Chapter preprint of the second edition of the Handbook of Satisfiability (2021)
Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh (Eds.)
Published by IOS Press. The print edition can be ordered at
https://www.iospress.nl/book/handbook-of-satisfiability-2/
The published electronic chapters (Version of Record) are available at
https://ebooks.iospress.nl/ISBN/978-1-64368-161-0/

Chapter 18

Bounded Model Checking

Armin Biere

Besides Equivalence Checking [KK97, KPKG02] the most important industrial application of SAT is currently Bounded Model Checking (BMC) [BCCZ99]. Both techniques are used for *formal* hardware verification in the context of electronic design automation (EDA), but have successfully been applied to many other domains as well. In this chapter, we focus on BMC.

In practice, BMC is mainly used for falsification resp. testing, which is concerned with violations of temporal properties. However, the original paper on BMC [BCCZ99] already discussed extensions that can prove properties. A considerable part of this chapter discusses these complete extensions, which are often called "unbounded" model checking techniques, even though they are build upon the same principles as plain BMC.

Two further related applications, in which BMC becomes more and more important, are automatic test case generation for closing coverage holes, and disproving redundancy in designs. Most of the techniques discussed in this chapter transfer to this more general setting as well, even though our focus is on property verification resp. falsification.

The basic idea of BMC is to represent a counterexample-trace of bounded length symbolically and check the resulting propositional formula with a SAT solver. If the formula is satisfiable and thus the path feasible, a satisfying assignment returned by the SAT solver can be translated into a concrete counterexample trace that shows that the property is violated. Otherwise, the bound is increased and the process repeated. Complete extensions to BMC allow to stop this process at one point, with the conclusion that the property cannot be violated, hopefully before the available resources are exhausted.

18.1. Model Checking

The origins of model checking go back to the seminal papers [CE82] and [QS82]. Clarke, Emerson and Sifakis won the 2007 Turing Award for their pioneering work on model checking. A workshop affiliated to the Federated Conference on Logic in Computer Science (FLOC'06) celebrated the 25th anniversary of model checking. The proceedings [VG08] of this workshop and *the* model checking book

[CGP99] are good starting points to learn about model checking. A more recent survey [PBG05] adds a perspective on SAT-based model checking.

In this chapter, we focus on SAT-based symbolic model checking [McM93], which originally relied on binary decision diagrams (BDDs) [Bry86] to symbolically represent systems. Operations on system states can then be mapped to BDD operations. In practice, BDDs can handle circuits with hundreds of latches, but often blow up in space.

BMC [BCCZ99] was an attempt to replace BDDs with SAT in symbolic model checking. However, SAT lacks the possibility to eliminate variables, which is a key operation in BDD-based model checking. The solution in BMC is to focus on falsification and, at least in a first approximation, drop completeness. This paradigm shift was hard to convey originally, but was accepted at the end, since SAT-based model checking, at least for falsification, scales much better [Kur08].

Another important direction in model checking is explicit state model checking. The SPIN model checker [Hol04] is the most prominent explicit state model checker and is mainly used for checking protocols. It draws its main power from partial order reduction techniques such as [Pel94]. Related techniques exist in BMC as well, see for instance [Hel01, JHN03]. However, for the rest of this chapter we focus on symbolic techniques for synchronous systems, for which partial order techniques do not seem to apply.

The first decade[1] of research in model checking witnessed a heated debate on which specification formalism is more appropriate: linear time logic versus branching time logic. Commonly only computation tree logic (CTL) [CE82], a branching time logic, and linear time logic (LTL) [Pnu77] are used. Originally, LTL was called propositional temporal logic (PTL) as a special case of "full" first-order temporal logic. However, nowadays LTL without further qualification is solely used for propositional linear temporal logic. Also note that PTL is also an acronym for past time (propositional) linear temporal, see for instance [BHJ+06].

LTL is arguably easier to understand and use, but at least in theory, LTL is harder [SC85] to check than CTL. If the system is represented symbolically, there is actually no difference as both problems are PSPACE complete [SC85, Sav70, PBG05]. Specifications in practice are typically in the intersection [Mai00] between LTL and CTL. If we restrict ourself to properties in the intersection, the problem of choosing between LTL and CTL boils down to which model checking algorithm to use. In this respect, BDD-based model checking has a slight bias towards CTL, whereas SAT-based model checking has a bias towards LTL. Thus, we mainly focus on LTL in the rest of this chapter. Further details on temporal logic and its history can be found in [Eme90, Var08].

18.1.1. LTL

As first promoted by Pnueli [Pnu77], temporal logic is an adequate specification formalism for concurrent resp. reactive systems. The syntax of the linear temporal logic LTL contains propositional boolean variables V, temporal operators and the usual propositional operators, including negation $\neg$ and conjunction $\wedge$. Typical

[1] A similar discussion took place in the recent process of standardizing temporal logic in the form of System Verilog Assertions (SVA) and the Property Specification Logic (PSL).

temporal operators are the "next time" operator $\mathbf{X}$, the "finally" operator $\mathbf{F}$, and the "globally" operator $\mathbf{G}$.

Examples of temporal formulas are as follows: $a \rightarrow \mathbf{X}b$ means the property b has to hold in the next time instance, unless a does not hold now. With $\mathbf{X}$ alone only properties about a finite future around the initial state can be specified. The other temporal operators allow to extend this finite view, and specify infinite behavior. The "globally" operator $\mathbf{G}$ allows to specify *safety properties* in form of invariants or assertions that need to hold in all reachable states. For instance, $\mathbf{G}\neg(a \wedge b)$ specifies mutual exclusion of a and b. The only *liveness* operator we consider, the "finally" operator, describes necessary behavior, e.g. $\mathbf{G}(a \rightarrow \mathbf{F}b)$, which requires each a to to be followed by b. More verbosely, the following invariant holds: if a is true then at the same time or later b has to hold, i.e. b cannot be postponed forever, after a has been assured. This is an invariant with a (potentially) liveness condition attached.

The interpretation of propositional variables may change over time but is uniquely determined by the current state of the model. This correspondence is captured via a labelling function $L\colon S \rightarrow \mathbb{P}(V)$, where S is the set of states. A propositional variable p is true in a system state s iff $p \in L(s)$. Beside the set of states S, a model has a set $I \subseteq S$ of initial states, and a transition relation $T \subseteq S \times S$. Such a model is also called *Kripke structure*. Often only models isomorphic to the set of interpretations of the boolean variables V are considered: then $S = \mathbb{P}(V)$ and $L(V') = V'$ for all "states" $s = V' \subseteq V$. A typical example are models of synchronous circuits, where V is the set of latches and input signals, and optionally includes output and internal signals as well. In the following, we fix one Kripke structure $K = (S, I, T, L)$ over the variables V.

The transition relation T is assumed to be total and the set I of initial states to be nonempty. As in the previous example, the transition relation is in general represented symbolically, e.g. as a circuit or a formula. In the following we simply write $T(s, s')$ for this formula, with the interpretation that $T(s, s')$ holds iff there is a transition from s to s', also written as $s \rightarrow s'$. Note that s is simply a vector of all variables V in the current state and s' a vector of their primed copies in the successor state. We use a similar interpretation for $I(s)$.

The semantics of LTL are defined along paths of the model. A path π is an infinite sequence of states $\pi = (s_0, s_1, s_2, \ldots)$, with $s_i \rightarrow s_{i+1}$. A path π is initialized if its first state $\pi(0) = s_0$ is an initial state. In the following, we also use the same notation to refer to single states of a path, e.g. $\pi(i) = s_i$ with $i \in \mathbb{N} = \{0, 1, 2, \ldots\}$. A suffix of a path is defined as $\pi^i = (s_i, s_{i+1}, \ldots)$. We now give a simplified[2] version of the standard (unbounded) semantics, defined recursively over the formula structure. An LTL formula f holds along a path π, written $\pi \models f$, iff

$$\begin{array}{llll}
\pi \models p & \text{iff} & p \in L(\pi(0)) & \pi \models \neg p \quad \text{iff} \quad p \notin L(\pi(0)) \\
\pi \models g \vee h & \text{iff} & \pi \models g \text{ or } \pi \models h & \pi \models g \wedge h \quad \text{iff} \quad \pi \models g \text{ and } \pi \models h \\
\pi \models \mathbf{F}g & \text{iff} & \exists j \in \mathbb{N}\colon \pi^j \models g & \pi \models \mathbf{G}g \quad \text{iff} \quad \forall j \in \mathbb{N}\colon \pi^j \models g \\
\pi \models \mathbf{X}g & \text{iff} & \pi^1 \models g &
\end{array}$$

[2] In particular we do not treat the "until" operator to make the following encoding easier to explain. The full semantics and its encoding can be found in [BHJ+06].

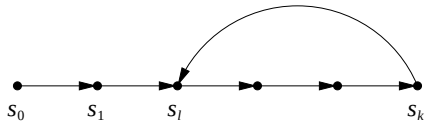

Figure 18.1. A (k,l)-lasso with $k = 5$, $l = 2$.

Here we assume that the formula is in negation normal form (NNF), i.e. negations are pushed down to the variables, with the help of the following axioms of LTL:

$$\neg(g \wedge h) \equiv (\neg g) \vee (\neg h) \qquad \neg\mathbf{F}g \equiv \mathbf{G}\neg g \qquad \neg\mathbf{G}g \equiv \mathbf{F}\neg g \qquad \neg\mathbf{X}g \equiv \mathbf{X}\neg g$$

Finally, a formula f holds in a Kripke structure K, written $K \models f$, iff $\pi \models f$ for all initialized paths π of K. The model checking problem is to determine whether $K \models f$ holds. Related to the model checking problem is the question of the existence of a witness: a formula f has a witness in K iff there is an initialized path π with $\pi \models f$. Clearly $K \models f$ iff $\neg f$ does not have a witness in K. Therefore, we can reduce the model checking problem to the search for witnesses using negation and translation into NNF.

18.2. Bounded Semantics

First observe that some infinite paths can be represented by a finite prefix with a loop: an infinite path π is a (k,l)-lasso, iff $\pi(k+1+j) = \pi(l+j)$, for all $j \in \mathbb{N}$. In this case, π can actually be represented as $\pi = \pi_{\text{stem}} \cdot (\pi_{\text{loop}})^\omega$, as shown in Fig. 18.1.

As LTL enjoys a small model property [LP85], the search for witnesses can be restricted to lassos, if K is finite. See [BCCZ99, CKOS05] for more details. Let us rephrase the unbounded semantics by fixing the path π, but working with different suffixes π^i of π.

$$\begin{array}{llll}
\pi^i \models p & \text{iff} & p \in L(\pi(i)) & \pi^i \models \neg p \quad \text{iff} \quad p \notin L(\pi(i)) \\
\pi^i \models g \vee h & \text{iff} & \pi^i \models g \text{ or } \pi^i \models h & \pi^i \models g \wedge h \quad \text{iff} \quad \pi^i \models g \text{ and } \pi^i \models h \\
\pi^i \models \mathbf{F}g & \text{iff} & \exists j \in \mathbb{N}\colon \pi^{i+j} \models g & \pi^i \models \mathbf{G}g \quad \text{iff} \quad \forall j \in \mathbb{N}\colon \pi^{i+j} \models g \\
\pi^i \models \mathbf{X}g & \text{iff} & \pi^{i+1} \models g &
\end{array}$$

To obtain "bounded semantics" we only look at the first $k+1$ states and let i range over $0 \ldots k$. If π is a (k,l)-lasso then $\pi^{k+1+j} = \pi^{l+j}$ for all $j \in \mathbb{N}$ and we get the following "bounded semantics" for lassos:

$$\begin{array}{lll}
\pi^i \models \mathbf{F}g & \text{iff} & \exists j \in \{\min(i,l), \ldots, k\}\colon \pi^j \models g \\
\pi^i \models \mathbf{G}g & \text{iff} & \forall j \in \{\min(i,l), \ldots, k\}\colon \pi^j \models g
\end{array}$$

$$\pi^i \models \mathbf{X}g \quad \text{iff} \quad \begin{cases} \pi^{i+1} \models g & \text{if } i < k \\ \pi^l \models g & \text{if } i = k \end{cases}$$

For $\mathbf{G}g$ to hold on π^i, the body g has to hold at the current position i and of course at all larger positions j, with $i \le j \le k$. However, if the current position

i is in the loop, then g also has to hold from the loop start up to i. Using the minimum over the current position i and the loop start l covers both cases, no matter whether i is in the loop or still in the stem of π. A similar argument applies to $\mathbf{F}g$.

Now assume that π is *not* a (k,l)-lasso for any l. Then the suffix π^{k+1} of π can have an arbitrary shape. Looking only at the first $k+1$ states of π is in general not enough to determine whether a formula holds along π, e.g. the "bounded semantics" can only be an approximation. Still the following but only sufficient conditions hold:

$$\pi^i \models \mathbf{F}g \quad \text{if} \quad \exists j \in [i \dots k] \colon \pi^j \models g$$
$$\pi^i \models \mathbf{X}g \quad \text{if} \quad \pi^{i+1} \models g \text{ and } i < k$$

These early termination criteria are useful for providing counterexamples for pure safety formulas or to more general specifications with a safety part. For instance, in order to falsify the safety property $\mathbf{G}p$, we need to find a witness for $\mathbf{F}\neg p$. If p does not hold in some initial state s in K, then $k = 0$ is sufficient. All paths starting from s are actually witnesses, even if none of them is a $(0,0)$-loop.

If π is not a (k,l)-lasso for any l and we do not want to examine the suffix beyond the bound k, then we cannot conclude anything about $\pi^k \models \mathbf{X}g$ nor $\pi^i \models \mathbf{G}g$ for any $i < k$. This conservative approximation avoids reporting spurious witnesses. In propositional encodings of potential witnesses, we have to replace such LTL formulas by $\bot$, where $\bot$ ($\top$) represents the boolean constant *false* (*true*).

18.3. Propositional Encodings

The bounded approximations of LTL semantics discussed above consider only the first $k+1$ states of π. This is the key to obtain a propositional encoding of the LTL witness problem into SAT.

Assume that we have a symbolic representation of K and let $s_0, \dots, s_k$ be vectors of copies of the state variables, i.e. for each time frame i, there is one copy V_i of V. Further let p_i denote the copy of p in time frame i. All encodings of the LTL witness problem include model constraints:

$$I(s_0) \wedge T(s_0, s_1) \wedge \dots \wedge T(s_{k-1}, s_k)$$

Looping constraints for $l \in \{0, \dots, k\}$ are added:

$$\lambda_l \rightarrow T(s_k, s_l)$$

We further assume that "at most" one λ_l holds. This cardinality constraint can be encoded with a circuit linear in k, for instance via Tseitin transformation [Tse68] of a BDD for this function.[3]

These constraints are always assumed. In particular, if the propositional encoding is satisfiable, the satisfying assignment can be interpreted as a prefix of

[3]Note that cardinality constraints and particularly at most constraints as in this case are symmetric functions, for which the variable order of the BDD does not matter: all reduced ordered BDDs (ROBDDs) for a specific symmetric function are isomorphic.

an initialized path π of K. If λ_l is assigned to $\top$ then π is a (k,l)-loop. What remains to be encoded are the semantics of LTL, in order to make sure that π extracted from a satisfying assignment is indeed a witness.

18.3.1. Original Encoding

The original encoding [BCCZ99] of the witness problem of LTL into SAT, is a straightforward encoding of the reformulations resp. the bounded semantics. It can be implemented as a recursive procedure that takes as parameters an LTL formula f, a fixed bound k, the loop start l, and the position i. The last two parameters range between 0 and k. Let us denote with ${}_l[f]^i_k$ the resulting propositional formula obtained by encoding f for these parameters. First we make sure, by enforcing the model and looping constraints, that the symbolically represented path is a (k,l)-loop:

$$
\begin{aligned}
{}_l[p]^i_k &\equiv p_i & {}_l[\neg p]^i_k &\equiv \neg p_i \\
{}_l[g \vee h]^i_k &\equiv {}_l[g]^i_k \vee {}_l[h]^i_k & {}_l[g \wedge h]^i_k &\equiv {}_l[g]^i_k \wedge {}_l[h]^i_k \\
{}_l[\mathbf{F}g]^i_k &\equiv \bigvee_{j=\min(l,i)}^{k} {}_l[g]^j_k & {}_l[\mathbf{G}g]^i_k &\equiv \bigwedge_{j=\min(l,i)}^{k} {}_l[g]^j_k \\
{}_l[\mathbf{X}g]^i_k &\equiv {}_l[g]^j_k \text{ with } j = i+1 \text{ if } i < k \text{ else } j = l
\end{aligned}
$$

Encoding witnesses without loops is similar. Let $[f]^i_k$ denote the result of encoding a witness without assuming that it is a (k,l)-loop for some l:

$$
[\mathbf{F}g]^i_k \equiv \bigvee_{j=i}^{k} [g]^j_k \qquad [\mathbf{G}g]^i_k \equiv \bot \qquad [\mathbf{X}g]^i_k \equiv \begin{cases} [g]^{i+1}_k & \text{if } i < k \\ \bot & \text{if } j = k \end{cases}
$$

The other cases are identical to the looping case. The full encoding is as follows:

$$
[f]_k \quad \equiv \quad [f]^0_k \ \vee \bigvee_{l=0}^{k} \lambda_l \wedge {}_l[f]^0_k
$$

The second part handles (k,l)-loops, while the first part makes no assumption whether such a loop exists. With an inductive argument $[f]^0_k \Rightarrow {}_l[f]^0_k$ follows. Therefore, there is no need to guard the left part with $\bigvee_{l=0}^{k} \lambda_l$, as it was originally presented in [BCCZ99].

For fixed k, there are $\Omega(|f| \cdot k^2)$ possible different parameters to ${}_l[f]^i_k$. In addition, an application of an equation introduces $\Omega(k)$ connectives to combine sub-results obtained from recursive calls. Thus, the overall complexity is at least cubic in k and linear in the size of the LTL formula $|f|$. For large k, this is not acceptable.[4]

[4] If the result of the encoding is represented as a circuit, then subformulas in the original encoding can be shared after restructuring the formula. In some cases this may even lead to a linear *circuit* encoding. But for binary temporal operators, including the "until" operator, this is in general not possible anymore.

18.3.2. Linear Encoding

The original encoding of [BCCZ99] presented in the last section is only efficient for simple properties such as $\mathbf{F}p$ or $\mathbf{G}p$, where $p \in V$. More involved specifications with deeply nested temporal operators produce quite some overhead. This is even more problematic for deeply nested binary temporal operators, such as the "until" operator $\mathbf{U}$, which we do not discuss in this chapter.

In symbolic model checking with BDDs, LTL is usually handled by a variant of the tableau construction of [LP85]. The tableau of [LP85] can be interpreted as a generalized Büchi automaton. It is conjuncted with K and a witness becomes a fair path on which all the fairness constraints occur infinitely often. In the context of Büchi automata, a fairness constraint is a set of states, which has to be "hit" by a path infinitely often, in order to be a fair path.

The LTL formula $\mathbf{GF}p$, i.e. infinitely often p, is a generic single fairness constraint with the following optimized encoding:

$$[\mathbf{GF}p]_k \equiv \bigvee_{l=0}^{k} \left(\lambda_l \wedge \bigvee_{i=l}^{k} p_i \right)$$

The formula is quadratic in k. However, it is possible to share common subformulas between different loop starts l. The resulting circuit is linear in k. This linear encoding of fairness constraints can be extended to multiple fairness constraints easily. Our first implementation of a bounded model checker used this technique in order to handle hundreds of fairness constraints [BCCZ99].

The tableau construction has symbolic variants [BCM+92, CGH97] as well as explicit variants [WVS83, VW94]. An explicit construction may explode in space immediately, since the Büchi automaton can be exponentially large in the size of the original LTL formula. This is rather unfortunate for BMC, but see [CKOS05] for a discussion on advantages and disadvantages of using an explicit automaton construction for BMC.

However, also symbolic tableau constructions – even with the presented optimized fairness encoding – require witnesses to be (k, l)-loops. This may prohibit early termination and requires larger bounds than necessary.

An improved but still quadratic encoding can be found in [FSW02]. A simpler and linear encoding was presented in [LBHJ04], which we explain next. A survey on the problem of encoding LTL (including past time LTL) can be found in [BHJ+06]. Another advanced encoding for weak alternating Büchi automata was presented in [HJK+06]. This encoding allows to handle all ω-regular properties which is a super set of LTL. All these symbolic encodings avoid the exponential blow-up of an explicit tableau construction and also allow to terminate earlier.

Before we explain the linear encoding, we first give an example why a simple recursive formulation is incorrect. Assume we have a single variable p and the following symbolic representation of a Kripke structure:

$$I(s) \equiv \overline{p} \qquad\qquad T(s, s') \equiv (s' = s)$$

The state space is $S = \{\bot, \top\}$, e.g. consists of all the valuations of p, see also Fig. 18.2. Only one state is reachable in K on which p does not hold. The

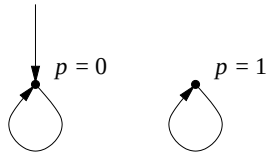

Figure 18.2. Kripke structure for counterexample to naive recursive encoding.

LTL formula $\mathbf{F}p$ can be characterized using the following least fixpoint equation: $\mathbf{F}p \equiv p \vee \mathbf{XF}p$. Therefore, it is tempting to introduce a new boolean variable to encode $\mathbf{F}p$, let us call it a, and encode this fixpoint equation for $k = 0$ as $a_0 \leftrightarrow (p_0 \vee a_0)$. Note that for $k = 0$, the only possible l, the target of the back loop, is $l = 0$ again. Model constraints, actually just the initial state constraint, together with this encoding, result in the following formula:

$$\overline{p}_0 \wedge \lambda_0 \wedge a_0 \wedge (a_0 \leftrightarrow (p_0 \vee a_0))$$

This formula seems to follow the semantics, e.g. the model constraints are enforced and the fixpoint characterization of $\mathbf{F}p$ is encoded. However, it is *satisfiable* by setting a_0 to $\top$. This should not be the case, since $\mathbf{F}p$ does *not* hold on the single possible path in which p never holds. The problem is that even though this recursive encoding of the LTL formula captures the intention of the fixpoint equation it ignores *least* fixpoint semantics. A similar problem occurs when handling stable models for logic programs [SNS02] or more general in the context of answer set programming (ASP) [Nie99], where the default semantics of recursive properties are defined through least fixpoints. This observation can actually be used positively, in order to succinctly encode bounded witness problem of LTL into ASP [HN03] instead into SAT.

To summarize the example, a naive encoding results in an equation system with cyclic dependencies. A solution to such a system is an arbitrary fixpoint. However, semantics of $\mathbf{F}g$ require a least fixpoint.

The basic idea of the linear encoding in [LBHJ04] is to use several iterations through the fixpoint equations of the LTL subformulas with top most $\mathbf{F}$ operator until the values do not change anymore. As it turns out, two backward iterations are actually enough. For each iteration, the value of an LTL formula at time frame i is encoded with the help of a new boolean variable. This is actually similar to using Tseitin variables [Tse68] to encode propositional formulas of arbitrary structure into CNF.

The variables for the first (inner resp. nested) iteration are written $\langle f \rangle_k^i$, those for the second (outer) iteration $\{f\}_k^i$. The rest of the encoding relates these variables among different subformulas, iterations and time points. The full set of constraints is shown in Fig. 18.3. The correctness of this encoding is established by the following theorem, which is an instance of a more general theorem proved in [BHJ+06]:

Theorem 1. *Let f be an LTL formula. If $\{f\}_k^0$ is satisfiable assuming in addition model and looping constraints, then there is a path π with $\pi \models f$.*

$$
\begin{aligned}
\{p\}_k^i &\equiv p_i \\
\{\neg p\}_k^i &\equiv \neg p_i \\
\\
\{g \vee h\}_k^i &\equiv \{g\}_k^i \vee \{h\}_k^i \\
\{g \wedge h\}_k^i &\equiv \{g\}_k^i \wedge \{h\}_k^i
\end{aligned}
$$

$$
\begin{aligned}
\langle \mathbf{F} g \rangle_k^i &\equiv \{g\}_k^i \vee \langle \mathbf{F} g \rangle_k^{i+1} & \{\mathbf{F} g\}_k^i &\equiv \{g\}_k^i \vee \{\mathbf{F} g\}_k^{i+1} && \text{if } i < k \\
\langle \mathbf{F} g \rangle_k^i &\equiv \{g\}_k^i & \{\mathbf{F} g\}_k^i &\equiv \{g\}_k^i \vee \textstyle\bigvee_{l=0}^k (\lambda_l \wedge \langle \mathbf{F} g \rangle_k^l) && \text{if } i = k \\
\\
\langle \mathbf{G} g \rangle_k^i &\equiv \{g\}_k^i \wedge \langle \mathbf{G} g \rangle_k^{i+1} & \{\mathbf{G} g\}_k^i &\equiv \{g\}_k^i \wedge \{\mathbf{G} g\}_k^{i+1} && \text{if } i < k \\
\langle \mathbf{G} g \rangle_k^i &\equiv \{g\}_k^i & \{\mathbf{G} g\}_k^i &\equiv \{g\}_k^i \wedge \textstyle\bigvee_{l=0}^k (\lambda_l \wedge \langle \mathbf{G} g \rangle_k^l) && \text{if } i = k \\
\\
& & \{\mathbf{X} g\}_k^i &\equiv \{g\}_k^{i+1} && \text{if } i < k \\
& & \{\mathbf{X} g\}_k^i &\equiv \textstyle\bigvee_{l=0}^k (\lambda_l \wedge \{g\}_k^l) && \text{if } i = k
\end{aligned}
$$

Figure 18.3. LTL constraints for the linear encoding of LTL into SAT.

For $i < k$ and for each subformula, there are at most k connectives in the outer iteration. For $i = k$, we need $2 \cdot (k + 1)$ binary disjunctions resp. conjunctions. The inner iteration adds k more. Altogether, the encoding is linear in k and the formula size $|f|$.

The previous example shows that one iteration is incorrect, at least for $\mathbf{F}g$. It is interesting to note that encoding the fixpoint equation $\mathbf{G}g \equiv g \wedge \mathbf{X}\mathbf{G}g$ does not suffer from the same problem, due to greatest fixpoint semantics and monotonicity of the encoding. Therefore, it is possible to only use one iteration for $\mathbf{G}$ as it is also clearly the case for the propositional operators and $\mathbf{X}$, for which we applied this optimization already in Fig. 18.3. More variants, extensions, proofs and experimental results can be found in [BHJ+06].

Also note that the "no-lasso case" is also captured by this encoding: The result is satisfiable if there is a finite prefix with $k+1$ states, such that all infinite paths with this prefix are witnesses.

Another option is to use the "liveness to safety" translation of [BAS02, SB04] which modifies the model, but also will result in a linear encoding of certain LTL formulas and in particularly in a linear encoding of fairness constraints.

18.4. Completeness

The encodings of the previous section allow to find witnesses for a particular bound k. If the resulting propositional formula turns out to be satisfiable, we are sure that we have found a witness. If the resulting formula is unsatisfiable we can increase k and search for a longer witness. If the LTL formula has a witness this process will find it. However, if it does not, when should we stop increasing k?

In this section, we discuss techniques that allow to *terminate* the search with the conclusion that no witness can be found. We focus on the special case of simple safety properties $\mathbf{G}p$, e.g. when searching for a witness of $\mathbf{F}\neg p$.

A first answer was given in the original paper [BCCZ99]. From graph theory we can borrow the concept of diameter, which is the longest shortest path between two nodes resp. states, or more intuitively the maximum distance between connected states. It is also often called eccentricity. If a bad state is reachable, then it is reachable in a shortest path from an initial state, which has length smaller or equal than the diameter.

A number such as the diameter, which allows us to stop BMC and conclude, that no witness can be found, is called completeness threshold (CT) [CKOS04, CKOS05]. Another trivial but in practice almost useless completeness threshold is $|S|$, the number of states.

Furthermore, since a shortest witness for $\mathbf{F}\neg p$ always starts with an initial state and ends in a bad state, where p does not hold, we can also use the following distances as CT: either the largest distance of any reachable state from an initial state or if it is smaller the largest distance of any state to the set of bad states, if it can reach a bad state. The former is also referred to as radius, more specifically as forward radius, the latter as backward radius with respect to the set of bad states.

In practice, the backward radius is often quite small. For instance, if p is inductive ($p_0 \wedge T(s_0, s_1) \Rightarrow p_1$) then the backward radius is 0, because is is impossible to go from a state in which p holds to a state in which p does not hold.

Unfortunately, computing diameters directly is quite hard. It is probably as hard as solving the witness problem in the first place. But there are further weaker CTs, which still are often small enough. The first example is the reoccurrence diameter of [BCCZ99], which is the length of the longest simple path in K. A simple path is another concept borrowed from graph theory and denotes a path, on which all states are different.

The reoccurrence diameter can be arbitrarily larger than the (real) diameter. Consider as example a fully connected graph with n nodes. We can easily generate a simple path of length n without reoccurring state. Since every state is reachable from any other step in one step due to full connectivity, the diameter is 1.

Analogously to the diameter, the forward and backward reoccurrence radii with their obvious definitions are CTs as well. Typically, the forward reoccurrence radius is way too large to be of any practical value, while again there exist many examples where the backward reoccurrence radius is small enough to obtain termination.

The main reason to work with reoccurrence diameters instead of real diameters is the possibility to formulate the former in SAT, while the latter is conjectured to need QBF. A simple path constraint to ensure unique states on a path is as follows:

$$\bigwedge_{0 \leq i < j \leq k} s_i \neq s_j$$

This formulation is quadratic in k. There are multiple solutions to avoid this quadratic overhead. One proposal [KS03, CKOS05] uses hardware implementations of sorting networks, which gives an $O(n \cdot \log^2 n)$ sized encoding, but the complexity of these networks usually results in slower SAT solving times [JB07]. This last paper [JB07] also describes how QBF can be used to encode simple path constraints. Similar result are reported in [DHK05, MVS+07].

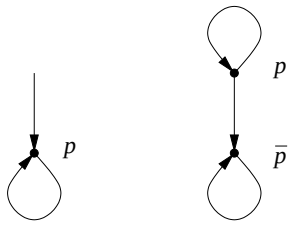

Figure 18.4. Example that k-induction really needs simple path constraints for completeness. The state on the left is the only reachable state, while the two states on the right are not reachable. There exists arbitrary long paths of length k, where p holds on the first k states but not on the last state. However, if these paths are assumed to be simple paths, then the induction step for k-induction becomes unsatisfiable for $k = 2$.

The currently most effective solution is given in [ES03]. The basic idea is to start without any simple path constraints, but then add those inequalities to the SAT solver, which are violated in an assignment returned by the SAT solver. Then the SAT solver is restarted. In practice, the number of incrementally added inequalities is usually very small and even large bounds with many state variables can be handled this way.

18.5. Induction

An important step towards complete BMC techniques was k-induction [SSS00], which also in practice is still quite useful. The basic idea is to strengthen the property p, for which we want to show that $\mathbf{G}p$ holds, by adding more predecessor states. The base case for k-induction is a simple bounded model checking problem:

$$I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge \neg p_k$$

If the base case is satisfiable, a witness has been found. Otherwise the induction step is checked:

$$p_0 \wedge T(s_0, s_1) \wedge p_1 \wedge T(s_1, s_2) \wedge \ldots \wedge p_{k-1} \wedge T(s_{k-1}, s_k) \wedge \neg p_k$$

This is almost a BMC problem, except that the initial state constraint is removed and p is assumed to hold on all states except for the last state. We start with the induction step for $k = 0$, which simply checks whether $\neg p_0$ is unsatisfiable as a propositional formula without any assumptions about the state. If the formula is indeed unsatisfiable, then $\mathbf{G}p$ holds trivially.

Then we check the base case for $k = 0$. If $I(s_0) \wedge \neg p_0$ is satisfiable, then p can be already violated in an initial state. Otherwise we move on to the next induction step at $k = 1$, which is $p_0 \wedge T(s_0, s_1) \wedge \neg p_1$. If this formula is unsatisfiable, we have actually proven that p is inductive for T and again $\mathbf{G}p$ holds.

These two special cases of early termination for BMC, stateless validity and inductiveness, were also discussed in [BCRZ99], but k-induction is able to increase k further and may terminate BMC even with $k > 1$. As the example in

Fig. 18.4 shows this method is not complete. Adding simple path constraints to the induction steps makes it complete, using the argument that the reoccurrence diameter is a CT for plain BMC.

Techniques such as invariant strengthening, discussed further down in Section 18.8, can help to reduce the bound until which k-induction with simple path constraints has to be carried out.

The base case and the induction step share large parts of subformulas even when increasing k to $k+1$.[5] To avoid copying these parts and to reuse learned clauses another important technique in this context is the usage of incremental SAT solvers [KWS00, WKS01, ES03]. It is even possible to actively copy learned clauses between time frames as suggested by [Str01]. Variants of k-induction for more general properties are also discussed in [AS06b, HJL05].

18.6. Interpolation

Before Aaron Bradley introduced IC3 [Bra11], model checking based on interpolation [Cra57] was considered to be the most efficient and robust model checking technique, as for instance the comparison in [ADK+05] showed. Nowadays IC3 [Bra11] and its variant PDR [EMB11] replaced interpolation based model checking in this regard [CLP+14, BvDH17]. Nevertheless interpolation found many applications beside model checking and is still useful in a portfolio approach to model checking. For more information on IC3 see [BK18].

The idea to use interpolation in model checking goes back to Ken McMillan [McM03]. The key idea is to extract an interpolant from a resolution proof for a failed BMC run and use the interpolant as over-approximation for image computation. The extraction algorithm was independently discovered by McMillan, but is very similar to those described in [Kra97, Pud97]. In this section, we present this extraction algorithm and provide a simple straight-forward proof for the propositional case. Our proof is inspired by [McM05] but stripped down to the propositional case. Further usage of interpolation is discussed in Chapter 33 on SMT and in Chapter 20 on Software Verification.

Let A, B be formulas in CNF, c, d clauses, and f, g propositional formulas. With $V(h)$ we denote the set of variables occurring in a formula h. A variable is *global* if it occurs both in A and in B. Let $G = V(A) \cap V(B)$ denote the global set of variables of A and B. A variable is called *local* to A if it only occurs in A and thus is not global. According to [Cra57] a formula f is an *interpolant* for A with respect to B iff it only contains global variables and

$$\text{(I1)} \quad A \Rightarrow f \qquad \text{and} \qquad \text{(I2)} \quad B \wedge f \Rightarrow \bot$$

We consider as proof objects *interpolating quadruples* of the form $(A, B)\ c\ [\, f\,]$, where the clause c is called the *resolvent* and f the *preliminary interpolant*. Then an interpolating quadruple is *well formed* iff

$$\text{(W1)} \quad V(c) \subseteq V(A) \cup V(B) \qquad \text{and} \qquad \text{(W2)} \quad V(f) \subseteq G \cup (V(c) \cap V(A)) \subseteq V(A)$$

[5] Note that p_i can also be assumed in the base case for $i = 0 \ldots k-1$, which would actually be learned by the SAT solver from the previous unsatisfiable base case anyhow.

Thus, an interpolating quadruple is well formed iff the resolvent c is a clause made of literals over variables from A and B, and f only contains variables from A. If in addition, a variable local to A occurs in f, then it also has to occur in c. In approximation to (I1) and (I2) a well formed interpolating quadruple is *valid* iff

$$\text{(V1)} \quad A \Rightarrow f \qquad \text{and} \qquad \text{(V2)} \quad B \wedge f \Rightarrow c$$

Note that $A \wedge B \Rightarrow c$ follows immediately from (V1) and (V2). Thus, if a well formed and valid interpolating quadruple with the empty clause $\bot$ as resolvent can be derived, then the preliminary interpolant f of this quadruple actually turns out to be an interpolant of A with respect to B, in particular, $A \wedge B$ is unsatisfiable.

We present tableau rules for deriving well formed and valid interpolating quadruples. This calculus can be interpreted as an annotation mechanism for resolution proofs over $A \wedge B$. It annotates clauses and resolvents with valid interpolating quadruples. Let $c \dot{\vee} l$ denote a clause made up of a subclause c and a literal l, such that $|l|$ does not occur in c, i.e. neither positively nor negatively. The variable of a literal l is written as $|l|$. The proof rules are as follows:

$$\text{(R1)} \quad \frac{}{(A,B)\ c\ [\,c\,]}\ c \in A \qquad \frac{(A,B)\ c \dot{\vee} l\ [\,f\,] \quad (A,B)\ d \dot{\vee} \bar{l}\ [\,g\,]}{(A,B)\ c \vee d\ [\,f \wedge g\,]}\ |l| \in V(B) \quad \text{(R3)}$$

$$\text{(R2)} \quad \frac{}{(A,B)\ c\ [\,\top\,]}\ c \in B \qquad \frac{(A,B)\ c \dot{\vee} l\ [\,f\,] \quad (A,B)\ d \dot{\vee} \bar{l}\ [\,g\,]}{(A,B)\ c \vee d\ [\,f|_{\bar{l}} \vee g|_{l}\,]}\ |l| \notin V(B) \quad \text{(R4)}$$

The notation $f|_{\bar{l}}$ denotes the *cofactor* of f with respect to $\bar{l}$, which is a copy of f, in which occurrences of l are replaced by $\bot$ and occurrences of $\bar{l}$ by $\top$.

This set of rules simulates the algorithm by McMillan described in [McM03] to extract an interpolant from a resolution refutation, with the exception of rule (R1), the base case for clauses from A. The original algorithm removes variables local to A from the preliminary interpolant immediately. In our first approximation to the algorithm, as given by the tableau rules, we delay the removal until variables local to A are resolved away in (R4), in order to be table to apply an inductive proof argument, i.e. (V2). If one is only interested in the final interpolant of a refutation, where the final resolvent is an empty clause, then it is obviously correct to follow the original algorithm and remove the local variables immediately, since they will be resolved away anyhow. The latter is also the approach taken in [Pud97, YM05], but does not allow to maintain (V1) and (V2).

Theorem 2. *The four rules (R1) – (R4) preserve well formedness and validity.*

Proof. The consequents of the two base case rules (R1) and (R2) are clearly well formed, i.e. (W1) and (W2), and validity, i.e. (V1) and (V2), is easily checked as well. In the inductive case, (W1) also follows immediately. Regarding rules (R3) and (R4), we can assume the antecedents to be well formed and valid.

Let us consider rule (R3) next. The formulas f and g and thus also $f \wedge g$ only contain variables from A. Any variable v of $f \wedge g$ that is local to A is different from $|l|$, since the latter is a variable from B. Therefore, v occurs in c or d and

thus in $c \vee d$, which concludes (W2). The first part (V1) of validity follows from the assumptions, i.e. $A \Rightarrow f$ and $A \Rightarrow g$ obviously imply $A \Rightarrow f \wedge g$. To show the second remaining part (V2) of validity we use soundness of resolution:

$$B \wedge (f \wedge g) \quad \Rightarrow \quad (B \wedge f) \wedge (B \wedge g) \quad \Rightarrow \quad (c \vee l) \wedge (d \vee \overline{l}) \quad \Rightarrow \quad (c \vee d)$$

Proving the consequent of rule (R4) to be well formed, i.e. (W2), is simple: The variable $|l|$ is removed both from the resolvent as well as from the preliminary interpolant of the consequent. In order to show (V1) we can assume $A \Rightarrow f$ and $A \Rightarrow g$. Any assignment σ satisfying A evaluates both formulas $A \rightarrow f$ and $A \rightarrow g$ to $\top$. Further assume $\sigma(A) = \sigma(l) = \top$. Then Shannon Expansion for g gives $\sigma(g) = \sigma(l \wedge g|_l \vee \overline{l} \wedge g|_{\overline{l}}) = \sigma(g|_l) = \top$. The other case $\sigma(A) = \sigma(\overline{l}) = \top$, with a similar argument, results in $\sigma(f|_{\overline{l}}) = \top$. Thus, $\sigma(f|_{\overline{l}} \vee g|_l) = \top$ for any satisfying assignment σ of A, which concludes part (V1).

The last case (V2) in the proof of the validity of the consequent of rule (R4) is proven as follows. In the assumption $B \wedge f \Rightarrow c \dot{\vee} l$ we can replace every occurrence of l by $\bot$ respectively every occurrence of $\overline{l}$ by $\top$ without making the assumption invalid. This implies $B \wedge f|_{\overline{l}} \Rightarrow c$, since $|l|$ does not occur in B, nor in c. With a similar argument we obtain $B \wedge g|_l \Rightarrow d$ and thus $B \wedge (f|_{\overline{l}} \vee g|_l) \Rightarrow (c \vee d)$. This completes the proof of the validity of the consequent of rule (R4) assuming validity of its antecedents and concludes the whole proof. □

The extended version of [YM05], which was published as a technical report [YM04] proves a variant of the algorithm given in [Pud97]. Their inductive argument is slightly more complicated than ours, i.e. (V1) and (V2). In addition, our formulation allows to derive an relation between A, B, c and its preliminary interpolant f. In particular, our preliminary interpolant f captures enough information from A, such that B together with f implies c.

The strongest interpolant of A is obtained from A by existentially quantifying over all local variables in A. Thus, interpolation can be seen as an over approximation of quantifier elimination. Consider the following example, where A contains the four clauses A_0 to A_3

$$A_0 : a \vee c \vee d \qquad A_1 : \overline{a} \vee c \vee d \qquad A_2 : a \vee \overline{c} \vee \overline{d} \qquad A_3 : \overline{a} \vee \overline{c} \vee \overline{d}$$

and B the following four clauses B_0 to B_3:

$$B_0 : b \vee \overline{c} \qquad B_1 : \overline{b} \vee \overline{c} \qquad B_2 : b \vee \overline{d} \qquad B_3 : \overline{b} \vee \overline{d}$$

Variable a is local to A, while b is local to B, and the other variables c and d are global. Quantifying a from A results in $\exists a[A] \equiv c \oplus d$, where "$\oplus$" is the XOR operator, i.e. c and d have different value. Since quantifying b from B results in $\exists b[B] \equiv \overline{c} \wedge \overline{d}$, i.e. c and d both have to be $\bot$ and are thus forced to the same

value, the CNF $A \wedge B$ is unsatisfiable. A possible refutation proof is as follows:

$$\begin{array}{lllll}
Q_0: & (A,B) & c \vee d\ [c \vee d] & & \text{resolved from } A_0 \text{ and } A_1 \\
Q_1: & (A,B) & \overline{c}\ \ [\top] & & \text{resolved from } B_0 \text{ and } B_1 \\
Q_2: & (A,B) & d\ \ [c \vee d] & & \text{resolved from } Q_0 \text{ and } Q_1 \\
Q_3: & (A,B) & \overline{d}\ \ [\top] & & \text{resolved from } B_2 \text{ and } B_3 \\
Q_4: & (A,B) & \bot\ \ [c \vee d] & & \text{resolved from } Q_2 \text{ and } Q_3
\end{array}$$

In brackets, we list the partial interpolants. The final interpolant of A with respect to B is $P \equiv c \vee d$, which is weaker than the strongest interpolant $\exists a[A]$, i.e. $\exists a[A] \Rightarrow P$, but it exactly captures the part of A which is required for the refutation: either c or d has to be $\top$.

18.7. Completeness with Interpolation

It has been shown in [McM03] that termination checks for bounded model checking do not need exact quantifier elimination algorithms as has been previously assumed [WBCG00, ABE00, AB02, MS03, McM02, PBZ03]. An over approximation as given by an interpolant extracted from a refutation of a failed bounded model checking run is enough.

The resulting method of [McM03], which we describe in this section, is originally restricted to simple safety properties. But using the "liveness to safety" translation of [BAS02, SB04] it can be applied to more general specifications, which also is reported to work reasonably well in practice.

Assume that we want to prove that p is not reachable, i.e. the property $\mathbf{G}p$ holds, or respectively there is no witness for $\mathbf{F}\neg p$. Let k be the backward radius with respect to $\neg p$. More precisely let k be the smallest number such that all states which can reach $\neg p$, can reach a state in which $\neg p$ holds in k' steps, where $k' \leq k$. The backward radius can be interpreted as the number of generations in backward breadth first search (BFS), starting with the set of bad states in which $\neg p$ holds until all states that can reach a bad state are found. In BDD-based model checking [McM93] this is the number of *preimage* computations until a fixpoint is reached, starting with the set of bad states.

If the property holds and no witness to its negation exists, then the following formula is unsatisfiable (after checking $I(s_0) \Rightarrow p_0$ separately):

$$\underbrace{I(s_0) \wedge T(s_0, s_1)}_{A} \ \wedge \ \underbrace{T(s_1, s_2) \wedge \ldots \wedge T(s_k, s_{k+1}) \wedge \bigvee_{j=1}^{k+1} \neg p_j}_{B}$$

Note that we "unroll" one step further as usual. Let P_1 be the interpolant of A with respect to B. Since $P_1 \wedge B$ is unsatisfiable, all states that satisfy P_1 cannot reach a bad state in k or fewer steps. As a generalization consider the following

sequence of formulas:

$$F_i \quad : \quad \underbrace{R_i(s_0) \wedge T(s_0, s_1)}_{A_i} \wedge \underbrace{T(s_1, s_2) \wedge \ldots \wedge T(s_k, s_{k+1}) \wedge \bigvee_{j=1}^{k+1} \neg p_j}_{B_i}$$

These are all BMC problems with the same bound $k+1$. In the base case, let $P_0 = R_0 = I$. In order to define R_i, under the assumption that F_i is unsatisfiable, let P_{i+1} be the interpolant of A_i with respect to B_i and $R_{i+1}(s) = R_i(s) \vee P_{i+1}[s_0/s_1]$, where $P_{i+1}[s_0/s_1]$ is obtained from P_{i+1} by replacing s_1 with s_0. If F_i becomes satisfiable, then P_j, R_j and F_j are undefined for all $j > i$.

With the same argument as in the base case, we can show that as long P_i is defined all states satisfying R_i cannot reach a bad state in k or less steps, or to phrase it differently: R_i-states are more than k steps away from bad states.

Now let use assume that there exists a smallest i for which F_i is satisfiable. Then there is a state s_0 which reaches a bad state in $k+1$ or less steps but does not reach a bad state in k or less steps. The former just immediately follows from F_i being satisfied, the latter from s_0 satisfying R_i and thus s_0 being at least $k+1$ steps away from bad states. However, this contradicts our assumption that k is the backward radius and there *are* no states that need more than k steps to reach a bad state. Thus, P_i, R_i and F_i are defined for all $i \in \mathbb{N}$.

In addition, since $R_i \Rightarrow R_{i+1}$, we have an increasing chain of weaker and weaker starting points, which for a finite model has to reach a fixpoint. Therefore, there is an n for which $R_n \equiv R_{n+j}$ for all $j \in \mathbb{N}$. As soon $P_{n+1} \Rightarrow R_n$, which can be checked by a SAT solver, R_n is inductive, is implied by $I(s_0)$ and is stronger than p. Therefore, we can stop and conclude $\mathbf{G}p$ to hold resp. that no witness to $\mathbf{F}\neg p$ exists. This conclusion is correct even if k is smaller than the backward radius and all F_i are defined, i.e. if there exists an n for which $P_{n+1} \Rightarrow R_n$.

However, if k is smaller than the backward radius it may happen that F_i is satisfiable for $i > 0$. In this case, we simply increase k and start a new sequence of F_j's. In any case, if F_0 ever turns out to be satisfiable, then we have found a witness, which is a counterexample to $\mathbf{G}p$.

To implement this algorithm, a SAT solver is required which is able to generate resolution proofs. This feature is easy to implement on top of DPLL style solvers, particularly for those that use learning. The overhead to produce resolution proofs is in general acceptable [ZM03, Gel07]. However, interpolants tend to be highly redundant [McM03]. In practice, it is necessary to shrink their size with circuit optimization techniques such as SAT sweeping [Kue04] and AIG rewriting [MCB06].

The outer loop increases k until either the BMC problem becomes satisfiable or the inner loop terminates because $P_{n+1} \Rightarrow R_n$ holds. As we have shown, k is bounded by the backward radius and thus it is beneficial to strengthen p as much as possible to decrease the backward radius which in order reduces not only the number of iterations of the outer loop but also the size (at least the length) of the BMC problems.

18.8. Invariant Strengthening

If a property p is inductive ($p_0 \wedge T(s_0, s_1) \Rightarrow p_1$), then a BMC run for $k = 1$ without initial state constraints is unsatisfiable, and proves that p holds in all states, i.e. $K \models \mathbf{G}p$, unless p is violated in the initial state.

In general, it is difficult to come up with strong enough inductive invariants. However, even if p does not imply the real invariant q, which we want to prove, p's inductiveness can still be used to strengthen q. Then we can try to prove $\mathbf{G}(p \wedge q)$ instead of $\mathbf{G}q$. The former often has a smaller backward radius, in particular the backward radius never increases after strengthening, and can help to terminate k-induction and interpolation earlier. This is particularly useful for k-induction, which suffers from an exponential gap between backward reoccurrence radius and real backward radius.

In the context of sequential equivalence checking, useful invariants are of course equalities between signals. If such an equality $\mathbf{G}(p = q)$ between two signals p and q is suspected, then we can try to check whether $p = q$ is inductive. This idea can be extended to multiple signal pairs, e.g. $\mathbf{G} \bigwedge_{j=1}^{n} (p^j = q^j)$. In this case, inductiveness is proven if the following SAT problems for $m = 1 \ldots n$ are all unsatisfiable:

$$\bigwedge_{j=1}^{n} (p_0^j = q_0^j) \;\wedge\; T(s_0, s_1) \;\wedge\; (p_1^m \neq p_1^m)$$

This idea is described in [vE98] and has been extended to property checking [BC00, CNQ07, AS06a, BM07] and can also make use of k-induction (see Section 18.5). Related to adding invariants is target enlargement [BKA02, BBC+05], which increases the set of bad resp. target states by some states that provably can reach a bad state.

18.9. Related Work

In the late 90ties, the performance of SAT solvers increased considerably [MSS99, Bor97, Zha97]. At the same time progress in BDDs stalled. It became clear that BDD-based symbolic model checking cannot handle more than a couple of hundred latches, which is much smaller than what most industrial applications require. This was the main motivation behind trying to apply SAT technology to model checking. The first angle of attack was to use QBF solvers, because these allow to solve the same problem as BDD-based model checking. However, at that time, QBF solvers were lagging behind SAT solvers. The first real implementations just started to emerge [CGS98]. Therefore, a paradigm shift was necessary.

The development of BMC was influenced by SAT-based planning [KS92]. See also Chapter 19 in this handbook, which is devoted to SAT-based planning. For simple safety properties, the tasks are similar: try to find a bounded witness, e.g. a *plan*, which reaches the goal resp. the bad state. The main contribution of BMC was to show how this idea can be lifted to infinite paths and thus produce witnesses for arbitrary temporal formulas. Furthermore initial attempts were

made to prove properties. Of course, the major difficulty was to convince the formal verification community that a focus on falsification can be beneficial.

Deciding QBF plays the same role for PSPACE-hard problems as SAT does for NP hard problems. Since symbolic model checking is PSPACE complete as well [Sav70], see [PBG05] for more details, it seems natural to use QBF solvers for symbol model checking, as it was already proposed in the original BMC paper [BCCZ99]. However, even though QBF solving is improving, there are very few successful applications of QBF to symbolic model checking [DHK05, CKS07, MVS+07]. Most results are negative [JB07].

Often properties are local and can be proven locally. A standard technique in this context is automatic abstraction refinement. Initially, the model checker abstracts the system by removing all the model constraints on variables apart from those that directly occur in the property. This abstraction is conservative in the following sense: if the property holds in the abstracted model, then it also holds in the concrete model. If it does not hold and the abstract counterexample cannot be mapped to the concrete model the abstraction has to be refined by adding back variables and model constraints. There are various techniques that use BMC in this context. They either use proofs and unsatisfiable cores [MA03] or follow the counterexample-guided abstraction refinement (CEGAR) paradigm [CGJ+03]. For further details, see [PBG05] and particularly Chapter 20. We also skipped most material on circuit based techniques, including quantifier elimination [WBCG00, ABE00, AB02, McM02, PBZ03, PSD06] circuit cofactoring [GGA04] and ATPG-based techniques, which are also discussed in [PBG05].

Finally, BMC has been extended to more general models, including software as discussed in Chapter 20. BMC is used for infinite systems [dMRS03], more specifically for hybrid systems [ABCS05, FH05]. In this context, bounded semantics are typically decidable, while the general model checking problem is not. Nevertheless, complete techniques, such as k-induction and interpolation can still be useful and allow to occasionally prove properties.

18.10. Conclusion

The main reason behind the success of BMC, is the tremendous increase in reasoning power of recent SAT solvers, particularly the breakthrough realized by Chaff [MMZ+01] right two years after the first publication on BMC. SAT and BMC became a standard tool in the EDA industry thereafter. Their importance will be emphasized as SAT solver's capacity continues to increase.

Acknowledgements

The author would like to thank Keijo Heljanko, Viktor Schuppan and Daniel Kröning for their very valuable comments on drafts of this chapter for the first edition of the handbook.

References

[AB02] A. Ayari and D. A. Basin. QUBOS: Deciding quantified boolean logic using propositional satisfiability solvers. In M. Aagaard and J. W. O'Leary, editors, *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD'02)*, volume 2517 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2002.

[ABCS05] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying industrial hybrid systems with MathSAT. *Electronic Notes in Theoretical Computer Science*, 119(2):17–32, 2005. In *Proceedings of the 2nd International Workshop on Bounded Model Checking (BMC'04)*.

[ABE00] P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic Reachability Analysis Based on SAT-Solvers. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2000.

[ADK+05] N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan. An analysis of SAT-based model checking techniques in an industrial environment. In D. Borrione and W. J. Paul, editors, *Proceedings of 13th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *Lecture Notes in Computer Science*, pages 254–268, 2005.

[AS06a] M. Awedh and F. Somenzi. Automatic invariant strengthening to prove properties in bounded model checking. In *Proceedings of the 43rd Design Automation Conference (DAC'06)*, pages 1073–1076. ACM, 2006.

[AS06b] M. Awedh and F. Somenzi. Termination criteria for bounded model checking: Extensions and comparison. *Electronic Notes in Theoretical Computer Science*, 144(1):51–66, 2006. In *Proceedings of the 3rd International Workshop on Bounded Model Checking (BMC'05)*.

[BAS02] A. Biere, C. Artho, and V. Schuppan. Liveness Checking as Safety Checking. *Electronic Notes in Theoretical Computer Science*, 66(2), 2002. In *Proceedings of the 7th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'02)*.

[BBC+05] G. P. Bischoff, K. S. Brace, G. Cabodi, S. Nocco, and S. Quer. Exploiting target enlargement and dynamic abstraction within mixed BDD and SAT invariant checking. *Electronic Notes in Theoretical Computer Science*, 119(2):33–49, 2005. In *Proceedings of the 2nd International Workshop on Bounded Model Checking (BMC'04)*.

[BC00] P. Bjesse and K. Claessen. SAT-based Verification without State Space Traversal. In W. A. H. Jr. and S. D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer Aided Design (FMCAD'00)*, volume 1954 of *Lecture Notes in Computer Science*, pages 372–389. Springer, 2000.

[BCCZ99] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In R. Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[BCRZ99] A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In N. Halbwachs and D. Peled, editors, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 1999.

[BHJ+06] A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science (LMCS'06)*, 2(5:5), 2006.

[BK18] A. Biere and D. Kroening. Sat-based model checking. In E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors, *Handbook of Model Checking.*, pages 277–303. Springer, 2018.

[BKA02] J. Baumgartner, A. Kuehlmann, and J. A. Abraham. Property Checking via Structural Analysis. In E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 2002.

[BM07] A. R. Bradley and Z. Manna. Checking safety by inductive generalization of counterexamples to induction. In *Proceedings of 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD'07)*, pages 173–180. IEEE Computer Society, 2007.

[Bor97] A. Borälv. The industrial success of verification tools based on Stålmarck's method. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 7–10. Springer, 1997.

[Bra11] A. R. Bradley. Sat-based model checking without unrolling. In R. Jhala and D. A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.

[Bry86] R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C(35), 1986.

[BvDH17] A. Biere, T. van Dijk, and K. Heljanko. Hardware model checking competition 2017. In D. Stewart and G. Weissenbacher, editors, *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, page 9. IEEE, 2017.

[CE82] E. M. Clarke and A. Emerson. Design and Synthesis of Synchroniza-

tion Skeletons Using Branching-Time Temporal Logic. In *Proceedings of the Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer, 1982.

[CGH97] E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design.*, 10(1):47–71, 1997.

[CGJ+03] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5), 2003.

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT press, 1999.

[CGS98] M. Cadoli, A. Giovanardi, and M. Schaerf. An Algorithm to Evaluate Quantified Boolean Formulae. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 262–267, 1998.

[CKOS04] E. M. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and complexity of bounded model checking. In B. Steffen and G. Levi, editors, *Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'04)*, volume 2937 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2004.

[CKOS05] E. M. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Computational challenges in bounded model checking. *Software Tools for Technology Transfer (STTT)*, 7(2):174–183, 2005.

[CKS07] B. Cook, D. Kroening, and N. Sharygina. Verification of boolean programs with unbounded thread creation. *Theoretical Computer Science*, 388(1-3):227–242, 2007.

[CLP+14] G. Cabodi, C. Loiacono, M. Palena, P. Pasini, D. Patti, S. Quer, D. Vendraminetto, A. Biere, and K. Heljanko. Hardware model checking competition 2014: An analysis and comparison of solvers and benchmarks. *JSAT*, 9:135–172, 2014.

[CNQ07] G. Cabodi, S. Nocco, and S. Quer. Boosting the role of inductive invariants in model checking. In R. Lauwereins and J. Madsen, editors, *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'07)*, pages 1319–1324. ACM, 2007.

[Cra57] W. Craig. Linear reasoning: A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.

[DHK05] N. Dershowitz, Z. Hanna, and J. Katz. Bounded model checking with QBF. In F. Bacchus and T. Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *Lecture Notes in Computer Science*. Springer, 2005.

[dMRS03] L. M. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification. In W. A. H. Jr. and F. Somenzi, editors, *Proceedings of the 15th International Conferences on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 14–26. Springer, 2003.

[EMB11] N. Eén, A. Mishchenko, and R. K. Brayton. Efficient implementation

of property directed reachability. In P. Bjesse and A. Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 125–134. FMCAD Inc., 2011.

[Eme90] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. MIT Press, 1990.

[ES03] N. Eén and N. Sörensson. Temporal Induction by Incremental SAT Solving. *Electronic Notes in Theoretical Computer Science*, 89(4), 2003. In *Proceedings of the 1st International Workshop on Bounded Model Checking (BMC'03)*.

[FH05] M. Fränzle and C. Herde. Efficient proof engines for bounded model checking of hybrid systems. *Electronic Notes in Theoretical Computer Science*, 133:119–137, 2005.

[FSW02] A. M. Frisch, D. Sheridan, and T. Walsh. A fixpoint encoding for bounded model checking. In *Proceedings of the 4th International Conference on Formal Methods in Computer Aided Design (FMCAD'02)*, volume 2517 of *Lecture Notes in Computer Science*, pages 238–255. Springer, 2002.

[Gel07] A. V. Gelder. Verifying propositional unsatisfiability: Pitfalls to avoid. In J. Marques-Silva and K. A. Sakallah, editors, *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 328–333. Springer, 2007.

[GGA04] M. K. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In *Proceedings International Conference on Computer-Aided Design (ICCAD'04)*, pages 510–517. IEEE Computer Society / ACM, 2004.

[Hel01] K. Heljanko. Bounded reachability checking with process semantics. In K. G. Larsen and M. Nielsen, editors, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2001.

[HJK+06] K. Heljanko, T. A. Junttila, M. Keinänen, M. Lange, and T. Latvala. Bounded model checking for weak alternating Büchi automata. In T. Ball and R. B. Jones, editors, *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2006.

[HJL05] K. Heljanko, T. A. Junttila, and T. Latvala. Incremental and complete bounded model checking for full PLTL. In K. Etessami and S. K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 98–111. Springer, 2005.

[HN03] K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.

[Hol04] G. Holzmann. *The SPIN Model Checker*. Addison Wesley, 2004.

[JB07] T. Jussila and A. Biere. Compressing BMC encodings with QBF.

Electronic Notes in Theoretical Computer Science, 174(3):45–56, 2007. In *Proceedings of the 4th International Workshop on Bounded Model Checking (BMC'06)*.

[JHN03] T. Jussila, K. Heljanko, and I. Niemelä. BMC via on-the-fly determinization. *Electronic Notes in Theoretical Computer Science*, 89(4), 2003. In *Proceedings of the 1st International Workshop on Bounded Model Checking (BMC'03)*.

[KK97] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Proceedings of the 34th Design Automation Conference (DAC'97)*, pages 263–268. ACM, 1997.

[KPKG02] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai. Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1377–1394, 2002.

[Kra97] J. Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal on Symbolic Logic*, 62(2):457–486, 1997.

[KS92] H. Kautz and B. Selman. Planning as satisfiability. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363. John Wiley & Sons, 1992.

[KS03] D. Kroening and O. Strichman. Efficient computation of recurrence diameters. In L. D. Zuck, P. C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'03)*, volume 2575 of *Lecture Notes in Computer Science*, pages 298–309. Springer, 2003.

[Kue04] A. Kuehlmann. Dynamic transition relation simplification for bounded property checking. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'04)*, pages 50–57. IEEE Computer Society / ACM, 2004.

[Kur08] R. P. Kurshan. Verification technology transfer. In H. Veith and O. Grumberg, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 46–64. Springer, 2008.

[KWS00] J. Kim, J. Whittemore, and K. Sakallah. On Solving Stack-Based Incremental Satisfiability Problems. In *Proceedings of the International Conference on Computer Design (ICCD'00)*, pages 379–382, 2000.

[LBHJ04] T. Latvala, A. Biere, K. Heljanko, and T. A. Junttila. Simple bounded LTL model checking. In *Proceedings of the 6th International Conference on Formal Methods in Computer Aided Design (FMCAD'04)*, volume 3312 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2004.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state programs satisfy their linear specification. In *ACM Symposium on Principles of Programming Languages*, pages 97–107, 1985.

[MA03] K. L. McMillan and N. Amla. Automatic abstraction without counterexamples. In H. Garavel and J. Hatcliff, editors, *Proceedings of the International Conference on Tools and Algorithms for the Construc-*

tion and Analysis of Systems (TACAS'03), volume 2619 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2003.

[Mai00] M. Maidl. *Using model checking for system verification*. PhD thesis, Ludwig-Maximilians-Universiät at München, 2000.

[MCB06] A. Mishchenko, S. Chatterjee, and R. K. Brayton. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In E. Sentovich, editor, *Proceedings of the 43rd Design Automation Conference (DAC'06)*, pages 532–535. ACM, 2006.

[McM93] K. L. McMillan. *Symbolic Model Checking: An approach to the State Explosion Problem*. Kluwer, 1993.

[McM02] K. L. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14th International Conference on Computer-Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2002.

[McM03] K. L. McMillan. Interpolation and SAT-based Model Checking. In J. W. A. Hunt and F. Somenzi, editors, *Proceedings of the 15th Conference on Computer-Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

[McM05] K. L. McMillan. An interpolating theorem prover. *Theoretical Computer Science*, 345(1), 2005.

[MMZ+01] M. H. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[MS03] M. Mneimneh and K. Sakallah. SAT-based sequential depth computation. In *Proceedings of the Asia South Pacific Design Automation Conference (ASPDAC'03)*, pages 87–92. ACM, 2003.

[MSS99] J. P. Marques-Silva and K. A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[MVS+07] H. Mangassarian, A. G. Veneris, S. Safarpour, M. Benedetti, and D. Smith. A performance-driven QBF-based iterative logic array representation with applications to verification, debug and test. In G. G. E. Gielen, editor, *Proceedings of the International Conference on Computer-Aided Design (ICCAD'07)*, pages 240–245. IEEE, 2007.

[Nie99] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[PBG05] M. Prasad, A. Biere, and A. Gupta. A survey on recent advances in SAT-based formal verification. *Software Tools for Technology Transfer (STTT)*, 7(2), 2005.

[PBZ03] D. Plaisted, A. Biere, and Y. Zhu. A Satisfiability Procedure for Quantified Boolean Formulae. *Discrete Applied Mathematics*, 130(2):291–328, 2003.

[Pel94] D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, pages 377–390. Springer-Verlag,

1994.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 1977.

[PSD06] F. Pigorsch, C. Scholl, and S. Disch. Advanced unbounded model checking based on AIGs, BDD sweeping, and quantifier scheduling. In *Proceedings 6th International Conference on Formal Methods in Computer-Aided Design (FMCAD'06)*, pages 89–96. IEEE Computer Society, 2006.

[Pud97] P. Pudlák. Lower bounds for resolution and cutting planes proofs and monotone computations. *Journal of Symbolic Logic*, 62(3), 1997.

[QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*. Springer, 1982.

[Sav70] W. J. Savitch. Relational between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[SB04] V. Schuppan and A. Biere. Efficient reduction of finite state model checking to reachability analysis. *Software Tools for Technology Transfer (STTT)*, 5(1-2):185–204, 2004.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[SNS02] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Journal on Artificial Intelleligence*, 138(1-2):181–234, 2002.

[SSS00] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In W. A. H. Jr. and S. D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer Aided Design (FMCAD'00)*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.

[Str01] O. Strichman. Pruning Techniques for the SAT-based Bounded Model Checking Problem. In T. Margaria and T. F. Melham, editors, *Proceedings of the 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'01)*, volume 2144 of *Lecture Notes in Computer Science*, pages 58–70. Springer, 2001.

[Tse68] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part II*, volume 8 of *Seminars in Mathematics*, pages 234–259. V.A. Steklov Mathematical Institute, 1968. English Translation: Consultants Bureau, New York, 1970, pages 115 – 125.

[Var08] M. Y. Vardi. From Church and Prior to PSL. In H. Veith and O. Grumberg, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 150–171. Springer, 2008.

[vE98] C. A. J. van Eijk. Sequential equivalence checking without state space traversal. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'98)*, pages 618–623. IEEE Computer So-

ciety, 1998.

[VG08] H. Veith and O. Grumberg, editors. *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*. Springer, 2008.

[VW94] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[WBCG00] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2000.

[WKS01] J. P. Whittemore, J. Kim, and K. A. Sakallah. SATIRE: A New Incremental Satisfiability Engine. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 542–545, 2001.

[WVS83] P. Wolper, M. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In *Proceedings of 24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, pages 185–194. IEEE Computer Society, 1983.

[YM04] G. Yorsh and M. Musuvathi. A combination method for generating interpolants. Technical Report MSR-TR-2004-108, Microsoft Research, 2004.

[YM05] G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In R. Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction (CADE'05)*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.

[Zha97] H. Zhang. SATO: An efficient propositional prover. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction (CADE'97)*, volume 1249 of *Lecture Notes in Computer Science*, pages 272–275. Springer, 1997.

[ZM03] L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'03)*, pages 10880–10885. IEEE Computer Society, 2003.