SMT-Based Bounded Model Checking for Multi-threaded Software in Embedded Systems

Lucas Cordeiro
Advisor: Bernd Fischer
ECS, University of Southampton
lcc08r@ecs.soton.ac.uk

ABSTRACT

The transition from single-core to multi-core processors has made multi-threaded software an important subject over the last years in computer-aided verification. Model checkers have been successfully applied to discover subtle errors, but they suffer from combinatorial state space explosion when verifying multi-threaded software. In our previous work, we have extended the encodings from SMT-based bounded model checking (BMC) to provide more accurate support for program verification and to use different background theories and solvers in order to improve scalability and precision in a completely automatic way. We now focus on extending this work to support an SMT-based BMC formulation of multi-threaded software which allows the state space to be reduced by abstracting the number of state variables and interleavings from the proof of unsatisfiability generated by the SMT solvers. The core idea of our approach aims to extract the proof objects produced by the SMT solvers in order to control the number of interleavings and to remove logic that is not relevant to a given property. This work aims to develop a new algorithmic method and corresponding tools based on SMT to verify embedded software in multi-core systems.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Model checking

General Terms

Computer-Aided Verification

Keywords

Formal Software Verification, SAT Modulo Theories, Multi-core systems

1. INTRODUCTION

Embedded computer systems are used in a wide range of sophisticated applications, such as mobile phones or set-top boxes providing internet connectivity. Multi-core processors with scalable shared memory are becoming very popular in embedded applications that require high computational power. In addition, the functionality demanded for embedded systems has increased significantly and an increasing number of functions are implemented in software rather than hardware. As a consequence, the verification of the software design and the correctness of its implementation have become increasingly difficult.

Bounded model checking (BMC) has been successfully applied to verify embedded software and discovered subtle errors in real designs [4]. BMC generates verification conditions (VCs) that reflect the exact path in which a statement is executed, the context in which a given function is called, and the bit-accurate representation of the expressions. Proving the validity of these VCs remains the main performance bottleneck in verifying large embedded software, despite attempts to cope with increasing system complexity by applying SMT (Satisfiability Modulo Theories) [3, 8, 10] solvers.

Recently, there have been attempts to extend BMC to the verification of multi-threaded software [11, 13, 14, 18]. The main challenge remains the classical state space explosion problem in which the number of interleavings grows exponentially with the number of threads and program statements. An important observation is that on one hand BMC finds counter-examples very quickly using the SMT solvers [8] and on the other hand the SMT solvers produce unsatisfiable cores that allow us to remove logic that is not relevant to a given property [16].

Grumberg et al. [12] propose an algorithmic method based on Boolean Satisfiability (SAT) and BMC to model check a multi-process system based on a series of under-approximated models. They realized that the unsatisfiable cores generated by the SAT solvers can also be used to control the number of allowed interleavings of the given set of processes. However, this method does not remove all redundant interleavings and is incomplete since it tries to prove or disprove the property up to a given length k. Grumberg et al. apply this method to check *reachability properties* of a leader election protocol, but they do not consider the verification of low-level embedded software in multi-core systems.

In our work, we plan to develop a strategy that aims to mitigate the complexity problems to model check embedded software in multi-core environments. In particular, our method aims to focus on embedded programs because they have characteristics that make model checking attractive, e.g., dynamic memory allocations and recursion are discouraged. On the other hand, embedded programs are reac-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE '10, May 2-8 2010, Cape Town, South Africa
Copyright 2010 ACM 978-1-60558-719-6/10/05 ...$10.00.

tive (i.e., do not terminate as opposed to most application software) and we need to ensure that the loops terminate for correctness. Consequently, we aim to extend the under-approximation and widening (UW) algorithm proposed in [12] with the purpose of addressing the verification of real-world embedded software in multi-core systems using different background theories and SMT solvers. We intend to propose a comprehensive SMT-based BMC procedure using interpolation methods [15] and investigate partial order reduction algorithms in the UW framework [1].

2. RELATED WORK

SMT-based BMC is gaining popularity in the formal verification community due to the advent of sophisticated SMT solvers built over efficient SAT solvers [9]. Ganai and Gupta describe a verification framework for BMC which extracts high-level design information from an extended finite state machine (EFSM) and apply several techniques to simplify the BMC problem [10]. However, the authors use only the theory of integer and real arithmetic, which does not reflect precisely the ANSI-C semantics. Armando et al. also propose a BMC approach using SMT solvers for ANSI-C programs [3], but they only make use of linear arithmetic, arrays, records and restricted bit-vectors arithmetic and, as a consequence, their SMT-CBMC prototype does not address important constructs of the ANSI-C language.

Qadeer and Rehof present a pragmatic method to discover bugs in concurrent software in which the program analysis is restricted to executions with a bounded number of context switches [17]. However, this method is incomplete since it considers the verification up to a given fixed context bound. In addition, the authors do not apply it to realistic and large concurrent software benchmarks and the integration of this context-bounded model checking algorithm into the explicit state model checker ZING [2] is left for future work. Rabinovitz and Grumberg describe an extension of CBMC to concurrent C programs [18], which translates C threads into static single assignment (SSA) form and adds constraints for a bounded number of context-switches (as described in [2]). This approach, however, requires the encoding of all interleavings into a single formula to be sent to a SAT solver.

Ganai and Gupta describe a lazy method for modelling multi-threaded concurrent systems using shared variables [11], but this method is restricted to two threads. Gupta et al. [14] extend [11, 13] by supporting more than two threads and by combining dynamic partial order reduction with symbolic state space exploration. However, this method is incomplete since it considers the concurrency semantics up to the bounded depth as in [2, 18]. Grumberg et al. propose an algorithmic method based on SAT and BMC to model check a multi-process system based on a series of under-approximated models [12]. This approach, however, does not integrate partial order reduction algorithms to reduce redundant interleavings and it does not address the problem of model checking real-world embedded software in multi-core environments.

To the best of our knowledge, there is no work that considers a complete SMT-based BMC formulation to verify real-world embedded software in multi-core systems using a set of under-approximations and widening models as well as the integration of partial order reduction algorithms into the UW framework. As a consequence, our main contribution is an algorithmic method to extract the unsatisfiable core produced by the SMT solvers in order to control the verification complexity and at same time ensure the completeness of the algorithm by means of interpolation. In this sense, we intend to verify real-world embedded software in contrast to closely related works that apply the technique to handcrafted small-size benchmarks [12, 18].

3. BMC FOR MULTI-THREADED SOFTWARE IN EMBEDDED SYSTEMS

In BMC, the program to be analyzed is modelled as a state transition system, which is built by extracting its behaviour from the control-flow graph (CFG). This graph is used as part of a translation process from program text to SSA-form [8]. Each thread is modelled as a CFG and nodes represent control points while edges represent transitions (or program statements). Each transition is enabled *iff* the condition that guards it is true and the process to which it belongs is at the corresponding control point. For example, Figure 1 shows the CFG of two threads T_X and T_Y. Each thread has two single transitions. The guards $x > 2$ and $x > 3$ as well as the control points $\{T_{X_0}, T_{X_1}\}$ and $\{T_{Y_0}, T_{Y_1}\}$ determine if a transition is enabled or not.

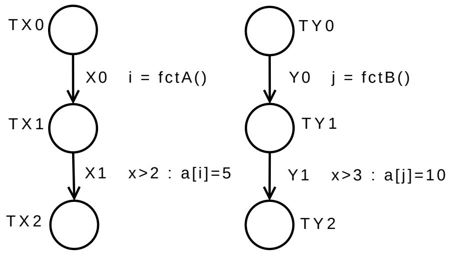

Figure 1: Control-flow graph of two threads.

A transition system $M = (S, R, S_0)$ is an abstract machine that consists of a set of states S (where $S_0 \subseteq S$ represents the set of initial states) and transitions R between states, i.e., for each $\gamma \in R$, $\gamma \subseteq S \times S$. A state $s \in S$ consists of the value of the program counter *pc* and the values of all program variables. An initial state s_0 assigns the initial program location of the CFG to the *pc*. We identify each transition $\gamma = (s_i, s_{i+1})$ between two states s_i and s_{i+1} with a logical formula $\gamma(s_i, s_{i+1})$ that captures the constraints on the values of the program counter and the program variables. Given a transition system M, a property ϕ, and a bound k, BMC unrolls the system k times and translates it into a verification condition ψ such that ψ is satisfiable *iff* ϕ has a counterexample of depth less than or equal to k. The model checking problem associated with SMT-based BMC for checking linear temporal logic (LTL) properties is then formulated by constructing the logical formula [3, 10]:

$$\psi^k = \underbrace{I(s_0) \wedge \bigwedge_{i=0}^{k-1} \gamma(s_i, s_{i+1})}_{constraints} \wedge \overbrace{P(s_k)}^{property} \tag{1}$$

where $P(s_k)$ represents a LTL property ϕ in step k, I is the function for the set of initial states of M and $\gamma(s_i, s_{i+1})$ is the function of the transition relation of M at time steps i and $i+1$. Hence, the formula $\bigwedge_{i=0}^{k-1} \gamma(s_i, s_{i+1})$ represents the set of all executions of M up to the length k or less. $P(s_k)$ is derived from the property being checked and represents the condition that it is violated by a bounded execution of M of length k or less. However, formula (1) encodes all allowed interleavings of the given threads. The core idea of our approach is to consider a series of under-approximations of a given model by encoding additional literals into the verification condition ψ and by extracting the proof objects generated from an SMT solver [9]. We then rewrite formula (1) as:

$$\psi^k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} \gamma(s_i, s_{i+1}) \wedge P(s_k) \bigwedge_{\forall j \in T} \bigwedge_{\forall l \in C} v_{jl} \qquad (2)$$

where $v_{jl} \in P$ are literals that encode the control points of each thread. We denote the set of threads by T, the set of control points by C, and the set of control literals by P. In the example of Figure 1, $T = \{T_X, T_Y\}$, $C = \{T_{X_0}, T_{X_1}, T_{X_2}, T_{Y_0}, T_{Y_1}, T_{Y_2}\}$, and the set of control literals $P = \left\{v_{X_{X_0}}, v_{X_{X_1}}, v_{X_{X_2}}, v_{Y_{Y_0}}, v_{Y_{Y_1}}, v_{Y_{Y_2}}\right\}$. Initially, each literal in P is set to be false, because we aim to control the number of interleavings. As in [12], we also intend, at every state, to expand only the transitions of the enabled thread with the smallest subscript (i.e., assume that $T_X < T_Y$). However, at this point in time, we are not aware of which heuristics we could use in order to update the set of variables in P on each iteration of our algorithm. As a running example, consider all thread interleavings of Figure 1.

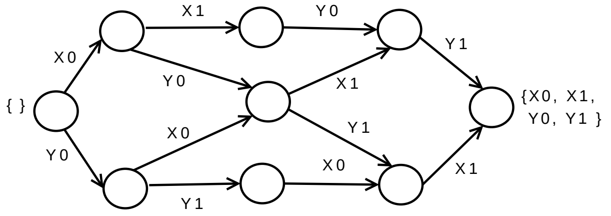

Figure 2: All interleavings of the threads in Figure 1.

At the first step, we consider that only transition X_0 is expanded from the initial state and consequently we execute symbolically program statements X_0, X_1, Y_0, Y_1 and build formula (2). If we do not find any counter-example when checking (2), we then analyze the proof objects generated from the SMT solver to determine if any of the literals in P is part of the proof objects. For example, if $\neg v_{X_{X_0}}$ was used by the proof, we should then remove it from P in order to continue to the next iteration since this literal was used to prove that the property holds and can now assume either true or false (while the others must be false).

Furthermore, it is important to note that it is difficult to determine statically whether transition X_1 is dependent on Y_1. If $i \neq j$ holds in some executions, transitions X_1 and Y_1 become independent, meaning that the two sequences $X_0; X_1; Y_0; Y_1$ and $Y_0; Y_1; X_0; X_1$ are equivalent. Differently from [11, 12, 18], we intend to integrate symbolic partial order reduction methods to take advantage of such information. Furthermore, we intend to analyse the proof objects to determine whether the proof does not depend on the under-approximation itself. It means that the proof will allow us to show that the property holds on the original model, and the SMT-based BMC procedure terminates without needing to increase the bound until the SMT solver explodes.

3.1 Case Studies

In order to validate our SMT-based BMC procedure, three different case studies specific to the domain of telecommunications software are considered. The first case study is the middleware for IPTV applications, which is not only software that allows two or more applications to exchange data, but also provides a set of drivers (e.g., memory allocation driver, ticker driver, and external event driver), TCP/IP socket I/O, and graphics facilities to make the software application development easier. The verification of IPTV middleware is a good example of an embedded software application that requires the verification of not only functional properties, but also temporal properties due to the real-time data transmission and control.

A digital TV platform also provides a set of processors and Application Specific Integrated Circuits (ASICs) that allows faster data decoding and encoding. As this kind of application requires high computational power, more than one core processor is often used. For this purpose, the second case study deals with software verification of multi-core systems. The third case study is related to the front-end of the digital TV system. The front-end consists essentially of a tuner Integrated Circuit (IC) that performs the demodulation and error correction functions for receiving terrestrial/satellite digital broadcasting. Consequently, the embedded software that controls the functions of the tuner also imposes challenges to verify the functional and temporal properties, since it will force the model checker to consider the stringent constraints of the hardware (e.g. real-time, memory allocation, interrupts, and concurrency).

3.2 Main Contributions

The main contribution of this PhD thesis will be a comprehensive SMT-based BMC procedure to verify embedded software in multi-core systems. We will provide details of an accurate translation from ANSI-C programs into quantifier-free formulae and show that our encoding allows us to reason about arithmetic overflow and to verify programs that make use of bit-level, pointers, unions and floating-point arithmetic (i.e., ANSI-C constructs commonly found in embedded software). We also intend to check the effectiveness of our encoding techniques by using different SMT solvers (CVC3, Boolector, and Z3) and exploit different background theories and solvers, based on an analysis of the syntactic structure of a given ANSI-C program.

Additionally, we will explore a new concept called continuous verification [7] to detect design errors and integration problems as quickly as possible by exploiting information from the software configuration management (SCM) system, systematically focusing the verification effort on new or modified functions. As a result, we will integrate the continuous verification approach with the combination of different encodings and solvers in order to allow us to go deeper into the system (compared to software model checkers only) and explore more exhaustively the state space (compared to testing only). Finally, we will develop a comprehensive SMT-based BMC procedure of multi-threaded software based on a set of UW models by removing redundant interleavings and consequently reduce the state space exploration.

4. WORK METHODOLOGY

The needed steps and directions to develop the proposed method can be organized in three different phases. In the domain analysis phase, we aim to investigate the resolution proofs and unsatisfiable cores generated by the SMT solvers in order to derive thread invariants. Furthermore, we intend to make a systematic examination of the interpolation procedures for fragments of first order logic. In the proposed methodology phase, an initial version of the algorithmic method and its scope to verify embedded software in multi-core systems are defined and implemented with support for the POSIX threads library. After that, this implementation is further refined in the validation phase by applying it to the case studies presented in Section 3.1. In order to carry out the activities of this PhD dissertation, an incremental and iterative approach will be used with the purpose of reducing risks and uncertainties. For each increment, the three phases will be addressed with different emphasis. For each thesis increment, technical reports will be written and if significant results have been achieved, then scientific papers will be reported to the academic community through publications in workshops and conferences.

5. RESULTS ACHIEVED SO FAR

As a start, we proposed a combination of techniques to verify statically and dynamically the "pure" and hardware-related embedded software as well as techniques that aim to find property violations at system level [6]. Apart from the state space explosion problem, we found that CBMC and SATABS allow us to verify full ANSI-C, but these model checkers make it difficult to specify more complex temporal properties in embedded software. NuSMV2 provides a variety of languages to specify the system's properties, but there is no straightforward mapping from the ANSI-C constructs to the NuSMV language [6].

As a consequence, we investigated SMT-based verification of ANSI-C programs and we described in [8] a new set of encodings that allowed us to reason accurately about bit operations, unions, floating-point arithmetic, pointers and pointer arithmetic. We integrated the CVC3, Boolector, and Z3 solvers with the CBMC front-end [5] and evaluated them using both standard software model checking benchmarks and typical embedded software applications from telecommunications, control systems, and medical devices. We also improved substantially the performance of SMT-based BMC for embedded software by making use of high-level information to simplify the unrolled formula. The results in [8] show that our approach outperforms the CBMC [5] and SMT-CBMC [3] model checkers if we consider the verification of embedded software.

However, for large embedded software, SMT-Based BMC suffers from the state space explosion problem. In [7] we propose a new approach called continuous verification to detect design errors as quickly as possible by looking at the Software Configuration Management (SCM) system and by combining dynamic and static verification to reduce the state space to be explored. We also give a set of encodings that use different background theories in order to improve scalability and precision in a completely automatic way. As a result, the continuous verification approach and the combination of different encodings and solvers allowed us to go deeper into the system (compared to software model checkers only) and to explore more exhaustively the state space (compared to testing only). Controlled experiments using a case study from the telecommunications domain with more than 30K of lines of C code shows that this hybrid solution improves the error-detection capability and reduces the verification time by up to 50% [7].

6. REFERENCES

[1] R. Alur et al. Partial-order reduction in symbolic state-space exploration. *FMSD*, 18(2):97–116, 2001.

[2] T. Andrews et al. Zing: Exploiting program structure for model checking concurrent software. *CONCUR*, pp. 1–15, 2004.

[3] A. Armando, J. Mantovani, and L. Platania Bounded model checking of software using SMT solvers instead of SAT solvers. *Int. J. Softw. Tools Technol. Transf.*, pp. 69–83, 2009.

[4] A. Biere. Bounded model checking. *Handbook of Satisfiability*, pp. 457–481. 2009.

[5] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. *TACAS*, *LNCS* 2988, pp. 168–176, 2004.

[6] L. Cordeiro et al. Semiformal verification of embedded software in medical devices considering stringent hardware constraints. *ICESS*, pp. 396–403, 2009.

[7] L. Cordeiro, B. Fischer, and J. Marques-Silva. Continuous verification of large embedded software using SMT-based bounded model checking. *ECBS*, 2009.

[8] L. Cordeiro, B. Fischer, and J. Marques-Silva. SMT-based bounded model checking for embedded ANSI-C software. *ASE*, pp. 137–148, 2009.

[9] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. *TACAS*, *LNCS* 4963, pp. 337–340, 2008.

[10] M. K. Ganai and A. Gupta. Accelerating high-level bounded model checking. *ICCAD*, pp. 794–801, 2006.

[11] M. K. Ganai and A. Gupta. Efficient modeling of concurrent systems in BMC. *SPIN*, *LNCS* 5156, pp. 114–133, 2008.

[12] O. Grumberg et al. Proof-guided underapproximation-widening for multi-process systems. *POPL*, pp. 122–131, 2005.

[13] V. Kahlon, S. Sankaranarayanan, and A. Gupta. Semantic reduction of thread interleavings in concurrent programs. *TACAS*, *LNCS* 5505, pp. 124–138, 2009.

[14] V. Kahlon, C. Wang, and A. Gupta. Monotonic partial order reduction: An optimal symbolic partial order reduction technique. *CAV*, *LNCS* 5643, pp. 398–413, 2009.

[15] K. L. McMillan. Interpolation and SAT-based model checking. *CAV*, *LNCS* 2725, pp. 1–13, 2003.

[16] K. L. McMillan and N. Amla. Automatic abstraction without counterexamples. *TACAS*, *LNCS* 2619, pp. 2–17, 2003.

[17] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. *TACAS*, *LNCS* 3440, pp. 93–107, 2005.

[18] I. Rabinovitz and O. Grumberg. Bounded model checking of concurrent programs. *CAV*, *LNCS* 3576, pp. 82–97, 2005.