BDDs: You Love ‘em, You Hate ‘em, You Cannot Live without ‘em (and here’s 1 reason why…)

Jessie Xu, Mark Williams, Hari Mony, Jason Baumgartner

IBM Corporation
Outline

- Preliminaries: Background, Contemporary Hint Status Quo

- Technical Contributions: Automation, Stagnation

- Experiments: Utility of Hints, BDDs vs SAT

- Conclusion
Please don’t tell me you don’t know what BDDs are…

- Reduced Order BDD (ROBDD)
  - Merge isomorphic nodes
  - Remove redundant nodes
Hardware Verification Semantics

- A verification problem may be cast as a **sequential netlist**
  - Recall AIGER: *safety properties* synthesized into simple assertion checks
  - *Assumptions* synthesized as constraints or “input filters”

- A **state** is a valuation to the state variables
  - *Reachable state computation* will solve such verification problems
Reachability Analysis

- Uses BDDs for efficient precise quantification; *breadth-first search*

```plaintext
function FORWARDREACH( TR, init states )
    reached = frontier = initial states
    while (true)
        image = compute_image( TR, frontier )
        frontier = compute_frontier( image, reached )
        if (frontier is empty) then break
        reached = reached \cup frontier
```
Reachability Analysis with Hints

- Problem: intermediate images result in large asymmetric BDDs
  - Final reached BDD may be compact
  - Intermediate blowup due to exploring distinct behaviors in parallel

- Solution: partition BFS into guided fixedpoints via \textit{hints}
Reachability Analysis with Hints

BFS vs. Guided Search for $Gcd$
(max. size for intermediate products)

BFS: 1131s.  Guided Search: 22s.

Greater #images; smaller BDDs

Borrowed from SRC review, covering “Hints to Accelerate Symbolic Traversal” CHARME ‘99
Original Reachability Algorithm with Hints

function FWD_WITH_HINTS( TR, init states, hints )

Final hint must be true to ensure exhaustiveness

while (hint = pop( hints )) do

   hint_TR = constrain( TR, hint )

   FORWARDREACH( hint_TR, reached )

Hints are manually provided and static

Goal: non-true hints get close enough to true that final iteration is easy

“Hints to Accelerate Symbolic Traversal” CHARME ‘99
Practical Observations

- *Arbitrary* hints often useful for *complex* problems

- Effective sequence: $\text{hint}_1 \subseteq \text{hint}_2 \subseteq \ldots \subseteq \text{hint}_i$
  - Then possibly $\text{hint}_1' \subseteq \text{hint}_2' \subseteq \ldots \subseteq \text{hint}_j'$

- Early work cited design insight to manually generate hints
  - Disable certain operations, limit address ranges, …
Outline

- Preliminaries: Background, Contemporary Hint Status Quo

- Technical Contributions: Automation, Stagnation

- Experiments: Utility of Hints, BDDs vs SAT

- Conclusion
Contribution 1: Netlist-Based Hint Generation

- Prior work focused upon manually-generated hints
  - Automated only to extract *branch conditions* in behavioral Verilog *CHARME 2005*

- Not applicable to:
  - Netlists of general format
  - Post-synthesis designs (equiv checking)
  - General types of designs
    - Pipelined, multithreaded, highly concurrent, arbitration, ...
  - A *transformation-based* tool (all HWMCC submissions)
    - Iterated with bit-level abstraction + reduction algorithms
Contribution 1: Netlist-Based Hint Generation

- Solution: derive hints directly from transition relation

- Rank inputs + state variables by how much they reduce $TR$
  - Select literal polarity with greatest reduction

- Greedily select best “N”
  - Proportional to design size; 10 – 15 works well
  - May prune N based upon to $TR$ reduction threshold

- *Predicates* may be more effective than *literals*, though:
  - Nontrivial to determine effective predicates
  - Literals are more efficient to manage with BDDs: cofactoring
Contribution 2: Dynamic Hint Iteration

- Effective hint sequence $\text{hint}_1 \subseteq \text{hint}_2 \subseteq \text{hint}_3 \subseteq \ldots$
  - *Conjunction of literals* become hint
  - Each iteration eliminates one literal

- Literals re-ranked each time a victim is selected
  - DVO occurred since generated: re-ranked literals more apt
  - BDD ops involved in ranking are efficient (literals)
Occasionally the *next* hint does not add any *new states*
- E.g., the design transitions on a function of related inputs

Wasteful to perform image + frontier computation

\[
\text{if } (\text{next-hint AND reached}) \subseteq (\text{current-hint AND reached})
\]
- Skip next-hint as redundant

~15% speedup in overall reachability performance
Contribution 3: Dynamic Hint Introduction

- Hints may degrade performance:
  - Inadequate BDD simplification vs increased #images

- Easy problems: BDD ops already efficient; ~Linear slowdown

- Hard problems: hints may not adequately simplify
Contribution 3: Dynamic Hint Introduction

- Solution: set BDD node limits
  - Threshold exceeded: saturate BDD to *UNKNOWN* value

- Upon *UNKNOWN*: generate more hints, increase limit 150%
  - 350000 nodes a good starting threshold

1) Iterative generation superior to generating all hints at once
   - DVO likely occurred between calls

2) *Iterative* generation superior to *restart* with current var order
   - Existing hints already constraining current BDDs
Contribution 4: Hint Truncation

- Occasionally a hints >> diameter
  - Known issue: stagnation with sparse images

- Pathological example: counter with parallel load port
  - Hint may disable parallel load: exponential diameter increase

- Solution: place upper-bound on #images per hint

- Provably limits increased #images by worst-case linear factor
Outline

- Preliminaries: Background, Contemporary Hint Status Quo
- Technical Contributions: Automation, Stagnation
  - Experiments: Utility of Hints, BDDs vs SAT
- Conclusion
Experimental Setup

- Focused on HWMCC 2011 benchmarks which were
  - Not trivially solved by logic optimization or random simulation
  - Feasible for reachability analysis either with or without hints
  - And, hints were triggered (else no comparison)

- Time limit 4 hours; memory limit 4GB

- Implemented in IBM’s SixthSense toolset
Experiments: Runtime

![Graph showing relationship between reachability with hints and without hints](image)
Experiments: Runtime

- Speedup proportional to benchmark complexity
  - Simpler problems slowed
  - Difficult problems sped up 1-2 orders of magnitude
    - 3 timeouts without hints; 1.8X cumulative speedup ignoring those
    - Often witness better trend in practice: hints **enable** reachability

- BDDs are heuristic! Variable orders, DVO + GC thresholds, …
- **Should hint introduction occur only at larger depth?**
- No; if BDDs too large, *much* time wasted in DVO etc
  - Parallelizable strategy: more- vs less-aggressive hint generation
  - Simpler problems are not a significant practical concern
Experiments: Memory
Experiments: Memory

- Significant clustering due to DVO + GC thresholds
- Simpler problems degrade, difficult problems benefit
#Hints vs Runtime

![Graph showing the relationship between the number of hints and runtime. The graph includes multiple curves representing different datasets: pdtvsarsar29, nusmvbrp, and pdtvsmultimp12. The x-axis represents the number of hints ranging from 10 to 30, while the y-axis represents the reachability computation runtime in seconds, ranging from 0 to 1200.]
#Hints vs Runtime

- Noisy U-shaped pattern
  - U reflects: BDD simplification vs increased #images
  - Noise is intrinsic in BDD-based reachability…
Importance of Reachability Analysis

- SOTA verification tools leverage a large variety of algos
  - Certain algos better-suited to certain problems than others
  - *Relentless push for 100% automation*
  - Algos include: reductions, abstractions, proof, falsification
    - *Many* flavors of each

- SAT-based techniques often held as being most scalable
  - Falsification: BMC, semi-formal extensions, …
  - Verification: induction, interpolation, IC3, …

- Experiment 2: assess performance of BDD vs SAT provers
Importance of Reachability Analysis

- On this benchmark suite:
  - Reachability with hints solved all 92 benchmarks
  - Reachability without hints has 3 timeouts (3.2%)
  - IC3 has 13 timeouts (14.1%)
  - Interpolation and induction each have 41 timeouts (44.6%)

- Not bad! Though…

- Practical verification tools leverage light-weight time-constrained algos before heavier-weight algos
Filtered out benchmarks solvable within 10 seconds

Of the 29 benchmarks remaining
- 7 using IC3 (24.1%)
- 3 using induction (10.3%)
- 19 solved most quickly using reachability (65.5%)

**BDDs Live! (With the proper engineering + heuristics...)**
Outline

- Preliminaries: Background, Contemporary Hint Status Quo
- Technical Contributions: Automation, Stagnation
- Experiments: Utility of Hints, BDDs vs SAT

Conclusion
Conclusions

- Huge disparity in runtime vs. benchmark for various algos
- SAT dominates easy problems
- BDDs Live! For complex problems
  - Easy to discount “Easy for technique X“ as easy…
  - **Hard** problems underrepresented in research?
- Hints are critical to enable complex reachability computation