M. Silva, and K. A. Sakallah. On Applying Incremental Satisfiability to Delay Fault Problem. In *Proc. of DATE 2000*, pages 380–384, 2000.
18. J. Kim, J. Whittemore, J. P. M. Silva, and K. A. Sakallah. On Solving Stack-Based Incremental Satisfiability Problems. In *Proc. of the ICCD 2000*, pages 379–382, 2000.
19. T. Larrabee. Test pattern generation using boolean satisfiability. In *IEEE Transaction on Computer-aided Design*, pages 4–15, 1992.
20. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. of the 38th DAC*, pages 530–535, 2001.
21. D. A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.
22. O. Shtrichman. Tuning SAT checkers for Bounded Model Checking. In *Proc. of the 12th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science. Springer Verlag, 2000.
23. O. Shtrichman. Pruning Techniques for the SAT-based Bounded Model Checking Problem. In *Proc. of CHARME'01*, pages 58–70, 2001.
24. J. Whittemore, J. Kim, and K. A. Sakallah. SATIRE: A New Incremental Satisfiability Engine. In *Proc. of the 38th Conference on Design Automation*, pages 542–545, 2001.