MCMT: A Model Checker Modulo Theories

Silvio Ghilardi[1] and Silvio Ranise[2]

[1] Dipartimento di Informatica, Università degli Studi di Milano (Italia)
[2] FBK-Irst, Trento (Italia)

Abstract. We describe MCMT, a fully declarative and deductive symbolic model checker for safety properties of infinite state systems whose state variables are arrays. Theories specify the properties of the indexes and the elements of the arrays. Sets of states and transitions of a system are described by quantified first-order formulae. The core of the system is a backward reachability procedure which symbolically computes pre-images of the set of unsafe states and checks for safety and fix-points by solving Satisfiability Modulo Theories (SMT) problems. Besides standard SMT techniques, efficient heuristics for quantifier instantiation, specifically tailored to model checking, are at the very heart of the system. MCMT has been successfully applied to the verification of imperative programs, parametrised, timed, and distributed systems.

1 Introduction

In [6], we have presented a fully declarative approach to verify safety properties of infinite state systems—whose variables are arrays—by backward reachability. Such systems can be used as suitable abstractions of many classes of systems ranging from parametrised protocols to sequential programs manipulating arrays. The idea is to use classes of quantified first-order formulae to represent an infinite set of states of the system so that the computation of pre-images boils down to symbolic manipulations. Using suitable theories over the elements and the indexes of the arrays, we are able to declaratively specify both the data manipulated by the system and its topology (in the case of parametrised systems) or properties of the indexes of arrays (in the case of imperative programs).

In the framework of [6], the key to mechanize backward reachability is to reduce the checks for fixed-point and safety to Satisfiability Modulo Theories (SMT) problems of first-order formulae containing (universal) quantifiers. Under suitable hypotheses on the theories over the indexes and the elements of the arrays, these SMT problems are decidable [6] by integrating a quantifier instantiation procedure with SMT solving techniques for quantifier-free formulae. In [8,9], we described heuristics to reduce the number of quantified variables and—most importantly—of instances while preserving the completeness of SMT solving. Unfortunately, the decidability of safety and fixed-point checks is not yet enough to ensure the termination of the backward reachability analysis. Theoretically, termination for this procedure can be ensured for well-structured systems [1]; in [6], we explained how to recast this notion in our framework so as

to import all the decidability results available in the literature. Pragmatically, it is well-known that termination of backward reachability can be obtained by using invariants and we discussed how natural it is to guess and use them in our framework and also gave a characterization of the completeness of the proposed method [7].

In this paper, we give the first comprehensive high-level description of MCMT v. 1.0, a significant extension of the prototype tool used in our previous work [7,8]. MCMT v. 1.0 uses Yices (`http://yices.csl.sri.com`) as the back-end SMT solver and is available at `http://www.dsi.unimi.it/~ghilardi/mcmt`. Besides various ameliorations and refinements to previously available functionalities as well as new utilities (like Bounded Model-Checking), MCMT v. 1.0 supports the following new features, which widen its scope of applicability and greatly improve its performances (in particular, w.r.t. termination) when used with care: (i) transitions with existentially quantified variables ranging over data values (and not only indexes), provided that the theory over data admits elimination of quantifiers; (ii) synthesis of invariants with two universally quantified variables (previously [7], they were limited to containing just one variable); (iii) a form of predicate abstraction, called *signature abstraction*, together with limited support for the acceleration of transitions [4]. For lack of space, only an excerpt of the experiments are described here (for full details, consult the MCMT web-page).

2 The MCMT way of life

We present our vision of model checking infinite state systems underlying MCMT. To this end, we believe it is convenient to recall two distinct and complementary approaches among the many possible alternatives available in the literature.

The first approach is pioneered in [1] and its main notion is that of well-structured system. Recently, it was implemented in two systems [2,3], which were able to automatically verify several protocols for mutual exclusion and cache coherence. One of the key ingredients to the success of these tools is their capability to perform accurate fixed-point checks so as to reduce the number of iterations of the backward search procedure. A fixed-point check is implemented by 'embedding' an old configuration (i.e. a finite representation of a potentially infinite set of states) into a newly computed pre-image; if this is the case, then the new pre-image is considered "redundant" (i.e., not contributing new information about the set of backward reachable states) and thus can be discarded without loss of precision. Indeed, the exhaustive enumeration of embeddings has a high computational cost. Furthermore, constraints are only used to represent the data manipulated by the system while its topology is encoded by *ad hoc* data structures. A change in the topology of the system requires the implementation from scratch of algorithms for both pre-image and embedding computation. On the contrary, MCMT uses particular classes of *first-order formulae* to represent configurations parametrised with respect to two theories, one for data and one for the topology so that pre-image computation reduces to a fixed set of logical manipulations and fixed-point checking to solve SMT problems containing uni-

versally quantified variables. To mechanize these tests, a quantifier-instantiation procedure is used, which is the logical counterpart of the enumeration of embeddings. Interestingly, this notion of embedding can be recaptured (via classical model theory) [6] in the logical framework underlying MCMT, a fact that allows us to import the decidability results of [1] for backward reachability. Another important advantage of the approach underlying MCMT over that proposed in [1] is its broader scope of applications with respect to the implementations in [2,3]. The use of theories for specifying the data and the topology allows one to model disparate classes of systems in a natural way. Furthermore, even if the quantifier instantiation procedure becomes incomplete with rich theories, it can soundly be used and may still permit the proof of the safety of a system. In fact, MCMT has been successfully employed to verify sequential programs (such as sorting algorithms) that are far beyond the reach of the systems described in [2,3].

The second and complementary approach to model checking infinite state systems relies on *predicate abstraction* techniques, initially proposed in [10]. The idea is to abstract the system to one with finite states, to perform finite-state model checking, and to refine spurious traces (if any) by using decision procedures or SMT solvers. This technique has been implemented in several tools and is often combined with interpolation algorithms for the refinement phase. As pointed out in [5,11], predicate abstraction must be carefully adapted when (universal) quantification is used to specify the transitions of the system or its properties, as it is the case for the problems tackled by MCMT. There are two crucial problems to be solved. The first is to find an appropriate set of predicates to compute the abstraction of the system. In fact, besides system variables, universally quantified variables may also occur in the system. The second problem is that the computation of the abstraction as well as its refinement reduce to solving proof obligations containing universal quantifiers. Hence, we need to perform suitable quantifier instantiations in order to enable the use of decision procedures or SMT solving techniques for quantifier-free formulae. The first problem is solved by Skolemization [5] or fixing the number of variables in the system [11] so that standard predicate abstraction techniques can still be used. The second problem is solved by adopting very straightforward (sometimes naive) and incomplete quantifier instantiation procedures. While being computationally cheap and easy to implement, the heuristics used for quantifier instantiation are largely imprecise and do not permit the detection of redundancies due to variable permutations, internal symmetries, and so on. Experiments performed with MCMT, tuned to mimic these simple instantiation strategies, show much poorer performance. We believe that the reasons of success of the predicate abstraction techniques in [5,11] lie in the clever heuristics used to find and refine the set of predicates for the abstraction. The current implementation of MCMT is orthogonal to the predicate abstraction approach; it features an extensive quantifier instantiation (which is complete for some theories over the indexes and is enhanced with completeness preserving heuristics to avoid useless instances), but it performs only a primitive form of predicate abstraction, called signature abstraction (see Section 4). Another big difference is how abstraction is

used in MCMT: the set of backward reachable states is always computed precisely while abstraction is only exploited for guessing candidate invariants which are then used to prune the set of backward reachable states. Since we represent sets of states by formulae, guessing and then using the synthesized invariants turns out to be extremely easy, thereby helping to solve the tension between model checking and deductive techniques that has been discussed a lot in the literature and is still problematic in the tools described in [2,3] where sets of states are represented by *ad hoc* data structures. We plan to enhance predicate abstraction techniques in future releases of MCMT, so as to find the best trade-off between the advantages of predicate abstraction and extensive quantifier instantiation.

3 The Input language for Safety Problems

The input language of MCMT can be seen as a parametrised extension of the one used by UCLID (`http://www.cs.cmu.edu/~uclid`). Formally, it is a sub-set of multi-sorted first-order logic, extended with the ternary expression constructor "if-then-else" (which is standard in the SMT-LIB format). For lack of space, we omit the presentation of the concrete syntax which is fully described in the on-line User Manual.

Sorts. We use the following distinguished sorts: Ind for indexes, $Elem_1, ..., Elem_m$ for elements of arrays, and $Arr_1, ..., Arr_m$ for array variables (where Arr_k corresponds to arrays of elements of sort $Elem_k$, for $k = 1, ..., m$).

Theories. We assume that the mono-sorted theories T_I and T_{E_k} are given over the sorts Ind and $Elem_k$, respectively, for $k = 1, ..., m$. The three-sorted theories $A_I^{E_k}$ are obtained as the combination of the theories T_I and T_{E_k} for each $k = 1, ..., m$ by adding the sort Arr_k to Ind and $Elem_k$, by taking the union of the symbols of T_I and T_{E_k}, and by adding the binary symbol $\_[\_]_k : Arr_k \times Ind \rightarrow Elem_k$ for reading the content of an array at a given index (the subscript k is omitted if clear from the context). Finally, we let $A_I^E := \bigcup_{k=1}^m A_I^{E_k}$.

Formats of formulae. We use two classes of formulae to describe sets of states: $\forall \underline{i}.\phi(\underline{i}, \underline{a})$ and $\exists \underline{i}.\phi(\underline{i}, \underline{a})$, where $\underline{i}$ is a tuple of variables of sort Ind, $\underline{a}$ is a tuple of length m of array variables of sorts $Arr_1, ..., Arr_m$, and ϕ is quantifier-free formula containing at most the variables in $\underline{i} \cup \underline{a}$ as free variables. The former are called $\forall^I$-formulae and the latter $\exists^I$-formulae. An $\exists^I$-formula $\exists \underline{i}.\phi$ is *primitive* when ϕ is a conjunction of literals; it is *differentiated* when it is primitive and ϕ contains as a conjunct the disequation $i_k \neq i_l$ for each $1 \leq k < l \leq length(\underline{i})$. By applying simple logical manipulations, it is always possible to transform any $\exists^I$-formula into a disjunction of primitive differentiated ones. To specify transitions, we use a particular class of formulae (called *transition formulae*) corresponding to a generalization of the usual notion of guarded assignment system:

$$\exists i_1, i_2, e. \left(G(i_1, i_2, e, \underline{a}) \ \wedge \ \bigwedge_{k=1}^{m} \forall j.\, a'_k[j] = Upd_k(j, i_1, i_2, e, \underline{a}) \right),$$

where i_1, i_2 are variables of sort Ind (having at most two existentially quantified variables is not too restrictive since many disparate systems can be formalized

in this format as shown by the experiments available on-line), e is a variable of sort $Elem_k$ (for some $k = 1..., m$), $\underline{a}$ is a tuple of array state variables, a_k (in $\underline{a}$) is the actual value of a state variable and a'_k is its value after the execution of the transition, G is a conjunction of literals (called the *guard*), and Upd_k is a function defined by cases (for $k = 1, ..., m$), i.e. by suitably nested if-then-else expressions whose conditionals are again conjunctions of literals. The format for transition formulae above—because of the presence of the existentially quantified variable e over data values—is the first significant amelioration of the actual version of MCMT as it allows one to specify classes of systems which were not previously accepted by the tool such as real time systems or those with non-deterministic updates. Notice that the theory T_{E_k} over the sort $Elem_k$ of the variable e must be Linear Arithmetic (over the integers or the reals). This limitation allows us to maintain the closure of the class of $\exists^I$-formulae under pre-image computation by exploiting quantifier elimination (implemented only in the latest version of MCMT).

Safety problem. Let I be a $\forall^I$-formula describing the set of initial states, Tr a finite set of transition formulae, and U an $\exists^I$-formula for the set of unsafe states. The *safety problem* solved by MCMT consists in establishing whether there exists an $n \geq 0$ such that the formula

$$I(\underline{a}^0) \wedge \tau(\underline{a}^0, \underline{a}^1) \wedge \cdots \wedge \tau(\underline{a}^{n-1}, \underline{a}^n) \wedge U(\underline{a}^n) \tag{1}$$

is A_I^E-satisfiable, where $\underline{a}^h = a_1^h, ..., a_m^h$ for $h = 0, ..., n$, and $\tau := \bigvee_{\tau_i \in Tr} \tau_i$. If there is no such n, then the system is *safe* (w.r.t. U); otherwise, it is said to be *unsafe* since the A_I^E-satisfiability of (1) implies the existence of a run (of length n) leading the system from a state in I to a state in U.

4 The main loop: deductive backward reachability

MCMT implements backward reachability to solve safety problems. For $n \geq 0$, the *n-pre-image* of an $\exists^I$-formula $K(\underline{a})$ is $Pre^0(\tau, K) := K$ and $Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K))$, where $Pre(\tau, K) := \exists \underline{a}'.(\tau(\underline{a}, \underline{a}') \wedge K(\underline{a}'))$. It is easy to show [6] that the class of $\exists^I$-formulae is closed under pre-image computation under the assumption that T_{E_k} admits elimination of quantifiers (if an existentially quantified variable of sort $Elem_k$ occurs in a transition formula). The formula $BR^n(\tau, U) := \bigvee_{i=0}^n Pre^i(\tau, U)$ represents the set of states which are backward reachable from the states in U in at most $n \geq 0$ steps. So, backward reachability consists of computing $BR^n(\tau, U)$ for increasing values of n and checking whether $BR^n(\tau, U) \wedge I$ is A_I^E-satisfiable or $\neg(BR^n(\tau, U) \rightarrow BR^{n-1}(\tau, U))$ is A_I^E-unsatisfiable. In the first case (*safety* test), one concludes the unsafety of the system while in the second (*fixed-point* test), it is possible to stop computing pre-images as no new states can be reached and, if the safety test has been passed, one can infer the safety of the system.

Figure 1 introduces the Tableaux-like calculus used by MCMT to implement backward reachability [7]. We initialize the tableau with the $\exists^I$-formula $U(\underline{a})$ representing the set of unsafe states. The computation of the pre-image is realized

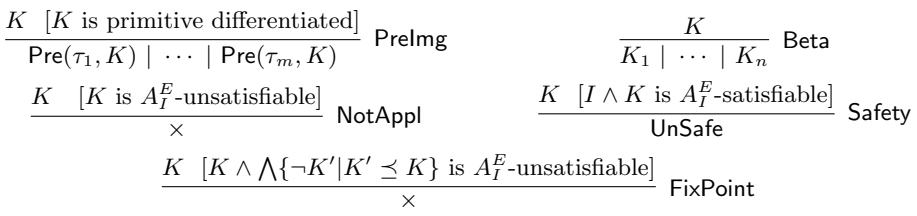

Fig. 1. The calculus underlying MCMT

by applying rule PreImg (we use square brackets to indicate the applicability condition of a rule), where $\mathsf{Pre}(\tau_h, K)$ computes the $\exists^I$-formula which is logically equivalent to $Pre(\tau_h, K)$. Since the $\exists^I$-formulae labeling the consequents of the rule PreImg may not be primitive and differentiated (because of nested if-then-else expressions and incompleteness of variable distinction), we need to apply the Beta rule to an $\exists^I$-formula so as to eliminate the conditionals by case-splitting and derive $K_1, \ldots, K_n$ primitive differentiated $\exists^I$-formulae whose disjunction is A_I^E-equivalent to K. By repeatedly applying PreImg and Beta, it is possible to build a tree whose nodes are labelled by $\exists^I$-formulae whose disjunction is equivalent to $BR^n(\tau, U)$ for some $n \geq 0$. Indeed, there is no need to fully expand the tree; it is useless to apply the rule PreImg to a node ν labelled by an A_I^E-unsatisfiable $\exists^I$-formula (rule NotAppl). One can terminate the whole search because of the safety test (rule Safety), in which case one can extract from the branch a *bad trace*, i.e. a sequence of transitions leading the array-based system from a state satisfying I to one satisfying U. A branch can be terminated by the fixed-point test described by rule FixPoint, where $K' \preceq K$ means that K' is a primitive differentiated $\exists^I$-formula labeling a node preceding the node labeled by K (nodes can be ordered according to the strategy for expanding the tree).

For the effectiveness of rules Safety and FixPoint, it is necessary to check the A_I^E-satisfiability of $\exists^I\forall^I$-formulae, i.e. formulae containing an alternation of quantifiers over variables of sort *Ind*. In MCMT, we have integrated a quantifier instantiation procedure with SMT solving techniques for quantifier-free formulae, augmented with heuristics to avoid the generation of useless instances and incrementality of satisfiability checks [8]. The technique is complete under some hypotheses on T_I and the T_{E_k}'s [6]. Even if such hypotheses are not satisfied, the instantiation procedure can still be soundly used without loss of precision for a final safety result of the backward reachability procedure, although its termination is less guaranteed. This point is particularly delicate and merits some discussion. Consider the verification of a mutual exclusion protocol (due to Szymanski) which can be considered as a typical example of problems about parametrised systems. To show the safety of this problem, MCMT generates 1153 satisfiable and 4043 unsatisfiable SMT problems. While our quantifier instantiation procedure with Yices is capable of solving all SMT problems, Yices alone with its quantifier handling techniques returns 'unknown' on all the 1153 satisfiable instances while it can solve 2223 of the unsatisfiable ones and returns 'unknown' on the remaining 1820. We are currently adding the capability of gen-

erating satisfiability problems in the SMT-LIB format to MCMT so as to evaluate the quantifier handling procedures available in various SMT solvers.

The main novelty of the latest version of MCMT is the more extensive support for invariant synthesis, abstraction, and *acceleration* for computing the repeated application of a sub-set of the transitions an arbitrary number of times in a single step. Invariant synthesis has been introduced in [7] but the implementation was able to generate universally quantified invariants with just one variable. MCMT v. 1.0 supports the generation of invariants with up to two universal quantifiers. While performing backward reachability (which is always precise), candidate invariants are guessed according to some heuristics [7] and then a resource bounded (secondary) backward reachability is used to keep or discard the candidates. The invariants found in this way are used in the main backward reachability procedure when checking for fix-points. The present version of MCMT features also a new technique for the synthesis of invariants, named *signature abstraction*: it consists of projecting away (by quantifier elimination, whenever possible) those literals containing a sub-set of the array variables; thereby obtaining an over-approximation of the set of reachable states. This is without loss of precision, since it is done in the secondary backward reachability procedure for invariant synthesis while the main procedure continues to compute the set of backward reachable states precisely. The last novelty of MCMT v. 1.0 is some support for acceleration along the lines of [4] in the hope of a better convergence of the backward reachable procedure; this is often the case for systems formalized by arithmetic constraints such as Petri nets.

5 Experiments

To show the flexibility and the performance of MCMT, we have taken some pain to build a library of benchmarks in the format accepted by our tool by translating safety problems from a variety of sources, such as the distributions of the infinite state model checkers described in [2,3] or imperative programs manipulating arrays taken from standard books about algorithms. For lack of space, we include here only an excerpt of the experiments (see the tool web-page for a full report).

We divide the problems in four categories: mutual exclusion (M) and cache coherence (C) protocols, imperative programs manipulating arrays (I), and heterogeneous (H) problems. We tried the tool in two configurations: the "Default Setting" is when MCMT is invoked without any option and the "Best Setting" is when the tool is run with some options turned on. In Table 1, the column 'd' is the depth of the tableaux obtained by applying the rules in Figure 1, '#n' is the number of nodes in the tableaux, '#del' is the number of subsumed nodes, '#SMT' is the number of invocations to Yices, '#i' is the number of invariants found by MCMT (for the "Default Setting," column '#i' is not shown because MCMT's default is to turn off invariant synthesis), and 'time' is the total amount of time (in seconds) taken by the tool to solve the problem on a Pentium Intel 1.73 GHz with 1 Gb Sdram running Linux Gentoo. In the cases when the tool seemed to diverge, we aborted execution, and put 't(ime)o(out)' in the column

Table 1. Some experimental results

	Default setting					Best setting					
Problem	d	#n	#del	#SMT	time	d	#n	#del	#SMT	#i	time
Lamport (M)	23	913	242	47574	120.62	23	248	42	19254	7	32.84
RickAgr (M)	13	458	119	35355	187.04	13	458	119	35355	0	187.04
Szymanski_at (M)	23	1745	311	424630	540.19	9	22	10	2987	42	1.25
German07 (C)	26	2442	576	121388	145.68	26	2442	576	121388	0	145.68
GermanBug (C)	16	1631	203	41497	49.70	16	1631	203	41497	0	49.70
GermanPFS (C)	33	11605	2755	858184	1861.0	33	11141	2673	784168	149	1827.0
SelSort (I)	-	-	-	-	to	5	13	2	1141	11	0.62
Strcat (I)	-	-	-	-	to	2	2	2	80	2	0.07
Strcmp (I)	-	-	-	-	to	2	1	1	21	3	0.01
Fischer (H)	10	16	2	336	0.16	10	16	2	336	0	0.16
Ticket (H)	-	-	-	-	to	3	4	2	201	10	0.06

of timings and leave the others empty ('-'). All systems—except 'GermanBug' (a bugged version of 'German07')—are certified to be safe by MCMT while for 'GermanBug,' the tool returns an error trace consisting of 16 transitions. Invariant synthesis (especially the signature abstraction technique introduced in this version of the tool) is helpful to reduce the solving time for problems in (M), and to obtain termination for those in (I), but has no effect on problems in (C).

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
2. P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers. In *TACAS*, volume 4424 of *LNCS*, pages 721–736, 2007.
3. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, LNCS, pages 145–157, 2007.
4. B. Bérard and L. Fribourg. Reachability Analysis of (Timed) Petri Nets Using Real Arithmetic. In *CONCUR*, LNCS, pages 178–193, 1999.
5. C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *POPL*, pages 191–202. ACM, 2002.
6. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT Model-Checking of Array-based Systems. In *IJCAR*, LNCS, 2008.
7. S. Ghilardi and S. Ranise. Goal Directed Invariant Synthesis for Model Checking Modulo Theories. In *TABLEAUX*, LNAI, pages 173–188. Springer, 2009.
8. S. Ghilardi and S. Ranise. Model Checking Modulo Theory at work: the intergration of Yices in MCMT. In *AFM (co-located with CAV09)*, 2009.
9. S. Ghilardi, S. Ranise, and T. Valsecchi. Light-Weight SMT-based Model-Checking. In *AVOCS 07-08*, ENTCS, 2008.
10. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *CAV*, volume 1254 of *LNCS*. Springer, 1997.
11. S. K. Lahiri and R. E. Bryant. Predicate abstraction with indexed predicates. *ACM Transactions on Computational Logic (TOCL)*, 9(1), 2007.