Translation Validation of Transformations of Embedded System Specifications using Equivalence Checking

Kunal Banerjee*
Dept of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India
kunalb@cse.iitkgp.ernet.in

Motivation

In the last two decades extensive research has been conducted addressing the design methodology of embedded systems and their verification. Application areas of such systems include but are not limited to cars, telecommunication equipment, medical systems, consumer electronics, robotics, the authentic systems, etc. The initial behavioural specification of an embedded system goes through significant optimizing transformations, automated and also human-guided, before being mapped to an architecture. Establishing the validity of these transformations is crucial to ensure that the intended behaviour of a system has not been faultily altered during synthesis. State-of-the-art verification methods fail to cope with the complexity of the problem. So, we have devised some efficient translation validation methodologies to handle various code transformations; our specific contributions are briefly described below.

Keywords

Translation Validation, Finite State Machine with Datapath (FSMD), Array Data-Dependence Graph (ADDG), Code Transformations, Recurrences, Bisimulation

1. VERIFICATION OF CODE MOTION TECHNIQUES USING SYMBOLIC VALUE PROPAGATION

Scheduling optimization is a crucial step in design of embedded systems. Code motions constitute important components of such optimizations in order to save hardware resources or meet hard or soft deadlines. Since code motions change the data flow of a program considerably, verifying correctness of the optimization process poses serious challenge. The situation is further aggravated when schedulers [6, 15] modify the control flow of the program as well. Our initial objective was to develop a unified verification approach for code motion techniques, including code motions across loop, and control structure modifications. This combination of features had not been previously achieved by any single verification technique. A preliminary version of our work appears in [2] which has been modified considerably to handle speculative code motions and dynamic loop scheduling [15] – the updated technique can be found in [3]. In addition to uniform and non-uniform code motion techniques, this work aims at verifying code motions across loops by propagating the variable values through all the subsequent path segments if mismatch in the values of some live variables is detected. Repeated propagation of values is possible until an equivalent path or a final path segment ending in the reset state is reached. In the latter case, any prevailing discrepancy in values indicates that the original and the transformed behaviours are not equivalent; otherwise they are. The variables whose values are propagated beyond a loop must be invariant to that loop for valid code motions across loops. The loop invariance of such values can be ascertained by comparing the propagated values that are obtained while entering the loop and after one traversal of the loop. The method has been implemented and satisfactorily tested on the outputs of a basic block based scheduler [14], a path based scheduler [6] and the high-level synthesis tool SPARK[7] for some benchmark examples.

2. EXTENDING THE FSMD FRAMEWORK FOR VALIDATING CODE MOTIONS OF ARRAY-HANDLING PROGRAMS

The FSMD models provide a formalism to represent any simple sequential behaviour. Literature has many examples where this model has been successfully applied for behavioural verification of programs. All these methods, however, cannot handle an important class of programs, namely those involving arrays. This limitation is overcome with Finite State Machine with Datapath *having Arrays* (FSMDA) models which are extension of FSMD models; the corresponding symbolic value propagation based equivalence checking algorithm has also been enhanced so that code motions of array-intensive behaviours can be validated; the details of this new mechanism is reported in [5]. Note that our tool detected a bug in the implementation of copy propagation for array variables in the SPARK [7] tool. Our tool can be downloaded from `http://cse.iitkgp.ac.in/~kunban/`

*The work presented here was supported by TCS Research Fellowship. The author acknowledges C Mandal and D Sarkar, Dept of CSE, IIT Kharagpur, for their technical participation in this work.

IRISS 2015, Goa, India

`EquivalenceChecker.tar.gz` along with the benchmarks, installation and usage guidelines.

3. DERIVING BISIMULATION RELATIONS FROM PATH BASED EQUIVALENCE CHECKERS

Constructing bisimulation relations between programs as a means of translation validation has been an active field of study. The problem is in general undecidable. Currently available bisimulation based mechanisms suffer from drawbacks such as non-termination and significant restriction on the structures of programs to be checked. Our research group have developed path based equivalence checkers – specifically, a path extension based equivalence checker [12, 13, 11] and a (more general) value propagation based equivalence checker [2, 3] – as an alternative translation validation technique to alleviate these drawbacks. In this work, both path extension and value propagation based equivalence checking of programs (flowcharts) are leveraged to establish a bisimulation relation between a program and its translated version by constructing the relation from the outputs of the equivalence checker. Moreover, none of the earlier methods that establish equivalence through construction of bisimulation relations has been shown to tackle code motion across loops; the present work demonstrates, for the first time, the existence of a bisimulation relation under such a situation. Note that the work on deriving bisimulation relations from path extension based equivalence checkers has been accepted in [4]; we plan to communicate the entire work to a journal in near future.

4. AN EQUIVALENCE CHECKING MECHANISM FOR HANDLING RECURRENCES IN ARRAY-INTENSIVE PROGRAMS

Compiler optimization of array-intensive programs involves extensive application of loop transformations and arithmetic transformations. Hence, translation validation of array-intensive programs requires manipulation of sets and relations of integer points (representing array indices) bounded by constraints to account for loop transformations and simplification of arithmetic expressions to handle arithmetic transformations as shown in our earlier works [8, 9, 10]. A major obstacle for verification of such programs is posed by the presence of recurrences, where an element of an array gets defined in terms of some other previously defined element(s) of the same array. Recurrences lead to cycles in the data-dependence graph of the program which make dependence analyses and simplifications (through closed-form representations) of the data transformations difficult. In our work reported in [1], array data-dependence graphs (ADDGs) are used to represent both the original and the optimized version of the program and a validation scheme is proposed where the cycles in the ADDGs are isolated from the acyclic portions and treated separately. Thus, this work provides a unified equivalence checking framework to handle loop and arithmetic transformations along with recurrences. An extended version of the work is currently under communication to a conference; we also intend to submit this work to a journal after formally treating the algorithm's correctness and complexity issues.

5. REFERENCES

[1] K. Banerjee. An equivalence checking mechanism for handling recurrences in array-intensive programs. In *Principles of Programming Languages (POPL): Student Research Competition*, page (accepted), January 2015.

[2] K. Banerjee, C. Karfa, D. Sarkar, and C. Mandal. A value propagation based equivalence checking method for verification of code motion techniques. In *ISED*, pages 67–71, 2012.

[3] K. Banerjee, C. Karfa, D. Sarkar, and C. Mandal. Verification of code motion techniques using value propagation. *IEEE Trans. on CAD of ICS*, 33(8):1180–1193, 2014.

[4] K. Banerjee, C. Mandal, and D. Sarkar. Deriving bisimulation relations from path extension based equivalence checkers. In *IMPECS-POPL Workshop on Emerging Research and Development Trends in Programming Languages (WEPL)*, page (accepted), January 2015.

[5] K. Banerjee, D. Sarkar, and C. Mandal. Extending the FSMD framework for validating code motions of array-handling programs. *IEEE Trans. on CAD of ICS*, 33(12):2015–2019, 2014.

[6] R. Camposano. Path-based scheduling for synthesis. *IEEE Trans on CAD of ICS*, 10(1):85–93, Jan. 1991.

[7] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. SPARK: A high-level synthesis framework for applying parallelizing compiler transformations. In *VLSI Design*, pages 461–466, 2003.

[8] C. Karfa, K. Banerjee, D. Sarkar, and C. Mandal. Equivalence checking of array-intensive programs. In *ISVLSI*, pages 156–161, 2011.

[9] C. Karfa, K. Banerjee, D. Sarkar, and C. Mandal. Experimentation with SMT solvers and theorem provers for verification of loop and arithmetic transformations. In *I-CARE*, pages 3:1–3:4, 2013.

[10] C. Karfa, K. Banerjee, D. Sarkar, and C. Mandal. Verification of loop and arithmetic transformations of array-intensive behaviours. *IEEE Trans. on CAD of ICS*, 32(11):1787–1800, 2013.

[11] C. Karfa, C. Mandal, and D. Sarkar. Formal verification of code motion techniques using data-flow-driven equivalence checking. *ACM Trans. Design Autom. Electr. Syst.*, 17(3):30, 2012.

[12] C. Karfa, C. Mandal, D. Sarkar, S. R. Pentakota, and C. Reade. A formal verification method of scheduling in high-level synthesis. In *ISQED*, pages 71–78, 2006.

[13] C. Karfa, D. Sarkar, C. Mandal, and P. Kumar. An equivalence-checking method for scheduling verification in high-level synthesis. *IEEE Trans on CAD of ICS*, 27:556–569, 2008.

[14] C. A. Mandal and R. M. Zimmer. A genetic algorithm for the synthesis of structured data paths. In *VLSI Design*, pages 206–211, 2000.

[15] M. Rahmouni and A. A. Jerraya. Formulation and evaluation of scheduling techniques for control flow graphs. In *EURO-DAC*, pages 386–391, 1995.