In *Proceedings of the 8th International Workshop on Post-Binary ULSI Systems*,
Freiburg im Breisgau, Germany, invited paper, pages 23–24, May 19, 1999.

Interval Diagram Techniques and Their Applications

Karsten Strehl and Lothar Thiele

Computer Engineering and Networks Lab (TIK)
Swiss Federal Institute of Technology (ETH)
Gloriastrasse 35, 8092 Zurich, Switzerland
eMail: {strehl,thiele}@tik.ee.ethz.ch
WWW: http://www.tik.ee.ethz.ch/˜symbolic

Abstract

Interval diagram techniques have been applied successfully to symbolic formal verification of process networks, Petri nets, and timed automata, outperforming conventional approaches and avoiding some of their drawbacks. Our approach is based on a novel, efficient form of representation for multi-valued functions called *interval decision diagram* (IDD) and the corresponding image computation technique using *interval mapping diagrams* (IMDs). Besides formal verification, interval diagram techniques may be used to perform symbolic scheduling of heterogeneous embedded systems. IDDs and IMDs are presented, their properties are described, and the above applications are sketched.

1 Introduction

Many formal verification methods such as symbolic model checking try to reduce the state explosion problem by implicit construction of the state space. The major limiting factor is the size of the symbolic representation mostly stored in huge binary decision diagrams (BDDs). When applying conventional symbolic formal verification techniques to certain model of computation classes, several difficulties occur that question their usefulness in this area. The mentioned models of computation include many high-level and data-flow oriented models, models consisting of finite-state components communicating via FIFO channels, and models containing internal clocks for representing timing behavior. Some examples are given below.

The traditional BDD-based methods of automated verification suffer from the drawback that a binary representation of the system model and its state is required. Even the use of multi-valued decision diagrams (MDDs) instead of BDDs cannot resolve the problem. *Interval diagram techniques*—using *interval decision diagrams* (IDDs) and *interval mapping diagrams* (IMDs)—have shown to be convenient for formal verification of, e.g., process networks [2], Petri nets [3], and timed automata [1]. They remedy some deficiencies of traditional approaches and often provide advantages regarding computation time and memory resources. The major enhancements of formal verification with IDDs and IMDs are:

- No state variable bounds due to binary coding or complementation are necessary as with conventional symbolic approaches.
- The transition relation representation is quite compact as only "state distances" are stored instead of combinations of state and successor. Accordingly, an innovative technique for image computation is used.
- Due to the enhanced merging capabilities of IDDs and the abandonment of binary coding, state set descriptions are more compact than using BDDs.

Due to similar transition behaviors, the above advantages of symbolic approaches based on interval diagram techniques may be transferred to the area of symbolic scheduling of, e.g., a model of computation called *FunState* [4]. FunState unifies many different well-known models of computation, supports stepwise refinement and hierarchy, and is suited to represent many different synchronization, communication, and scheduling policies. We briefly present the used interval diagrams and the above applications for formal verification and symbolic scheduling.

2 Interval Diagram Techniques

For formal verification of, e.g., Petri nets [3], process networks [2], and timed automata [1], interval diagram techniques—using interval decision diagrams (IDDs) and interval mapping diagrams (IMDs)—have shown to be a favorable alternative to BDD techniques. This results from the fact that for this kind of models of computation, the transition relation has a very regular structure that IMDs can conveniently represent. While BDDs have to represent explicitly all possible state variable value pairs before and after a certain transition, IMDs store only the *state distance*—the difference between the state variable values before and after the transition. Especially for models with large state variable values, this approach is reasonable and useful. IDDs are used to represent state sets during computations.

2.1 Interval Decision Diagrams

IDDs are a generalization of BDDs and MDDs allowing diagram variables to be integers and child nodes to be associated with intervals rather than single values. In Figure 1 a), an example IDD is shown. It represents the Boolean function $s(u,v,w) = (u \leq 3) \wedge (v \geq 6) \vee (u \geq 4) \wedge (w \leq 7)$ with $u,v,w \in [0,\infty)$.

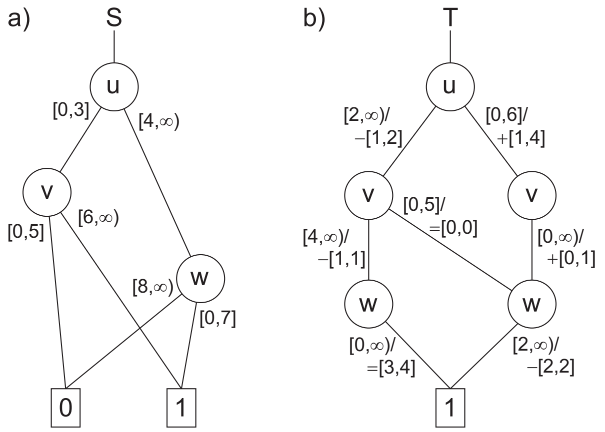

Figure 1: Interval decision diagram and interval mapping diagram.

Equivalent to BDDs, IDDs have a reduced and ordered form, providing a canonical representation of a class of Boolean functions—which is important with respect to efficient fixpoint computations often necessary for formal verification. Methods such as the *If-Then-Else* operator ITE are defined similarly to their BDD equivalents and may be computed as usual for decision diagram applications using a computed table to improve performance.

2.2 Interval Mapping Diagrams

IMDs are represented by graphs similar to IDDs. Their edges are labeled with functions mapping intervals onto intervals. The graph contains only one terminal node. Figure 1 b) shows an example IMD.

With regard to transition relations, IMDs work as follows. Each edge is labeled with a condition—the *predicate interval*—on its source node variable and the kind and amount of change—the *action operator* and the *action interval*—the variable is to undergo. Each path represents a possible state transition which is executable if all edges along the path are enabled. The combination of predicate and action interval parameterizes the mapping function and completely defines its behavior.

2.3 Image Computation

In [2], an efficient algorithm is described to perform forward or backward image computation using an IDD S for the state set and an IMD T for the transition relation, resulting in an IDD S' representing the image state set. This algorithm may be used to perform reachability analysis or symbolic model checking by fixpoint computation. IMDs are dedicated to image computation especially for process networks and related models of computation as the state distance (action interval) combined with the respective firing condition (predicate interval) may be stored more efficiently than many state pairs.

3 Formal Verification

Symbolic model checking allows for the verification of certain temporal properties of state transition systems, where the explicit construction of an entire reachability graph is avoided by implicitly depicting it using symbolic representations. System properties to be checked may be specified as temporal logics formulae and verified with well-known techniques based on image computation as described above. Besides symbolic model checking, often symbolic reachability analysis is performed computing symbolically the set of states reachable from one or several initial states. Using interval diagram techniques for symbolic formal verification essentially is analogous to conventional techniques.

3.1 Process Networks and Petri Nets

The behavior of process networks may be reproduced by decomposing the transition behavior of each process into atomar transitions, changing the internal state of the process and consuming and producing tokens in dependence on the internal state. For Petri nets, this is unnecessary due to the lack of internal states.

Several diverse system models based on Petri nets [3] and process networks [2] have been investigated which led to promising results concerning the numbers of nodes and edges as well as the computation time. For both criteria, IDDs and IMDs turned out to be superior to the conventional approach using BDDs. The necessary numbers of BDD edges were up to about 7 times the numbers of IDD edges, while the BDDs sometimes contained more than 12 times more nodes than the IDDs. Using interval diagram techniques reduced the computation times by a factor of up to more than 5.

3.2 Timed Automata

Timed automata are used to model timing behavior of state transition systems. They are derived from conventional finite state automata and are represented by state-transition graphs with timing constraints using finitely many clocks. Discrete-valued clocks may be regarded as particular integer state variables of which the values increase simultaneously when time progresses. This *time forward projection* may be performed by repeated and simultaneous incrementation of all clock values about a *time distance* of 1, depending on the actual system state and thus similar to state distances.

Using interval diagram techniques, formal verification of timed automata has been performed by reachability analysis on discrete time [1]. Our approach has been compared to another, similar verification technique based on BDDs. For large models, the IDD/IMD approach significantly outperformed the BDD approach with regard to computation time—by a factor of up to 6. Noteworthy is that the weaker slope of the IDD/IMD computation time seems to be an indication that the algorithmic complexity for this kind of application is better. Furthermore, we expect possible improvements of a factor of up to 5 or 10 as we had to compare our experimental implementation with an efficient state-of-the-art tool. Besides timed automata, interval diagram techniques may be applied to the analysis of discrete hybrid automata on integer variables.

4 Symbolic Scheduling

The FunState formalism combines dataflow properties with finite state machine behavior and enables the explicit representation of non-determinism and scheduling using a combination of functional units and state machines. A symbolic scheduling method for heterogeneous embedded systems represented by FunState models has been developed [4]. Symbolic scheduling methods often outperform both ILP and heuristic methods while yielding exact results.

The new scheduling method is able to deal with mixed data/control flow specifications and takes into account different mechanisms of non-determinism as occurring in the design of embedded systems. Constraints imposed by other already implemented components are respected. The scheduling approach avoids the explicit enumeration of execution paths by using symbolic techniques based on interval diagram techniques and guarantees to find a deadlock-free and bounded schedule if one exists. The generated schedule consists of statically scheduled basic blocks which are dynamically called at run time.

References

[1] Karsten Strehl. Using interval diagram techniques for the symbolic verification of timed automata. Technical Report TIK-53, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, July 1998.

[2] Karsten Strehl and Lothar Thiele. Symbolic model checking of process networks using interval diagram techniques. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD-98)*, pages 686–692, San Jose, California, November 8–12, 1998.

[3] Karsten Strehl and Lothar Thiele. Interval diagram techniques for symbolic model checking of Petri nets. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE99)*, Munich, Germany, March 9–12, 1999.

[4] Karsten Strehl, Lothar Thiele, Dirk Ziegenbein, Rolf Ernst, and Jürgen Teich. Scheduling hardware/software systems using symbolic techniques. In *Proceedings of the 7th International Workshop on Hardware/Software Codesign (CODES'99)*, Rome, Italy, May 3–5, 1999.