Model Checking Software Requirement Specifications using Domain Reduction Abstraction [*]

Yunja Choi and Mats Heimdahl
Department of Computer Science and Engineering, University of Minnesota
200 Union Street S.E., 4-192, Minneapolis, MN 55455, USA
{yuchoi,heimdahl}@cs.umn.edu

Abstract

As an automated verification and validation tool, model checking can be quite effective in practice, especially when it is used in the context of specification-centered or component-based software development frameworks. Nevertheless, model checking has been quite inefficient when dealing with systems with data variables over a large (or infinite) domain, which is a serious limiting factor for its applicability in practice.

To address this issue, a static abstraction technique, domain reduction abstraction, based on data equivalence and trajectory reduction has been investigated and its prototype automation has been implemented as an extension of the symbolic model checker NuSMV. Unlike other on-the-fly dynamic abstraction techniques, domain reduction abstraction statically analyzes specifications and automatically produces an abstract model which can be reused over time — a feature suitable for regression verification.

This paper describes domain reduction abstraction and its prototype implementation. We also provide some performance data on an industrial avionics specification and discuss various application areas.

1 Introduction

Important classes of software systems can be viewed as consisting of a finite *control component* and a (typically infinite or very large) *data component*. Examples are prevalent in safety critical embedded systems such as aircraft control, train control, and medical device systems. In such systems, the transitions between control variables are guarded by various linear and non-linear data conditions, and the transitions between data values may be subject to various constraints which can be non-deterministic. For example, a temperature control system can have the guarding condition $temp < 10$ for a control transition and $temp' = temp + [-\alpha, \beta]$ as a constraint for the temperature change.

In previous papers [9, 10] we have investigated abstractions over the *input domain* of the systems —a technique we call *domain reduction abstraction* [9, 10]. Domain reduction abstraction statically analyzes a model, extracts numeric conditions and constraints, reduces the data domain by selecting representative data values that subsume possible system behaviors, and leaves the control part of the system unchanged. In domain reduction abstraction, the effort of computing a sound abstract system is expended before verification and the abstraction cost is unrelated to the number of properties to be verified. We have observed that the verification of a substantial software specification will involve hundreds of properties that will have to be re-verified every time the software specification changes—something that happens quite frequently. In the face of scores of properties and frequent *regression verification*, we anticipate that domain reduction abstraction will compare well with other proposed techniques that require the abstraction process to be performed for each property [3, 6, 11, 16, 19, 20].

In this report, we describe a prototype integration of domain reduction abstraction into the symbolic model checker NuSMV [21] in connection with a linear/integer programming tool lp_solve [1]. Using our prototype implementation, we demonstrate the usability and efficiency of domain reduction abstraction in the context of model checking software requirement specifications. The case examples demonstrate the following; (1) the abstraction is fully automated, (2) the abstract domain can be reused unless numeric conditions are changed, and thus, it is suitable for *regression verification*, and (3) counter examples are straight-forward to interpret, and thus, easy to be used for quantitative analysis such as deviation analysis [17] or for specification-based test case generation [14]. Our implementation was developed as a technology demonstration and there are many rather obvious performance improvements we will investi-

[*]This work has been partially supported by NASA grant NAG-1-224 and NASA contract NCC-01-001.

gate in the near future.

The remaining of this paper is organized as follows; Section 2 briefly recalls domain reduction abstraction at an intuitive level. Section 3 discusses our automation framework followed by some application results over industrial specifications. We conclude with a discussion on the promise of our approach.

2 Domain Reduction Abstraction

For seamless integration of automated verification into an engineering process, it is highly desirable that any abstraction process is fully automated and largely hidden from the engineers. To this end, we have investigated an alternative static abstraction approach named domain reduction abstraction.

Domain reduction abstraction is motivated from the observation that only a subset of data values from a data domain has distinct effects on system behaviors. For example, a system with a data condition $\{x < y\}$ over the domain $x = 0..100$, $y = 0..100$ would behave the same as one with a reduced domain $x = \{1, 2\}$, $y = \{1, 2\}$; the reduced domain contains combinations of x, y values that make the data condition true or false. The approach is based on selecting representative values that will exercise all possible truth values of the data conditions in the system. To give the reader a general understanding of our approach, we provide an informal outline of the abstraction technique in this section. Formal proofs of the soundness of the approach and an automation algorithm can be found in [9, 10].

We tackle the problem of numeric variables with two complementary abstractions; first, when the values of the data variables only depend on inputs from the system environment (we call it *constraint-free data transition systems*), a simple data abstraction technique based on a data equivalence relation is used. When the change of the data values is constrained by some data transition rules (we call it *constrained data transition systems*), one data trajectory (a series of data values satisfying all data-constraints) will be computed and used as a representative for all data trajectories with the same characteristic. Here, the characteristic of a data trajectory is determined by the ordered set of data-equivalence classes the data trajectory passes through.

Figure 1 shows a view of domain reduction abstraction. We partition the domain of numeric variables into equivalence classes by the valuation of the numeric conditions appearing in the transition conditions and verification properties; an equivalence relation over the data domain is defined for the purpose as

$$x \equiv y \text{ iff } c(x) = c(y), \forall c \in C.$$

where C is the set of numeric data conditions in the system. It is shown that an abstract system over a reduced domain

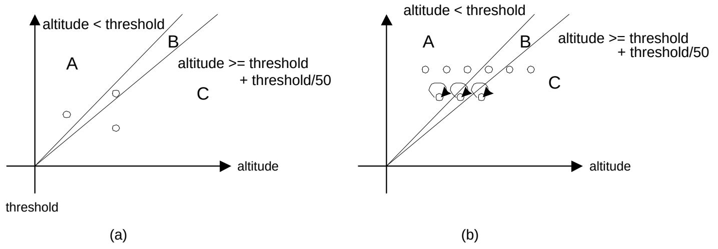

Figure 1. Abstraction approach

consisting of representative values from each equivalence class is an exact abstraction of the original system [10].

For example, the partition A represents the region that satisfies $altitude < threshold \wedge altitude < threshold + \frac{threshold}{50}$. We replace the set of possible values in a data-equivalence class with a randomly selected representative from the class (see (a) in Figure 1). This is a variation of the data abstraction technique suggested by Clarke et al. [12]; instead of mapping each partition class to a symbolic enumerated value, we simply select a representative from each class effectively removing the mapping process. The idea behind this technique is closely related to that of partition testing [4].

When data-constraints must be taken into account, such as the constraint $altitude' = altitude + 10$, random representatives cannot be selected since they are likely to violate the data constraints. Nevertheless, we can refine the abstraction by computing a minimal data trajectory that we use as a representative for all data trajectories passing through the same set of data-equivalence classes in the same order. For example, all data-trajectories passing through the equivalence classes A, B, C (in that order) can be simulated by one minimal trajectory (see (b) in Figure 1). Simulation of the original system by the abstract system is ensured by introducing data stuttering so that the minimal trajectory can always be as long as any other trajectory. This approach provides us a conservative abstraction of the original system such that all the behaviors (transitions) of the original system are included in the abstract system [10, 9].

3 Automation

Figure 2 (a) shows an overview of our existing verification framework for specifications written in RSML^{-e}. In this framework, we validate the behavioral aspect of RSML^{-e} specifications using the NIMBUS execution environment [18] and then translate the system model into the input language of the symbolic model checker NuSMV [21] using a built-in translator for verification purposes.

RSML^{-e} [22] is a hierarchical, synchronous data-flow specification language that has similar language constructs to those of NuSMV, and thus, the translation is rather straightforward and has been successfully used for verifying interesting properties for industrial specifications [8]. Nevertheless, the straightforward translation of data variables over large data domains often caused the state-space explosion problem when model checking, giving us a compelling reason to incorporate an automated abstraction technique into the framework. To address this problem, we have implemented domain reduction abstraction as an extension to NuSMV.

Figure 2 (b) illustrates how the abstraction extension interacts with our existing verification framework; it extracts all information about numeric data variables, such as domain information, data conditions and constraints, from the flattened system model in NuSMV after taking property specifications into account. We have modified the NuSMV source code so that NuSMV initiates the abstraction process and generates an abstract model, combining the parse tree information saved in the beginning of the process and the reduced domain after the abstraction process.

The prototype abstraction tool has a couple of intermediate processes; the rewriting process simplifies numeric conditions into a form that can be comprehended by *lp_solve* [1] which is a linear/integer programming tool used to select representative data values. Since *lp_solve* can handle only linear data conditions, current automation of domain reduction abstraction is also limited to systems with linear data conditions. The classification process classifies the elements in the set of system data conditions so that different classes do not share the same variables. By using classification, we can reduce the cost for feasibility checking for possible data equivalence classes from 2^n to $2^{m_1} + 2^{m_2} + \cdots + 2^{m_k}$, where n is the number of data conditions in the system and each m_i is the number of data conditions in each classified set of data conditions with $\Sigma_{i=1}^{k} m_i = n$.

In this abstraction approach, changes on specifications do not necessarily require rerunning the whole abstraction process as long as there is no change on data part; we can simply regenerate the abstract model by combining the changed parse tree and the same reduced domain produced earlier.

4 Applications

In this section, we present some application results over industrial avionics specifications. All tests were performed on a 800 MHz Linux machine with 512 M memory.

We have applied the prototype abstraction tool to two components of the Flight Control System (FCS) development at Rockwell-Collins: a version of Flight Guidance Systems (FGS) and a version of Flight Management Systems (FMS). Since the FMS specification is still under development and consider proprietary, we can only provide a detailed discussion of the FGS.

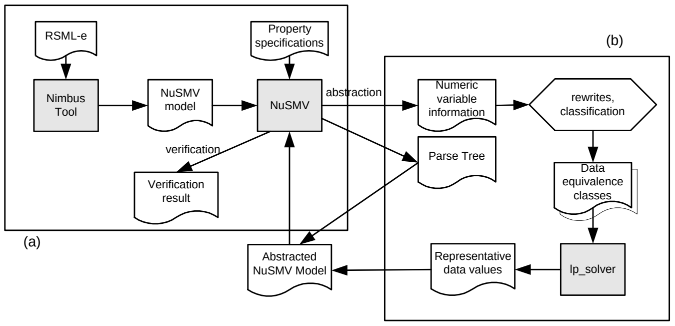

Figure 2. Verification Framework

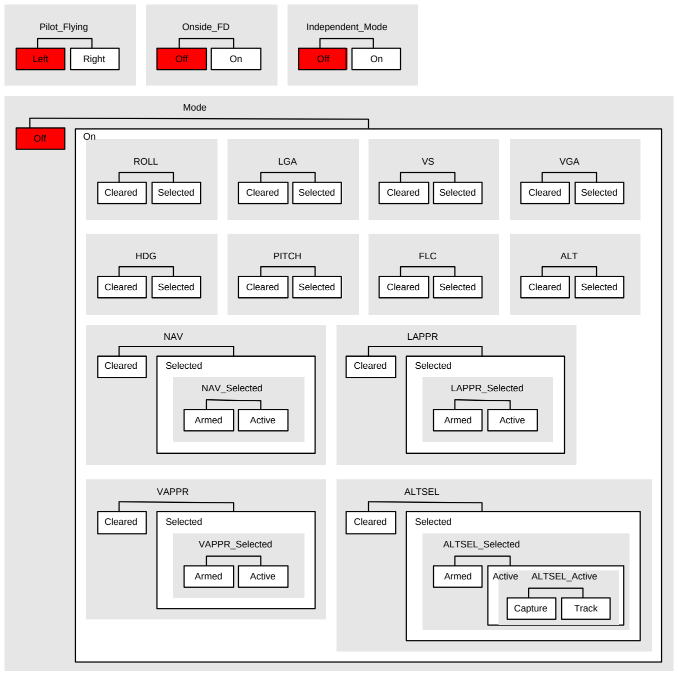

Figure 3. A Flight Guidance System

The FGS (Figure 3) compares the measured state of an aircraft (position, speed, and altitude) to the desired state and generates pitch and roll guidance commands to minimize the difference between the measured and desired state. The FGS can be broken down to mode logic, which determines which lateral and vertical modes of operation are active and armed at any given time, and the flight control laws that accept information about the aircraft's current and desired state and compute the pitch and roll guidance commands.

While a FGS defines the control logic of an aircraft, it interacts with a Flight Management System (FMS) whose mode logic is largely dependent on complex data conditions. Though the version of the FMS used in this application in its current form is neither complete nor correct, we believe the high complexity of the numeric data conditions of the system is a good test to assess the applicability of our approach in practice.

FGS and FMS require various numeric data variables to interact with its environment; the range of these numeric data variables vary from hundreds to ten thousands, and thus, it is infeasible to model check these systems without applying abstractions. Since the abstraction technique was not available when researchers at Rockwell-Collins first started using our verification framework, all numeric data variables had to be manually abstracted out which requires significant modifications of the system specification. Domain reduction abstraction applies to the original specification (after translation); it changes the domains of data variables and makes some other additions to the data constraints (to introduce data stuttering) leaving most of the mode logic unchanged. Therefore, there is no need of change in the specification level.

Figure 4 shows the system usage data before and after applying data abstraction on these two systems; part (a) "RSML^{-e} size" represents the number of lines of code of the RSML^{-e} specifications, "NuSMV size" is the number of lines of code after the specifications are automatically translated into the NuSMV input language. There are four major factors regarding numeric data variables that have effect on the model checking complexity: the number of numeric variables, the number of numeric data conditions, the number of numeric data constraints, and the size of domain of each data variable. For example, the relative comparison between "number of BDD variables", which is the number of variables to encode system variables in the OBDD, and "number of BDD & ADD nodes", which is the number of nodes actually required to store all variable information in NuSMV, gives us a rough idea how large the original domains are; roughly speaking, $1,013,026/1,395 \approx 726$ ($1,831,391/2,067 \approx 886$) nodes are required before abstraction, $183,331/851 \approx 215$ ($176,610/835 \approx 211.5$) after abstraction to encode a BDD variable in FGS (FMS). This shows how expensive it is to represent numeric data values over large domain in a symbolic way. The row "memory in use" shows the memory usage up to the variable encoding stage. After abstraction, the memory usage dropped to $8M$ ($8.8M$) from $25.5M$ ($43.9M$) for the FGS (FMS).

The time for abstraction was 11 seconds for the FGS and about 1 hour 15 minutes for the FMS. This large difference is because the abstraction performance depends on the number of *inter-related* numeric data variables and data conditions, not on the overall number of variables and conditions; the most complex subgroup of the FGS after classification has only 4 inter-related conditions over 3 numeric variables whereas that of FMS has 17 inter-related data conditions over 12 numeric variables.

	FGS	FMS		FGS	FMS		FGS	FMS
RSML-e size in LOC	3,832	3,366	time for abstraction	11 s	< 1.3 h	verification time	87.25 s	
NuSMV size in LOC	2,953	1,839	number of calls to lp_solve	55	131,174	total memory in use	16 M	
number of numeric variables	25	92	number of data variables in a maximum sub-group	3	12	time for a counter example generation	7425.66 s	8.34 s
number of numeric data conditions	30	26	number of data conditions in a maximum sub-group	4	17	total memory in use	75.4 M	11 M
number of numeric data constraints	9	4				time for a counter example generation using BMC	96 s	
number of BDD variables	1,395	2,067	number of BDD variables	851	835	total memory in use for BMC	25 M	
number of BDD & ADD nodes	1,013,026	1,831,391	number of BDD & ADD nodes	183,331	176,610			
memory in use	25.5 M	43.9M	memory in use	8 M	8.8 M			

(a) before abstraction (b) after abstraction (c) verification performance

Figure 4. NuSMV usage data

Following two properties are part of the requirement properties for the FGS system identified by domain engineers.

R1 *Only one lateral mode shall ever be active at any time.*

R2 *The $APPR$ Switch Lamp shall be lit if and only if $LAPPR$ mode is selected.*

NuSMV verified $R1$ in 88 seconds over the abstract FGS model whereas it ran out of memory while verifying it over the original model. It refuted the property $R2$ in 7425.66 seconds and 75.4 M system memory generating a 4-step counter example. Over the abstract FMS model, NuSMV generates a counter example after 8.34 seconds consuming 11 M memory for a property;

R3 *The criteria for aircraft climb phase and the criteria for aircraft descent phase are never met at the same time.*

It appears that the technique can also extend the capability of the bounded model checking (BMC) which is much more efficient in counter example generation than symbolic model checking; the built-in bounded model checker in NuSMV failed to build a boolean formula for the original system but successfully generated the same 4-step counter example for the same property $R2$ over the abstract FGS model in 95 seconds consuming 25 M system memory, reducing the time and memory usage greatly (Figure 4 (c)). It will be worth while to investigate further the impact of domain reduction abstraction on the bounded model checking.

In order to validate the correctness of the abstraction tool, same 194 properties identified by researchers at Rockwell-Collins are tested over a manually abstracted version of FGS, which has been evolved and verified for over a year using NuSMV, and an automatically abstracted model of FGS; the manually abstracted model takes 4390.5 seconds and 117 M of system memory verifying 189 properties and generating counter examples for 5 properties. An abstract model generated using domain reduction abstraction takes 4172.96 seconds and 115 M of memory for the verification and refutation plus 11 seconds and less than 1 M of memory for the abstraction, producing the same result as that of manually abstracted model. Note that the performance data presented here is a result of using the $-coi$ option that enables *cone of influence reduction* and the $-dynamic$ option that enables *dynamic reordering of BDD variables* in NuSMV. These options play a crucial role to make model checking feasible for the FGS and the FMS that have over 800 BDD variables.

These application results show a promise of the approach in practice; even though the tool is a prototype implementation with a focus on correctness, not on performance, its performance is tolerable for systems with closely interrelated numeric data conditions. Considering that checking hundreds of properties over a reasonable sized system normally takes several hours (for example, checking a total 298 of properties over a version of the FGS model takes over 10 hours refuting 24 properties among them), spending an hour as a preprocess to reduce the verification time can be tolerable. Also, since the current prototype tool has much rooms for performance optimization such as improvement of the current brute-force feasibility checking mechanism, this performance data can be improved greatly; after inserting monitoring code in the current implementation, we observed that 125,520 out of 131,174 calls to lp_solve for the FMS were with unsatisfiable sets of data conditions. This number of calls can be reduced significantly if we identify some infeasible sets of data conditions up-front. For example, if we identify 2 out of 17 data conditions in a set are inconsistent, we can avoid 2^{15} calls to lp_solve.

5 Discussion

In this paper we presented an extension to the NuSMV model checker—an extension that implements domain reduction abstraction. We also illustrated the applicability of domain reduction abstraction on some industrial size problems. The results show dramatic reduction of the size of the models and this reduction made the use of model checking feasible. The abstraction process is automated and requires no user intervention. Counter examples generated from the abstracted model are straight-forward to understand, and thus, there is no need for interpretation. It generates an abstract model that can be re-used over time unless some numeric conditions are changed. Even when some numeric conditions are changed, we need to apply the abstraction only to those subgroups affected by the change — a useful feature when a large number of properties need to be verified over time. In addition, since domain abstraction is orthogonal to other abstraction techniques, we can apply other existing techniques [2, 5, 6, 7, 12, 13, 15] in concert with domain reduction abstraction if necessary.

Though we believe our technique is promising, there are many remaining areas that need improvement. First, we need to investigate a better way of eliminating infeasible data equivalence classes as we pointed out in the previous section. Second, although the application result presented in the previous section shows the promise of our approach, it is too early to claim that the approach is practical in general. We would like to collect more data on industry problems to assess the cost-effectiveness. Lastly, the automation is implemented for a limited type of data conditions and constraints. We plan to expand the capability to non-linear and/or limited non-deterministic data transition constraints.

References

[1] Mixed integer/linear programming tool lp_solve version 3.1. ftp://ftp.ics.ele.tue.nl/pub/lp_solve/.

[2] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, March 1996.

[3] T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of c programs. *SIGPLAN Notices (ACM Special Interest Group on Programming Languages)*, 36(5):203–213, May 2001.

[4] B. Beizer. *Software testing techniques*. Van Nostrand Reinhold, New York, 2nd edition, 1990.

[5] R. Bharadwaj and C. Heitmeyer. Model checking complete requirements specifications using abstraction. In *First ACM SIGPLAN Workshop on Automatic Analysis of Software*, 1997.

[6] T. Bultan, R. A. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In *Computer Aided Verification*. Springer Varlag, 1997.

[7] W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *Proc. of CAV'97, LNCS 1254*, pages 316–327. Springer, June 1997.

[8] Y. Choi and M. Heimdahl. Model checking RSML^{-e}requirements. In *Proceedings of the 7th IEEE/IEICE International Symposium on High Assurance Systems Engineering*, October 2002.

[9] Y. Choi, M. P. Heimdahl, and S. Rayadurgam. Domain reduction abstraction. Technical Report 02-013. University of Minnesota, April 2002.

[10] Y. Choi, S. Rayadurgam, and M. Heimdahl. Automatic abstraction for model checking software systems with interrelated numeric constraints. In *Proceedings of the 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE-9)*, pages 164–174, September 2001.

[11] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proceedings of the 12th International Conference on Computer Aided Verification*, pages 154–169, July 2000.

[12] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transaction on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

[13] E. Emerson and K. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Thirteenth Annual IEEE Symposium on Logics in Computer Science*, pages 70–80, 1998.

[14] A. Gargantini and C. Heitmeyer. Using model checking to generate tests from requirements specifications. *Software Engineering Notes*, 24(6):146–162, November 1999.

[15] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proceedings of the Computer Aided Verification(CAV 1997)*, 1997.

[16] N. Halbwachs, Y. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

[17] M. P. Heimdahl, Y. Choi, and M. Whalen. Deviation analysis via model checking. In *International Conference on Automatied Software Engineering*, September 2002.

[18] M. P. Heimdahl, J. M. Thompson, and M. W. Whalen. Executing state-based specifications in a heterogeneous environment. Technical Report TR 98-029, University of Minnesota, Department of Computer Science, Minneapolis, MN, 1998.

[19] T. A. Henzinger, R. Jhala, R. Majumdar, and R. Majumdar. Lazy abstraction. In *Proceedings of the 29th Symposium on Principles of Programming Languages*, January 2002.

[20] K. S. Namjoshi and R. P. Kurshan. Syntatic program transformations for automatic abstraction. In *12th International Conference, CAV2000*, pages 435–449, July 2000.

[21] NuSMV: A New Symbolic Model Checking. Available at http://http://nusmv.irst.itc.it/.

[22] M. W. Whalen. A formal semantics for RSML^{-e}. Master's thesis, University of Minnesota, May 2000.