Parallel SAT-Based Parameterised Three-Valued Model Checking

Nils Timm, Stefan Gruner, and Prince Sibanda

Department of Computer Science, University of Pretoria, South Africa
{ntimm,sgruner}@cs.up.ac.za

Abstract. *Parameterisation* in three-valued model checking (PMC) allows to establish logical connections between unknown parts in state space models. The application of parameterisation enhances the precision of models without increasing their state space, but it leads to an exponential growth of the number of model checking instances that have to be checked consecutively. Here, we introduce a technique for PMC via parallel SAT solving which enables us to significantly reduce the time overhead of PMC by exploiting similarities among the instances. We define bounded semantics and a propositional logic encoding of PMC. Moreover, we introduce a concept for sharing clauses between the instances of parallel SAT-based PMC. Our experiments show that our new approach leads to a practically relevant speed-up of parameterised three-valued model checking.

1 Introduction

Three-valued predicate abstraction [16, 21] is an established technique in software verification. It proceeds by generating a state space model of the system to be analysed over the values *true*, *false* and *unknown*, where the latter value is used to represent the loss of information due to abstraction. The evaluation of temporal logic properties on such models is known as *three-valued model checking* (3MC) [6, 8]. In case of an *unknown* result in verification, the abstraction is iteratively refined by adding more predicates over the variables of the system until a level of abstraction is reached where the property can be either definitely proved or refuted. Refinement does, however, not guarantee that eventually a three-valued model can be constructed that is both precise enough for a definite outcome and small enough to be manageable with the available computational resources.

In [22] we introduced *parameterised three-valued model checking* (PMC) as an extension of 3MC. Predicates and transitions in PMC models can be either associated with the values *true*, *false*, *unknown*, or with expressions over *Boolean parameters*. Parameterisation is an alternative way to state that the value of certain predicates or transitions is actually not known and that the checked property has to yield the same result under each possible parameter instantiation. PMC is thus conducted via evaluating a property under all parameter instantiations and checking whether the results are consistent. Parameterisation particularly allows to establish *logical connections* between unknowns in the abstract model:

While unknown parts in 3MC are never related to each other, the parameterisation approach enables to represent facts like 'a certain pair of transitions has unknown but *complementary* truth values', or 'the value of a predicate is unknown but remains *unchanged* along all states of a certain path'. Such facts can be automatically derived from the system to be verified, and covering these facts in a model can be crucial for the success and the efficiency of model checking. We showed that combining classical refinement and parameterisation in abstraction-based model checking is highly suited for obtaining the necessary precision for a definite result while keeping the state space small. However, parameterisation generally leads to an *exponential* increase in time complexity, since any property of interest must be checked for each possible parameter instantiation.

Here, we introduce a technique for parameterised three-valued model checking via parallel SAT solving which enables us to considerably reduce the time overhead of PMC by effectively exploiting similarities between the occurring instances. Our approach is based on *bounded model checking* (BMC) [3]. We define bounded semantics for PMC and we introduce a parameterised propositional encoding of PMC problems. In order to obtain the overall result of an encoded PMC problem, the satisfiability of each possible parameter instantiation of the encoding is tested and it is checked whether all single results are consistent.

An integral part of modern satisfiability solving algorithms is *conflict-driven clause learning* [17]: SAT solvers search for a satisfying assignment of the input formula by successively selecting unassigned Boolean variables, assigning them to either *true* or *false*, and propagating the resulting constraints to the clauses of the formula. In case the solver decisions lead to an unsatisfied clause, a so-called conflict clause is learned and added to the formula. Then the solver tracks back by revising a former assignment decision and continuing the search from this point. Clause learning enables to quickly prune parts of the search space and is thus crucial for the performance of SAT solving. In our approach, we exploit the fact that the instances associated with a parameterised encoding exhibit considerable similarities in terms of large common subformulae. Thus, a conflict clause that was learned during the SAT test of one instance can be shared with another instance that is SAT checked at the same time, provided that the new clause was derived based on a common part of the two instances.

We implemented a parallel SAT-based model checker for PMC problems. The checks of the instances of a parameterised encoding are performed simultaneously and clauses that have been learned are are shared between the instances. In experiments we show that our concept of clause sharing in parallel SAT-based PMC leads to substantial savings in verification time.

2 Background

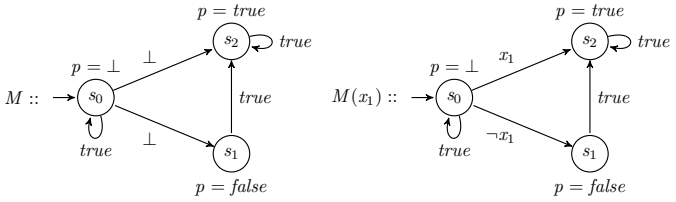
We start with an introduction to *pure* and *parameterised* three-valued models. The key feature of these models is a third truth value $\bot$ (i.e. *unknown*) for transitions and labellings, which is used to model uncertainty. The parameterised case

additionally allows *Boolean parameter expressions* for transitions and labellings, which enables to establish logical connections between unknown parts.

Definition 1 (Parameterised Three-Valued Kripke Structure). *A* parameterised three-valued Kripke structure *over a set of atomic predicates AP and a set of Boolean parameters $X = \{x_1, \ldots, x_m\}$ is a parameterised tuple $M(\overset{m}{x}) = (S, s_0, R(\overset{m}{x}), L(\overset{m}{x}))$ where*

- *S is a finite set of states and $s_0 \in S$ is the initial state,*
- *$R(\overset{m}{x}) : S \times S \to \{true, \bot, false\} \cup B(X)$ is a transition function with $\forall\, s \in S : \exists\, s' \in S : R(\overset{m}{x})(s, s') \in \{true, \bot\} \cup B(X)$ where $B(X)$ denotes the set of Boolean expressions over X,*
- *$L(\overset{m}{x}) : S \times AP \to \{true, \bot, false\} \cup B(X)$ is a labelling function that associates a truth value or a parameter expression with each predicate in each state.*

Note that $(\overset{m}{x})$ is an abbreviation for the parameter tuple $(x_1, \ldots, x_m)$. An *instantiation* of a parameterised three-valued Kripke structure $M(\overset{m}{x})$ is a *pure* three-valued Kripke structure $M(\overset{m}{a})$ where $(\overset{m}{a}) \in \{true, false\}^m$. Hence, all parameters are substituted by *Boolean* truth values. However, predicates and transitions that were not parameterised in $M(\overset{m}{x})$ may still hold the value *unknown* in $M(\overset{m}{a})$. A structure is also *pure* if $X = \varnothing$. If the tuple of parameters is clear from the context we will just refer to M, R, L. An example for a pure three-valued Kripke structure M and a parameterised Kripke structure $M(x_1)$ is depicted below.



In abstraction-based model checking a parameterised three-valued Kripke structure can be obtained by refining a pure three-valued Kripke structure [22]. For instance, if the transitions (s_0, s_1) and (s_0, s_2) of our example structure M correspond to a complementary branch (e.g. *if-then-else* or *while-do*) in the modelled system, then $M(x_1)$ is a sound refinement of M.

In the following, we first introduce model checking for pure three-valued Kripke structures and then generalise it to the parameterised case. A path π of a pure three-valued Kripke structure M is an infinite sequence of states $s_0 s_1 s_2 \ldots$ with $R(s_i, s_{i+1}) \in \{true, \bot\}$. π_i denotes the i-th state of π, whereas π^i denotes the i-th suffix $\pi_i \pi_{i+1} \pi_{i+2} \ldots$ of π. By Π_M we denote the set of all paths of M starting in the initial state. We use the temporal logic LTL for specifying properties with regard to paths.

Definition 2 (Syntax of LTL). *Let AP be a set of atomic predicates and $p \in AP$. The* syntax of LTL formulae *ψ is given by*

$$\psi \ ::= \ p \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \mathbf{G}\psi \mid \mathbf{F}\psi \mid \psi \mathbf{U} \psi.$$

Due to the extended domain of truth values in three-valued Kripke structures, the evaluation of LTL formulae is not based on classical two-valued logic. In three-valued model checking we operate under the three-valued Kleene logic $\mathcal{L}_3$ [10] whose semantics is given by the truth tables below.

$\wedge$	*true*	$\bot$	*false*
true	*true*	$\bot$	*false*
$\bot$	$\bot$	$\bot$	*false*
false	*false*	*false*	*false*

$\vee$	*true*	$\bot$	*false*
true	*true*	*true*	*true*
$\bot$	*true*	$\bot$	$\bot$
false	*true*	$\bot$	*false*

$\neg$	
true	*false*
$\bot$	$\bot$
false	*true*

$\mathcal{L}_3$ has a reflexive *truth ordering* $\leq_{\mathcal{L}_3}$ (in words: 'less or equally true as') with $\mathit{false} \leq_{\mathcal{L}_3} \bot \leq_{\mathcal{L}_3} \mathit{true}$. Based on $\mathcal{L}_3$, LTL formulae can be evaluated on paths of three-valued Kripke structures according to the following definition.

Definition 3 (Three-Valued Evaluation of LTL). *Let $M = (S, s_0, R, L)$ over AP be a pure three-valued Kripke structure. Then the* evaluation *of an LTL formula ψ on a path $\pi \in \Pi_M$, written $[\pi \models \psi]$, is defined as follows*

$$
\begin{array}{lcl}
[\pi \models p] & := & L(\pi_0, p) \\
[\pi \models \neg\psi] & := & \neg\,[\pi \models \psi] \\
[\pi \models \psi \vee \psi'] & := & [\pi \models \psi] \;\vee\; [\pi \models \psi'] \\
[\pi \models \mathbf{G}\psi] & := & \bigwedge_{i\in\mathbb{N}} (R(\pi_i, \pi_{i+1}) \wedge [\pi^i \models \psi]) \\
[\pi \models \mathbf{F}\psi] & := & \bigvee_{i\in\mathbb{N}} ([\pi^i \models \psi] \wedge \bigwedge_{j=0}^{i-1} R(\pi_j, \pi_{j+1}))
\end{array}
$$

Due to space limitations we have omitted the definitions for the operators $\wedge$, $\mathbf{X}$ and $\mathbf{U}$. The complete definitions can e.g. be found in [24, 20]. The evaluation of LTL formulae on entire three-valued Kripke structures is what is called *three-valued model checking* (3MC) [6, 8].

Definition 4 (Three-Valued LTL Model Checking). *Let $M = (S, s_0, R, L)$ over AP be a three-valued Kripke structure. Moreover, let ψ be an LTL formula over AP. The* universal value *of ψ in M, written $[M \models_U \psi]$, is defined as*

$$[M \models_U \psi] \;:=\; \bigwedge_{\pi\in\Pi_M} [\pi \models \psi]$$

The existential value *of ψ in M, written $[M \models_E \psi]$, is defined as*

$$[M \models_E \psi] \;:=\; \bigvee_{\pi\in\Pi_M} [\pi \models \psi]$$

Universal model checking can always be transferred into existential model checking based on the equation $[M \models_U \psi] \;=\; \neg\,[M \models_E \neg\psi]$.

In 3MC there exist three possible outcomes: *true*, *false* and $\bot$. For our example Kripke structure M we get $[M \models_U \mathbf{G}\neg p] = \neg[M \models_E \mathbf{F}p] = \bot$ and $[M \models_U \mathbf{GF}\neg p] = \neg[M \models_E \mathbf{FG}p] = \bot$. Hence, M is not precise enough for a definite result in these verification tasks. $\mathbf{G}\neg p$ is a temporal logic formula that is a typical example of a universal safety property, whereas $\mathbf{GF}\neg p$ is an example of a liveness property. Safety and liveness are the most vital requirements in software verification. In our approach, we therefore particularly focus on these two

kinds of properties, or rather their existential counterparts $\mathbf{F}p$ and $\mathbf{FG}p$. From now on, we only consider the existential case, since it is the basis for *bounded* model checking which we later apply.

3MC reduces to two-valued model checking (2MC) if the Kripke structure M is actually two-valued, i.e. $R^{-1}(\bot) = \varnothing$ and $L^{-1}(\bot) = \varnothing$. In this case, only definite outcomes are possible. Moreover, 3MC can always be reduced to two instances of 2MC if the LTL formula is restricted to LTL^+, which is the negation-free fragment of LTL. LTL^+ formulae are evaluated on *complement-closed* Kripke structures. In these structures each $p \in AP$ has a complementary $\bar{p} \in AP$ such that $L(s,p) = \neg L(s,\bar{p})$. Every Kripke structure can be straightforwardly extended to a complement-closed one, without increasing the number of states. For the evaluation on complement-closed Kripke structures, each LTL formula can be transferred into an equivalent LTL^+ formula. Thus, the restriction to LTL^+ does not limit the expressiveness of our approach. The reduction of 3MC to two instances of 2MC is based on *completions* of complement-closed structures.

Definition 5 (Completion). *Let $M = (S, s_0, R, L)$ over AP be a three-valued Kripke structure. Then $M^p = (S, s_0, R^p, L^p)$ is the* pessimistic completion *and $M^o = (S, s_0, R^o, L^o)$ is the* optimistic completion *with $\forall\, s, s' \in S$ and $\forall\, p \in AP$:*

$$L^p(s,p) := \begin{cases} \mathit{false} & \mathit{if}\ L(s,p) = \bot \\ L(s,p) & \mathit{else} \end{cases} \qquad R^p(s,s') := \begin{cases} \mathit{false} & \mathit{if}\ R(s,s') = \bot \\ R(s,s') & \mathit{else} \end{cases}$$

$$L^o(s,p) := \begin{cases} \mathit{true} & \mathit{if}\ L(s,p) = \bot \\ L(s,p) & \mathit{else} \end{cases} \qquad R^o(s,s') := \begin{cases} \mathit{true} & \mathit{if}\ R(s,s') = \bot \\ R(s,s') & \mathit{else} \end{cases}$$

From [7] we now get the following theorem that allows us to reduce three-valued model checking to two-valued model checking.

Theorem 1. *Let $M = (S, s_0, R, L)$ be a complement-closed three-valued Kripke structure and ψ an LTL$^+$ formula. Then the following holds:*

$$[M \models_E \psi] = \begin{cases} \mathit{true} & \mathit{if}\ [M^p \models_E \psi] = \mathit{true} \\ \mathit{false} & \mathit{if}\ [M^o \models_E \psi] = \mathit{false} \\ \bot & \mathit{else} \end{cases}$$

Hence, if a formula holds for the pessimistic completion we can conclude that it also holds for the three-valued Kripke structure. The same applies to *false* results obtained for the optimistic completion. All definitions wrt. pure 3MC can be straightforwardly generalised to *parameterised* three-valued model checking (PMC) [22], since PMC reduces to multiple instances of pure 3MC. In PMC we consider all parameter instantiations of a parameterised Kripke structure.

Definition 6 (Parameterised Three-Valued LTL$^+$ Model Checking). *Let $M(\vec[m]{x}) = (S, s_0, R(\vec[m]{x}), L(\vec[m]{x}))$ be a parameterised three-valued Kripke structure over AP and $X = \{x_1, \ldots, x_m\}$. Moreover, let ψ be an LTL$^+$ formula over*

AP. The existential value of ψ *in* $M(\vec{x})$*, written* $[M(\vec{x}) \models_E \psi]$*, is defined as*

$$[M(\vec{x}) \models_E \psi] := \begin{cases} \mathit{true} & \mathit{if}\ \forall (\vec{a}) \in \{t,f\}^m\ ([M(\vec{a}) \models_E \psi] = \mathit{true}) \\ \mathit{false} & \mathit{if}\ \forall (\vec{a}) \in \{t,f\}^m\ ([M(\vec{a}) \models_E \psi] = \mathit{false}) \\ \bot & \mathit{else} \end{cases}$$

Thus, if checking a temporal logic property yields *true* for all instantiations, this result is transferred to the parameterised Kripke structure. The same holds for *false* results for all instantiations. In all other cases PMC returns *unknown*. For our example $M(x_1)$, we get $[M(x_1) \models_E \mathbf{F}p] = \mathit{true}$ since $\mathbf{F}p$ holds existentially for both $M(\mathit{true})$ and $M(\mathit{false})$. By same argumentation we can show that $[M(x_1) \models_E \mathbf{FG}p]$ also yields *true*. In contrast to the pure three-valued M, the parameterised $M(x_1)$ captures the fact that the transition values of (s_0, s_1) and (s_0, s_2) are unknown but also *complementary*, which gives us the necessary precision for the definite verification results. In the remainder of this work we will show how PMC problems can be encoded in propositional logic and then efficiently solved via SAT solving with clause sharing. A prerequisite for the encoding is to *bound* the length of the considered paths in model checking.

3 Bounded Semantics

Bounded model checking (BMC) [4, 3] via satisfiability solving requires to consider finite prefixes of paths, bounded in length by some $k \in \mathbb{N}$. Such finite prefixes $\pi_0 \dots \pi_k$ can still represent infinite paths if the prefix has a *loop,* i.e. the last state π_k has a successor state that is also part of the prefix.

Definition 7 (k-Loop). *Let* π *be a path of a three-valued Kripke structure* M *and let* $l, k \in \mathbb{N}$ *with* $l \leq k$*. Then* π *has a* (k,l)*-loop, if* $R(\pi_k, \pi_l) \in \{\mathit{true}, \bot\}$ *and* π *is of the form* $u \cdot v^\omega$ *where* $u = \pi_0 \dots \pi_{l-1}$ *and* $v = \pi_l \dots \pi_k$*. Moreover,* π *has a* k*-loop, if there exists an* $l \in \mathbb{N}$ *with* $l \leq k$ *such that* π *has a* (k,l)*-loop.*

For the bounded evaluation of temporal logic formulae on paths of Kripke structures we have to distinguish between paths *with* and *without* a k-loop.

Definition 8 (Three-Valued Bounded Evaluation of LTL$^+$). *Let* $M = (S, s_0, R, L)$ *over* AP *be a complement-closed three-valued Kripke structure, let* $k \in \mathbb{N}$ *and let* π *be a path of* M *without a* k*-loop. Then the* k*-bounded evaluation of an LTL*$^+$ *formula* ψ *on* π*, written* $\left[\pi \models^i_k \psi\right]$ *where* $i \in \mathbb{N}$ *with* $i \leq k$ *denotes the current position along the path, is inductively defined as follows*

$$\begin{array}{lcl} \left[\pi \models^i_k p\right] & := & L(\pi_i, p) \\ \left[\pi \models^i_k \psi \vee \psi'\right] & := & \left[\pi \models^i_k \psi\right] \ \vee\ \left[\pi \models^i_k \psi'\right] \\ \left[\pi \models^i_k \mathbf{G}\psi\right] & := & \mathit{false} \\ \left[\pi \models^i_k \mathbf{F}\psi\right] & := & \bigvee_{j=i}^{k} ([\pi \models^j_k \psi] \wedge \bigwedge_{l=i}^{j-1} R(\pi_l, \pi_{l+1})) \end{array}$$

If π *has a* k*-loop, then* $[\pi \models^i_k \psi] := [\pi^i \models \psi]$*. Moreover, the existential value of* ψ *in* M *under the bounded semantics is* $[M \models_{E,k} \psi] := \bigvee_{\pi \in \Pi_M} [\pi \models^0_k \psi]$*.*

The bounded evaluation of LTL$^+$ approximates the unbounded evaluation wrt. the truth ordering of $\mathcal{L}_3$: $[M \models_{E,k} \psi] \leq_{\mathcal{L}_3} [M \models_E \psi]$. Hence, all *true* results in three-valued BMC can be transferred to the corresponding unbounded case, whereas *unknown* and *false* results do not allow to draw such conclusions about the unbounded case. At least, a *false* result for a bound k tells us that there is definitely no path prefix of length k for which ψ holds. Moreover, a *false* result can be transferred to the unbounded case, if k has reached a *completeness threshold*, which can be derived based on M and ψ [3]. For instance, the completeness threshold for a safety formula ψ is linear in the number of states of M. If all possible values for k are considered, then the bounded semantics are equivalent to the unbounded one: $[M \models_E \psi] = \bigvee_{k \in \mathbb{N}} [M \models_{E,k} \psi]$. The bounded semantics for 3MC can be straightforwardly extended to the parameterised case, as PMC reduces to multiple instances of pure 3MC.

Definition 9 (Bounded Parameterised Three-Valued Model Checking). *Let $M(\overset{m}{x}) = (S, s_0, R(\overset{m}{x}), L(\overset{m}{x}))$ be a parameterised three-valued Kripke structure over AP and $X = \{x_1, \ldots, x_m\}$. Moreover, let ψ be an LTL$^+$ formula over AP and $k \in \mathbb{N}$. The existential value of ψ in $M(\overset{m}{x})$ under the bounded semantics, written $[M(\overset{m}{x}) \models_{E,k} \psi]$, is defined as*

$$[M(\overset{m}{x}) \models_{E,k} \psi] := \begin{cases} \mathit{true} & \mathit{if}\ \forall (\overset{m}{a}) \in \{t,f\}^m\ ([M(\overset{m}{a}) \models_{E,k} \psi] = \mathit{true}) \\ \mathit{false} & \mathit{if}\ \forall (\overset{m}{a}) \in \{t,f\}^m\ ([M(\overset{m}{a}) \models_{E,k} \psi] = \mathit{false}) \\ \bot & \mathit{else} \end{cases}$$

BMC is typically performed incrementally, i.e. the bound is iteratively increased until the property of interest can be either proven or the completeness threshold is reached. For our running example from the previous section, we require the following iterations in order to prove that $\mathbf{F}p$ holds existentially for $M(x_1)$.

$$[M(x_1) \models_{E,0} \mathbf{F}p] = \bot \quad [M(x_1) \models_{E,1} \mathbf{F}p] = \bot \quad [M(x_1) \models_{E,2} \mathbf{F}p] = \mathit{true}$$

For $k = 0$ we can only consider the state s_0 where p is $\bot$. For $k = 1$ the prefixes $(s_0 s_1)$ and $(s_0 s_2)$ are considered. Here $\mathbf{F}p$ holds for the instantiation $M(\mathit{true})$ but not for $M(\mathit{false})$. Thus, the overall result is still $\bot$. For $k = 2$ we get under both instantiations a prefix where finally p holds. Next, we will see how bounded PMC can be reduced to satisfiability solving.

4 Propositional Logic Encoding

Now we reduce bounded parameterised three-valued model checking to the new satisfiability problem SAT_{X3}. For a parameterised three-valued Kripke structure $M(\overset{m}{x}) = (S, s_0, R(\overset{m}{x}), L(\overset{m}{x}))$ over AP and X, an LTL$^+$ formula ψ and a bound $k \in \mathbb{N}$, a propositional logic formula $[\![M(\overset{m}{x}), \psi]\!]_k$ is constructed such that

$$[M(\overset{m}{x}) \models_{E,k} \psi] = \mathrm{SAT}_{X3}([\![M(\overset{m}{x}), \psi]\!]_k)$$

$[\![M(\vec{x}), \psi]\!]_k$ is defined over a set of Boolean atoms, the set of Boolean parameters X, and the constants *true*, *false* and $\bot$. Hence, $[\![M(\vec{x}), \psi]\!]_k$ is parameterised wrt. X. We will show that solving SAT_{X3} reduces to solving classical SAT for each possible parameter instantiation. First, we give a step-by-step description on how $[\![M(\vec{x}), \psi]\!]_k$ is constructed for a given bounded PMC problem.

A propositional logic encoding of bounded model checking problems for Kripke structures with three-valued labelling functions was introduced in [24]. Here we generalise it to our *parameterised* three-valued Kripke structures. The construction of $[\![M(\vec{x}), \psi]\!]_k$ is divided into the translation of the Kripke structure $M(\vec{x})$ into a formula $[\![M(\vec{x})]\!]_k$ and the translation of the temporal logic property ψ into a formula $[\![\psi]\!]_k$. We start with the encoding of the Kripke structure $M(\vec{x}) = (S, s_0, R(\vec{x}), L(\vec{x}))$, which first requires to encode its states as Boolean formulae. For this, we introduce a set of Boolean atoms $\{A, B, \ldots\}$. Let L be the set of propositional logic formulae over $\{A, B, \ldots\}$ and the constants *true* and *false*. Then an encoding of the states of a Kripke structure is defined as follows.

Definition 10 (State Encoding). *Let $M(\vec{x}) = (S, s_0, R(\vec{x}), L(\vec{x}))$ be a parameterised Kripke structure. A Boolean encoding of its states corresponds to an injective mapping $e : S \to L$ where $\forall\, s \in S : e(s)$ is a conjunction of literals.*

The states s_0, s_1, and s_2 of our example structure $M(x_1)$ can be encoded over the set of atoms $\{A, B\}$ as follows: $e(s_0) = \neg A \wedge \neg B \quad e(s_1) = \neg A \wedge B \quad e(s_2) = A \wedge \neg B$.

An *assignment* is a mapping $\tau : \{A, B, \ldots\} \to \{\mathit{true}, \mathit{false}\}$. For instance, the assignment $\tau(A) = \mathit{false}$ and $\tau(B) = \mathit{true}$ characterises the state s_1, since it is the only assignment that makes the encoding $e(s_1)$ *true*. An entire Kripke structure can now be translated into the formula $[\![M(\vec{x})]\!]_k$ that characterises path prefixes of length k in $M(\vec{x})$. Since we consider states as parts of such prefixes, we have to extend the encoding of states by index values $i \in \{0, \ldots, k\}$ where i denotes the position along a path prefix. For example, $e(s_1)_i = A_i \wedge B_i$ refers to s_1 as the i-th state of a certain prefix. Moreover, we get an extended set of indexed atoms $\{A_0, B_0, \ldots, A_k, B_k, \ldots\}$.

Definition 11 (Kripke Structure Encoding). *Let $M(\vec{x}) = (S, s_0, R(\vec{x}), L(\vec{x}))$ be a parameterised three-valued Kripke structure and e an encoding of its states. We define Init_0 as the predicate characterising the initial state of $M(\vec{x})$ with*

$$\mathit{Init}_0 := e(s_0)_0$$

and $T_{i,i+1}$ as the predicate that characterises the transitions of $M(\vec{x})$ with

$$T_{i,i+1} := \bigvee_{s \in S} \bigvee_{s' \in S} e(s)_i \wedge e(s')_{i+1} \wedge R(\vec{x})(s, s').$$

Then the entire encoding of $M(\vec{x})$ for a bound $k \in \mathbb{N}$ is defined as

$$[\![M(\vec{x})]\!]_k := \mathit{Init}_0 \wedge \bigwedge_{i=0}^{k-1} T_{i,i+1}$$

Thus, for our example $M(x_1)$ we get $Init_0 = \neg A_0 \wedge \neg B_0$ and $T_{i,i+1} = (\neg A_i \wedge \neg B_i \wedge \neg A_{i+1} \wedge \neg B_{i+1} \wedge \mathit{true}) \vee (\neg A_i \wedge \neg B_i \wedge \neg A_{i+1} \wedge B_{i+1} \wedge \neg x_1) \vee (\neg A_i \wedge \neg B_i \wedge A_{i+1} \wedge \neg B_{i+1} \wedge x_1) \vee (\neg A_i \wedge B_i \wedge A_{i+1} \wedge \neg B_{i+1} \wedge \mathit{true}) \vee (A_i \wedge \neg B_i \wedge A_{i+1} \wedge \neg B_{i+1} \wedge \mathit{true})$.

As we can see, each clause of $T_{i,i+1}$ encodes a transition of the Kripke structure, where the last literal, resp. constant, of each clause encodes the value of the transition. The assignment $\tau(A_0) = \mathit{false}$, $\tau(B_0) = \mathit{false}$, $\tau(A_1) = \mathit{true}$ and $\tau A(B_1) = \mathit{false}$ characterises that the first transition of a k-prefix $s_0 \dots s_k$ is the transition (s_0, s_2) which is parameterised by x_1.

The second part of the encoding concerns the temporal logic formula ψ. Again, we need to distinguish the cases where ψ is evaluated on a path prefix *with* and *without* a loop.

Definition 12 (LTL$^+$ Encoding without Loop). *Let p be an atomic predicate, ψ and ψ' LTL$^+$ formulae, and $k, i \in \mathbb{N}$ with $i \leq k$.*

$$\begin{array}{lcllcl} [\![p]\!]^i_k & := & \bigvee_{s \in S} e(s)_i \wedge L(s,p) & [\![\mathbf{G}\psi]\!]^i_k & := & \mathit{false} \\ [\![\psi \vee \psi']\!]^i_k & := & [\![\psi]\!]^i_k \vee [\![\psi']\!]^i_k & [\![\mathbf{F}\psi]\!]^i_k & := & \bigvee_{j=i}^{k} [\![\psi]\!]^j_k \end{array}$$

For instance, encoding the LTL$^+$ formula $\mathbf{F}p$ for our example Kripke structure $M(x_1)$ and for bound $k = 2$ yields the following propositional logic formula

$$[\![\mathbf{F}p]\!]^0_2 \;=\; \bigvee_{i=0}^{2} ((\neg A_i \wedge \neg B_i \wedge \bot) \vee (\neg A_i \wedge B_i \wedge \mathit{false}) \vee (A_i \wedge \neg B_i \wedge \mathit{true}))$$

Encoding temporal logic formulae for the evaluation on prefixes *with* a loop additionally requires to explicitly refer the starting position l of the loop.

Definition 13 (LTL$^+$ Encoding with Loop). *Let p be an atomic predicate, ψ and ψ' LTL$^+$ formulae, and $k, i, l \in \mathbb{N}$ with $i, l \leq k$.*

$$\begin{array}{lcllcl} {}_l[\![p]\!]^i_k & := & \bigvee_{s \in S} e(s)_i \wedge L(s,p) & {}_l[\![\mathbf{G}\psi]\!]^i_k & := & \bigwedge_{j=min(i,l)}^{k} {}_l[\![\psi]\!]^j_k \\ {}_l[\![\psi \vee \psi']\!]^i_k & := & {}_l[\![\psi]\!]^i_k \vee {}_l[\![\psi']\!]^i_k & {}_l[\![\mathbf{F}\psi]\!]^i_k & := & \bigvee_{j=min(i,l)}^{k} {}_l[\![\psi]\!]^j_k \end{array}$$

For our example we have that ${}_l[\![\mathbf{F}p]\!]^0_2 = [\![\mathbf{F}p]\!]^0_2$ for each possible l. Thus, in this particular case a distinction between loop and loop-free prefixes is not necessary. For the overall encoding we get $[\![M(x_1), \mathbf{F}p]\!]_2 := [\![M(x_1)]\!]_2 \wedge [\![\mathbf{F}p]\!]^0_2$.

The general case requires to distinguish between loop and loop-free prefixes. For this, a subformula $T_{k,l}$ (similar to the transition encoding) is used that characterises all (k,l)-loops of the encoded structure. The general encoding is

$$[\![M(\vec[m]{x}), \psi]\!]_k := [\![M(\vec[m]{x})]\!]_k \wedge ([\![\psi]\!]^0_k \vee \bigvee_{l=0}^{k} (T_{k,l} \wedge {}_l[\![\psi]\!]^0_k))$$

Since $[\![M(\vec[m]{x}), \psi]\!]_k$ is not only defined over atoms but also over parameters from X and the constant $\bot$, standard satisfiability testing is not straightforwardly applicable. Thus, we define our new satisfiability problem SAT_{X3} for our propositional logic encoding, which reduces to multiple instances of classical SAT.

Definition 14 ($\mathbf{SAT}_{X3}$). *Let $F = [\![M(\vec{x}), \psi]\!]_k$ be a propositional logic encoding of a bounded PMC problem $[M(\vec{x}) \models_{E,k} \psi]$. Then*

$$\mathrm{SAT}_{X3}(F) := \begin{cases} \mathit{true} & \mathit{if}\ \forall (\vec{a}) \in \{t,f\}^m\ (\mathrm{SAT}([\![M(\vec{x}), \psi]\!]^p_k[(\vec{x})/(\vec{a})]) = \mathit{true}) \\ \mathit{false} & \mathit{if}\quad \mathrm{SAT}([\![M(\vec{x}), \psi]\!]^o_k) = \mathit{false} \\ \bot & \mathit{else} \end{cases}$$

where $[\![M(\vec{x}), \psi]\!]^p_k$ is the pessimistic completion *of $[\![M(\vec{x}), \psi]\!]_k$, i.e. the formula obtained by substituting all occurrences of $\bot$ by* false*, and $[\![M(\vec{x}), \psi]\!]^o_k$ is the* optimistic completion *obtained by substituting all occurrences of $\bot$ by* true*. Moreover, $[\![M(\vec{x}), \psi]\!]^p_k[(\vec{x})/(\vec{a})]$ denotes the substitution of $(\vec{x})$ in $[\![M(\vec{x}), \psi]\!]^p_k$ by $(\vec{a})$.*

Here $\mathrm{SAT}(F)$ returns *true* for a propositional logic formula F if there exists an assignment that makes the formula *true*, whereas it returns *false* if there does not exist such an assignment. Note that $\mathrm{SAT}([\![M(\vec{x}), \psi]\!]^o_k) = \mathit{false}$ is equivalent to $\forall (\vec{a}) \in \{t,f\}^m(\mathrm{SAT}([\![M(\vec{x}), \psi]\!]^o_k)[(\vec{x})/(\vec{a})] = \mathit{false})$. Hence, checking whether SAT_{X3} yields *false* requires a single SAT test only. The result of the overall SAT_{X3} test is equivalent to the result of the encoded model checking problem:

Theorem 2. *Let $M(\vec{x})$ be a parameterised three-valued Kripke structure over a set of atomic predicates AP, let ψ be an LTL$^+$ formula, and $k \in \mathbb{N}$. Then*

$$[M(\vec{x}) \models_{E,k} \psi] \;=\; \mathrm{SAT}_{X3}([\![M(\vec{x}), \psi]\!]_k)$$

Proof. See `http://www.cs.up.ac.za/cs/ntimm/ProofTheorem2.pdf`

Thus, bounded PMC can be reduced to multiple instances of SAT. For our example encoding $[\![M(x_1), \mathbf{F}p]\!]_2$ there exists a satisfying assignment for each possible instantiation which allows us to conclude that $[M(x_1) \models_{E,2} \mathbf{F}p] = \mathit{true}$.

5 Solving $\mathbf{SAT}_{X3}$ with Parallelisation and Clause Sharing

In this section we show how the SAT checks necessary to solve SAT_{X3} can be simultaneously performed by a parallel composition of solvers and additionally accelerated by *clause sharing*. SAT solvers require the input formula to be in conjunctive normal form (CNF). A CNF formula F over a set Boolean variables V is a conjunction of clauses, where each clause is a disjunction of literals. The *Tseitin transformation* [23] allows to translate any propositional formula into a SAT-equivalent CNF formula which length is linear in the size of the original formula. Thus, we assume that our encoding $F = [\![M(\vec{x}), \psi]\!]_k$ has been transformed into CNF, where F is defined over the indexed atoms of the encoding, the parameter set $X = \{x_1, \ldots, x_m\}$ and the constants *true*, *false* and $\bot$. Remember that checking SAT_{X3} requires to test the satisfiability of the pessimistic completion under each possible parameter instantiation. Instantiating the parameters of the encoding is equivalent to adding an *assumption* for each $x \in X$. Assumptions can be implemented by an assignment $\tau : X \to \{\mathit{true}, \mathit{false}\}$. By

$F\,|_\tau$ we denote the formula F under the assumption that the parameters are instantiated with regard to τ. The set of assumptions over X that we have to consider is

$$ASS = \{\bigcup_{i=1}^{m}\{(x_i, a_i)\} \mid a_i \in \{true, false\}\}$$

For our running example we get $ASS = \{\tau_1, \tau_2\}$ with $\tau_1 = \{(x_1, false)\}$ and $\tau_2 = \{(x_1, true)\}$. Checking SAT_{X3} requires to distinguish the cases where $\bot$ is assigned to *true* resp. *false*. Only for the pessimistic case $\bot = false$ we need to consider each possible parameter instantiation. We thus extend the domain of our assumptions to $X \cup \{\bot\}$ and introduce the pessimistic set of assumptions $ASS^p := \{\tau \cup \{(\bot, false)\} \mid \tau \in ASS\}$ and the optimistic assumption $\tau^o = \{(\bot, true)\}$. The problem SAT_{X3} for F can now be reformulated as follows

$$\mathrm{SAT}_{X3}(F) := \begin{cases} true & if\ \forall \tau \in ASS^p\ (\mathrm{SAT}(F\,|_\tau) = true) \\ false & if\ \mathrm{SAT}(F\,|_{\tau^o}) = false \\ \bot & else. \end{cases}$$

The number of SAT instances induced by SAT_{X3} is exponential in the number of parameters. Thus, in a sequential scenario parameterisation can lead to an exponential growth of computation time. Since all these instances are independent problems, SAT solving can be done concurrently. Provided that parallel processing is available to a sufficient extent, the runtime overhead of parameterisation can thus be significantly reduced. For our example we need three processors in order to entirely suspend the overhead induced by parameterisation:

$$\begin{array}{ll} & T_0\left(F\,|_{\{(\bot, true)\}}\right) \\ \| & T_1\left(F\,|_{\{(x_1, false),(\bot, false)\}}\right) \\ \| & T_2\left(F\,|_{\{(x_1, true),(\bot, false)\}}\right) \end{array}$$

Here T_0 to T_2 are solver threads executed concurrently that return whether the input formula is satisfiable or not. Note that it is not always necessary for all threads to terminate in order to solve SAT_{X3}. In case T_0 returns *false* we already know that the overall result is *false*, in case T_1 and T_2 return contrary results the overall result is $\bot$, and if T_1 and T_2 both return *true* then we also get *true* for SAT_{X3}.

This method can be additionally accelerated by exploiting the fact that the SAT instances associated with SAT_{X3} exhibit similarity in the sense of common subformulae. Modern SAT solvers are based on the search for a satisfying assignment of the input CNF formula by incrementally selecting unassigned variables, assigning them by either *true* or *false*, and propagating the resulting constraints to the clauses of the formula. In case the solver decisions lead to an unsatisfied clause, a so-called *conflict clause* is learned via resolution and added to the set of clauses. Moreover, the solver tracks back by revising a former assignment decision and continuing the search from this point until a satisfying assignment is found or the search space is entirely explored. Such a *conflict-driven clause*

learning (CDCL) [17] can help the solver to quickly prune certain branches of the search space and is thus crucial for the performance of satisfiability solving.

Learned clauses can also be *reused* or *shared*. In *incremental SAT solving* [9] a set of similar input formulae is processed by *consecutively* executed solvers. The inputs typically have a common subformula F while the differences are expressed by adding different assumptions $\tau_1, \ldots \tau_n$ to F. These assumptions are fixed assignments which are never revoked during the solving process. This guarantees that all conflict clauses learned in assumption-based incremental SAT solving inherently contain the assumptions they depend on. Hence, learned clauses can be reused without any restriction since changing from one assumption τ to another assumption τ' does not affect the clauses of F but automatically disables conflict clauses that are not compatible with τ'. In the context of *parallel* SAT solving the concept of *clause sharing* has been introduced [13, 11]. In this approach multiple copies of the input formula are checked concurrently. Generated conflict clauses can be shared, which enables to prune the search space at multiple points at the same time. Clause sharing has also been considered for parallel (non-parameterised) BMC [25, 2]. Here multiple solvers check the same BMC encoding but for different bounds. Similar to incremental SAT solving the differences are expressed via assumptions and learned clauses are shared via a global database or message passing.

For our *parameterised* scenario we adopt the concept of assumption-based clause sharing. We already reduced SAT_{X3} to multiple SAT instances consisting of the common formula F and different assumptions that characterise the possible parameter instantiations and the optimistic resp. pessimistic completion. Hence, all learned conflict clauses can be shared in a parallel scenario and thus used to prune the search space of multiple instances at the same time. Next, we will discuss the implementation of our approach and we will see how the runtime performance of parallel SAT-based bounded PMC can significantly profit from clause sharing.

6 Implementation and Experimental Results

We have prototypically implemented a SAT-based bounded LTL model checker for parameterised three-valued models which employs the Java-based solver library Sat4j [12]. The checker iterates over the bound k, starting with $k = 0$, until a definite result can be obtained or a predefined threshold for k is reached. In each iteration, after constructing a parameterised encoding F of a bounded PMC problem, each instance $F\,|_{\tau^o}, F\,|_{\tau_1}, \ldots, F\,|_{\tau_n}$ is processed by a solver thread. In the *basic* mode of our implementation the threads are executed in parallel whereby learned clauses are *not* shared. In the *enhanced* mode we additionally apply *clause sharing*. For this purpose, the solvers save copies of learned clauses in a global database D. The solvers provided by Sat4j inherently employ conflict-driven clause learning. Such CDCL solvers regularly *restart* while processing a SAT instance. Restarts typically happen after having learned a certain amount of clauses. For our clause sharing approach we use these restarts as the points

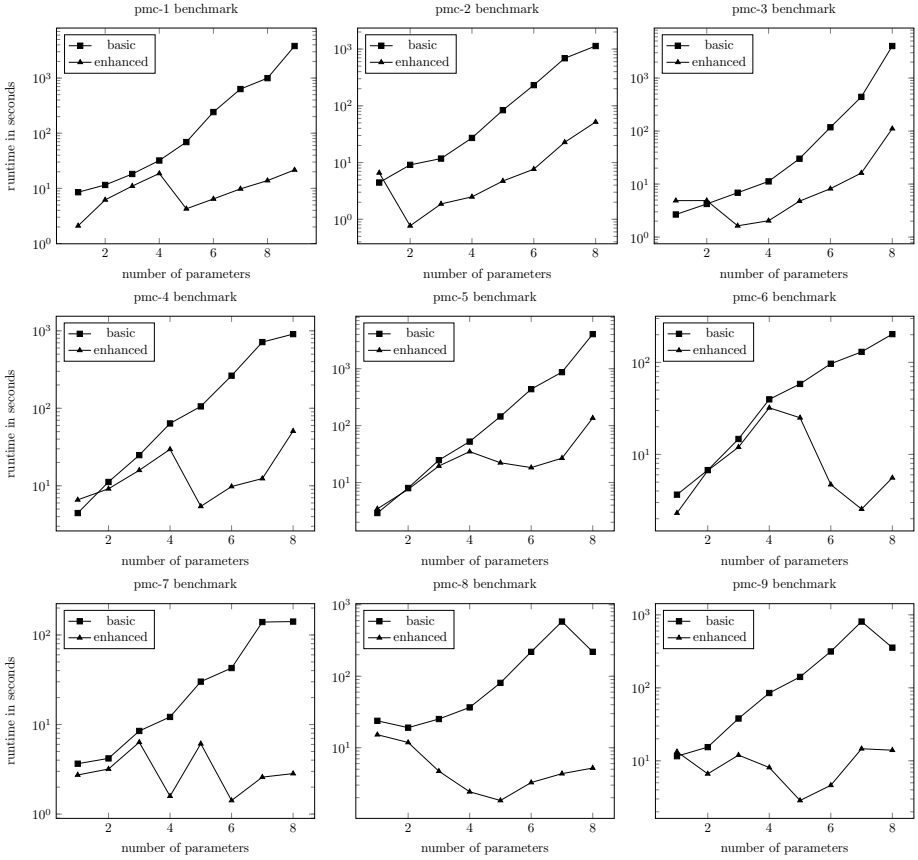
of synchronisation with D: Every time a solver restarts, it waits for read access to D. Then, it reads the clauses that have been placed into D by other solvers since its own most recent restart. (Remember that the assumption-based approach ensures that the solver will only make use of those clauses from D for pruning its search space, that were learned based on assumptions compatible to its own assumption τ.) Finally, the solver waits for exclusive read-write access to D and adds the clauses that have been learned by itself since its last restart. The shorter a conflict clause the stronger it prunes the search space. To keep the clause sharing mechanism efficient and the communication overhead caused by exclusively accessing D small, we currently only share *unit* (single-literal) clauses. In both the basic and the enhanced mode a k-bounded iteration terminates when the solver processing the optimistic completion of F returns *false*, or when all solvers processing an instance of the pessimistic completion return *true*, or as soon as the so far single results already indicate an overall *unknown* result. In the latter case, the $(k+1)$-bounded encoding is constructed and its instances are processed in the subsequent iteration. The following algorithm illustrates the general procedure of a single iteration. Here the input variable *mode* indicates whether each solver thread synchronises with the initially empty shared clause database D at a restart (*enhanced*) or not (*basic*).

Data: parameterised encoding F, assumption set $ASS = \{\tau^o, \tau_1, \ldots, \tau_n\}$, $mode \in \{basic, enhanced\}$, shared clause database $D = \varnothing$
Result: truth value of $SAT_{X3}(F)$
begin
  start new solver thread $T_0(F|_{\tau^o}, mode)$
  for $i = 1$ **to** n **do**
    start new solver thread $T_i(F|_{\tau_i}, mode)$
  end
  upon event
    T_0 returns *false* **do**
      return *false*
    T_1 to T_n return *true* **do**
      return *true*
    $\exists\, T_i, T_j$ where T_i returns *true* and T_j returns *false* **do**
      return $\perp$
  end
end

Algorithm 1: $SolveSAT_{X3}$

In our experiments we compared the runtime performance of the basic and the enhanced mode for encodings with increasing numbers of parameters. The items of our benchmark set correspond to parameterised three-valued BMC problems with fixed bounds that we encoded with our tool according to the definitions from Section 4. For each item *pmc-1* to *pmc-9* we considered variants with 1 to 8 parameters. The variants were constructed by applying different parameterisations to the Kripke structure of the underlying BMC problem. We transformed the SAT encodings into CNF. The resulting parameterised propositional logic

formulae consisted of up to 40000 variables and 150000 clauses. For all possible instantiations of a parameterised three-valued encoding the satisfiability was checked concurrently. Our experiments were conducted on a 2.6 GHz quad-core Intel Core i5 with 8 GB. The results for our benchmark set are shown in the diagrams below. Note that a *logarithmic* scale is used for the runtime axis. A table containing the numerical results of our experiments can be found online[1].



The items *pmc-1* to *pmc-3* represent PMC problems that yield *true* for each parameter variant (1 to 8). Hence, the termination of each variant necessitated that all solvers processing a parameter instance of the pessimistic completion returned *true*. Obtaining *true* results in SAT-based PMC generally involves the highest computational effort because of the high number of solvers that need to terminate. As shown by the diagrams, the runtime in the basic mode grows approximately exponentially with each additional parameter. For variants with a *small* number of parameters the runtimes of the basic and the enhanced mode are nearly on the same level. However, for variants with a *higher* number of parameters, i.e. a higher number of solvers that can share learned clauses, the enhanced

[1] `http://www.cs.up.ac.za/cs/ntimm/Table.pdf`

mode pays off: The solving time still grows with each additional parameter but is always orders of magnitude shorter then the basic approach runtime.

The items *pmc-4* to *pmc-6* represent PMC problems that yield *unknown* for the parameter variants 1 to 7 and *true* for the variant with 8 parameters. Hence, the last parameter variant finally brought the required precision for a definite result. The termination of the variants 1 to 7 necessitated that two solvers processing a parameter instance of the pessimistic completion returned contrary results (*true* and *false*). Also here we observed an exponential growth in runtime per additional parameter in the basic mode and a significantly better performance in the enhanced mode, provided that a certain number of parameters, i.e. cooperating solvers is present. In addition, we observed that solving the variant with 8 parameters in the enhanced mode involves an exceptional increase in runtime, which complies with the fact that obtaining a *true* result (variant 8) generally requires more computational effort than obtaining an *unknown* result (variants 1 to 7). This exceptional increase in runtime is not observable in the basic mode where computational costs are generally high. Also for variant 8 the enhanced approach with clause sharing is orders of magnitude faster than the basic approach.

The items *pmc-7* to *pmc-9* represent problems that yield *unknown* for the variants 1 to 7 and *false* for variant 8. Thus, the termination of variant 8 necessitated that the solver processing the optimistic completion returned *false*. For this set of items we made the same observations as for the previous benchmark items with regard to the general performance advantage of the enhanced mode. Additionally, we observed that solving variant 8 involved no significant increase or even a decrease in runtime. This is consistent with the fact that obtaining a *false* result in SAT-based PMC only requires the solver processing the optimistic completion to terminate with *false*.

In summary, our experiments showed that, regardless of the final verification result, clause sharing can considerably improve the runtime performance of parallel SAT-based PMC. The savings in solving time are particularly significant for variants with a higher number of parameters where solver cooperation in terms of sharing clauses is possible to a large extent. This enables us to benefit from the extra precision of parameterisation in three-valued model checking, without suffering from a substantial overhead in solving time.

7 Related Work

SAT-based BMC [4, 3] was originally introduced for the bounded evaluation of properties on *two-valued* models. Later, BMC has also been defined for *pure three-valued* models [24, 15]. A number of approaches have been proposed to accelerate SAT solving in general and SAT-based BMC in particular. *Conflict-driven clause learning* (CDCL) [17] is a concept for using clauses that were learned during the test of a single SAT instance for pruning its search space. In *incremental* SAT solving [9] a series of similar SAT instances is solved sequentially. The differences between the instances are expressed by considering the

same input formula under different assumptions. On this basis, clauses learned via CDCL can be reused when solving subsequent instances. *Parallel* SAT solving [5, 14, 18] is an approach to solve a single instance by dividing it into disjoint parts that are then processed by a set of solvers. In early works [5], CDCL was used by each solver for solely pruning its own search space. Later approaches [14, 18] considered parallel solving with *clause sharing*: Learned clauses are exchanged in order to increase the overall performance. In SAT-based BMC, CDCL has been applied in the sense that clauses learned for a k-bounded instance F_k are reused when solving the instance F_{k+1} [19, 1]. Also here assumptions are used to express the differences between F_k and F_{k+1}. This idea has been transferred to a parallel setting where solvers concurrently operate on BMC instances with different bounds, and synchronisation with a shared clause database happens at the restarts of the solvers [25, 2]. While these works deal with two-valued models, our approach considers parameterised three-valued models. Parameterisation opens another dimension for sharing clauses. We adopted assumption-based clause sharing and applied it between the *parameter instances* associated with PMC. Since these instances share large common subformulae, this enables us to accelerate the performance of SAT-based PMC.

8 Conclusion and Outlook

Parameterisation [22] is a concept for enhancing the precision of abstraction-based three-valued model checking by capturing facts in the model that can be derived from the software system to be verified. The application of parameterisation does not increase the state space but it leads to an exponential growth of the number of model checking runs. In this paper, we introduced a technique for PMC via parallel SAT solving. We defined a propositional logic encoding of bounded PMC problems and we proved that our encodings are sound in the sense that satisfiability results can be straightforwardly transferred to the encoded model checking problem. The number of SAT instances associated with an encoding is still exponential in the number of parameters. However, these instances exhibit considerable similarities in the sense of common subformulae. We showed that the concept of assumption-based clause sharing, which exploits such similarities in order to achieve a better performance in parallel SAT solving, can be transferred to our parameterised scenario. We presented a prototype tool for satisfiability-based PMC with clause sharing. In our experiments, we demonstrated that clause sharing leads to a significant acceleration of PMC. Thus, our new SAT-based approach enhances PMC as it allows us to profit from the extra precision of parametrisation in three-valued model checking, without creating a noticeable runtime overhead. Since we showed that PMC is reducible to multiple instances of classical SAT, our approach will also benefit from future improvements in SAT solver technology.

So far, we conducted our experiments on a single-processor system with multiple cores. We already achieved substantial runtime savings with our clause sharing approach by using these very limited parallel computing resources. As

future work we plan more extensive experiments on a cluster for parallel computing. Moreover, we intend to extend parallelisation and clause sharing to the satisfiability checks for the different *bounds* $k = 0, 1, 2, \ldots$ in bounded PMC. We also plan to experiment with different *policies* with regard to size constraints of clauses to be shared in order to discover the best trade-off between the communication costs due to sharing and the speed-up due to additional pruning. Moreover, we intend to use *multiple copies* of each instantiation in the parallel composition of solver threads. We expect that this allows to increase the amount of clauses that can be shared and thus leads to a further acceleration of the overall solving time. Another direction for future research is to summarise the exponential number of SAT instances of the pessimistic completion to a single instance of a *quantified boolean formula* (QBF) $F = \forall\, x_1 \ldots \forall\, x_m \,\exists\, v_1 \ldots \exists\, v_n \left(F\,|_{(\bot, false)}\right)$ and then to check F via a QBF solver. Though QBF is PSPACE-complete, whereas SAT is only NP-complete, it would be interesting to study whether the reduction to a single instance of a higher complexity class pays off in terms of solving performance for particular PMC problems.

References

1. Abrahám, E., Becker, B., Klaedtke, F., Steffen, M.: Optimizing bounded model checking for linear hybrid systems. In: Verification, Model Checking, and Abstract Interpretation. pp. 396–412. Springer (2005)
2. Ábrahám, E., Schubert, T., Becker, B., Fränzle, M., Herde, C.: Parallel sat solving in bounded model checking. Journal of Logic and Computation 21(1), 5–21 (2011)
3. Biere, A.: Bounded model checking. In: Handbook of Satisfiability, pp. 457–481 (2009)
4. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. Springer (1999)
5. Böhm, M., Speckenmeyer, E.: A fast parallel sat-solver?efficient workload balancing. Annals of Mathematics and Artificial Intelligence 17(2), 381–400 (1996)
6. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: Halbwachs, N., Peled, D. (eds.) CAV 1999. pp. 274–287. LNCS, Springer-Verlag Berlin Heidelberg, London, UK (1999)
7. Bruns, G., Godefroid, P.: Generalized model checking: Reasoning about partial state spaces. In: Palamidessi, C. (ed.) CONCUR 2000 - Concurrency Theory, Lecture Notes in Computer Science, vol. 1877, pp. 168–182. Springer-Verlag Berlin Heidelberg (2000)
8. Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-valued symbolic model-checking. ACM Transactions on Software Engineering and Methodology (TOSEM) 12(4), 371–408 (2003)
9. Eén, N., Sörensson, N.: Temporal induction by incremental sat solving. Electronic Notes in Theoretical Computer Science 89(4), 543–560 (2003)
10. Fitting, M.: Kleene's three valued logics and their children. Fundamenta Informaticae 20(1-3), 113–131 (Mar 1994)
11. Hamadi, Y., Jabbour, S., Sais, J.: Control-based clause sharing in parallel sat solving. In: Autonomous Search, pp. 245–267. Springer (2012)
12. Le Berre, D., Parrain, A., et al.: The sat4j library, release 2.2, system description. Journal on Satisfiability, Boolean Modeling and Computation 7, 59–64 (2010)

13. Lewis, M., Schubert, T., Becker, B.: Speedup techniques utilized in modern sat solvers. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science, vol. 3569, pp. 437–443. Springer Berlin Heidelberg (2005)
14. Schubert, T., Lewis, M., Becker, B.: Pamira-a parallel sat solver with knowledge sharing. In: Microprocessor Test and Verification, 2005. MTV'05. Sixth International Workshop on. pp. 29–36. IEEE (2005)
15. Schuele, T., Schneider, K.: Three-valued logic in bounded model checking. In: Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design. pp. 177–186. IEEE Computer Society (2005)
16. Shoham, S., Grumberg, O.: 3-valued abstraction: More precision at less cost. Information and Computation 206(11), 1313–1333 (2008)
17. Silva, J.P.M., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability, pp. 131–153 (2009)
18. Sinz, C., Blochinger, W., Küchlin, W.: Pasat?parallel sat-checking with lemma exchange: Implementation and applications. Electronic Notes in Discrete Mathematics 9, 205–216 (2001)
19. Strichman, O.: Accelerating bounded model checking of safety properties. Formal Methods in System Design 24(1), 5–24 (2004)
20. Timm, N.: Bounded Model Checking für partielle Systeme. Master's thesis, University of Paderborn (2009)
21. Timm, N.: Three-Valued Abstraction and Heuristic-Guided Refinement for Verifying Concurrent Systems. Phd thesis, University of Paderborn (2013)
22. Timm, N., Gruner, S.: Parameterisation of three-valued abstractions. In: Braga, C., Mart-Oliet, N. (eds.) Formal Methods: Foundations and Applications, pp. 162–178. Lecture Notes in Computer Science, Springer International Publishing (2015)
23. Tseitin, G.: On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning, pp. 466–483. Symbolic Computation, Springer-Verlag Heidelberg (1983)
24. Wehrheim, H.: Bounded model checking for partial Kripke structures. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) Theoretical Aspects of Computing - ICTAC 2008, Lecture Notes in Computer Science, vol. 5160, pp. 380–394. Springer-Verlag Berlin Heidelberg (2008)
25. Wieringa, S., Niemenmaa, M., Heljanko, K.: Tarmo: A framework for parallelized bounded model checking. arXiv preprint arXiv:0912.2552 (2009)