And-Invert-Graphs (AIGs) for Equivalence Verification, SAT Modulo Theory (SMT) Solvers, and the Motivation for Algebraic Reasoning

Priyank Kalla

Associate Professor
Electrical and Computer Engineering, University of Utah
kalla@ece.utah.edu
http://www.ece.utah.edu/~kalla

August 28, 2017

CDCL Solvers: Panacea?

- Where does SAT fail?
- For hard UNSAT instances, such as equivalence verification

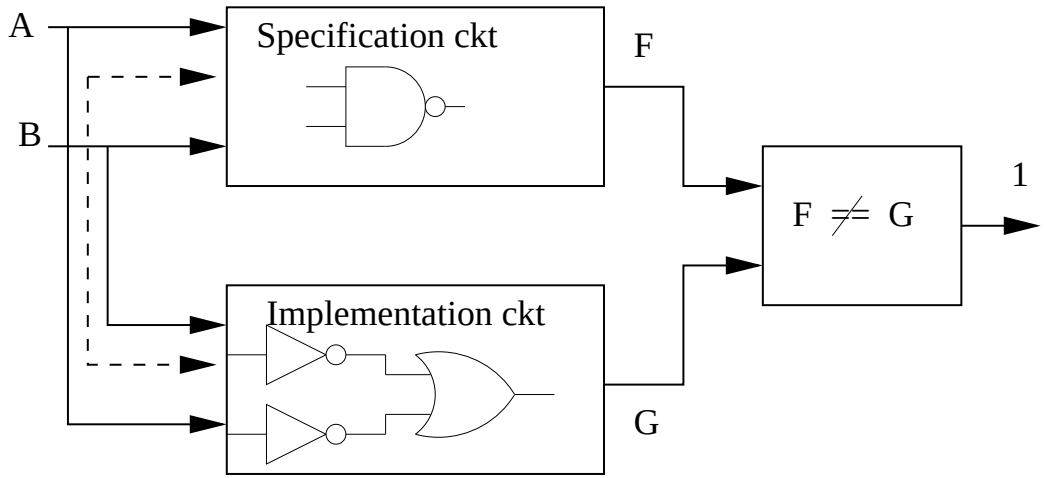


Figure: Miter the circuits F, G

- Prove UNSAT, or find a counter-example
- Limitations: No internal structural equivalences
- EDA-techniques: Circuit-SAT, AIG-reductions, constraint-learning
- Key idea: identify internal structural equivalences

Combinational Equivalence Checking (CEC)

- Direct application of SAT to CEC is inefficient
- Bug-catching (UNSAT) is easier, proof of correctness is harder
- Datapath-dominated circuits are particularly harder to verify
- How to use the power of SAT, along with logic design, synthesis, and optimization concepts, to efficiently solve the CEC problem?
- How was CEC solved prior to SAT and BDDs?
  - Techniques borrowed heavily from circuit synthesis, testing and simulation
  - Logic Synthesis = sequence of transformations
  - Verification = reverse these transformations? Kind of...

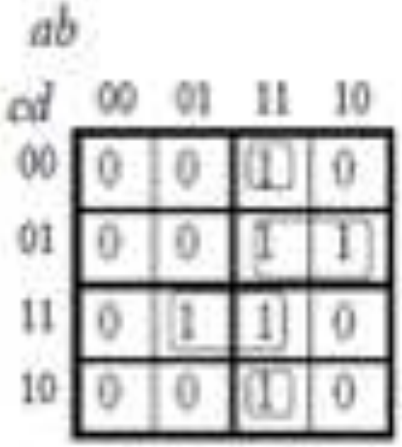
Circuit-SAT solvers & AIGs

- CSAT: SAT solvers, specifically tuned to operate on circuits
- And-Invert-Graphs (AIGs): An engine to enable circuit-SAT
- The origins of AIGs are in logic synthesis and technology decompositions
- AIGs are a versatile data-structure to represent Boolean functions and circuits
- AIGs can be functionally reduced (FRAIGs)
- FRAIGs are semi-canonical, help to identify sub-circuit equivalences
- The tool ABC from UC Berkeley (URL on class website): AIG based logic synthesis and verification

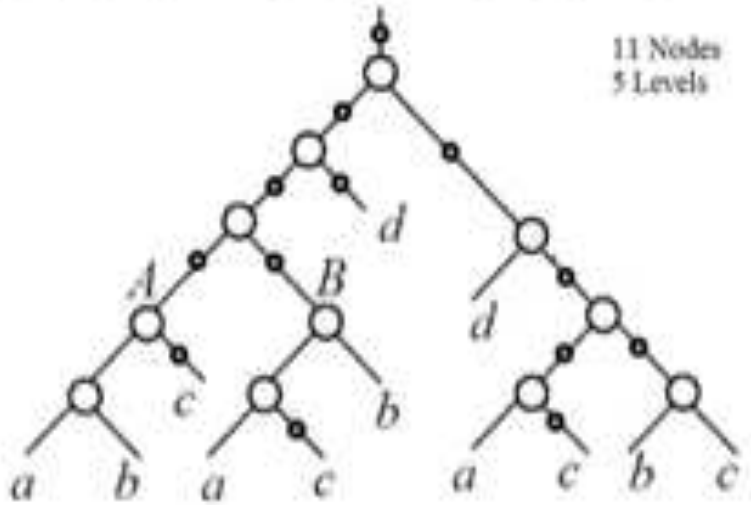
AIGs

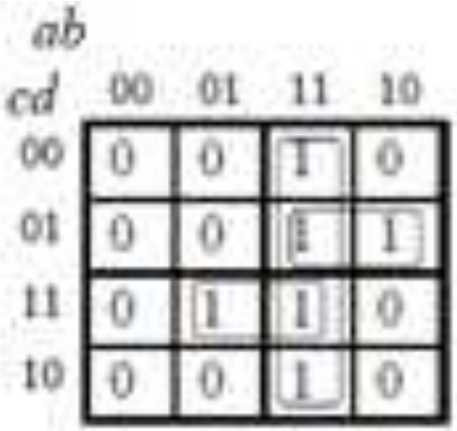
- AIGs are Boolean networks composed of 2-input AND gates and Inverters
- Construction time proportional to circuit size (unlike BDDs)
- Enhanced with Simulation, SAT & BDDs: very powerful for synthesis and verification
- Build AIGs from circuits, FRAIG-sweep, solve SAT, CEC, Synthesis, etc.

AIGs - Examples

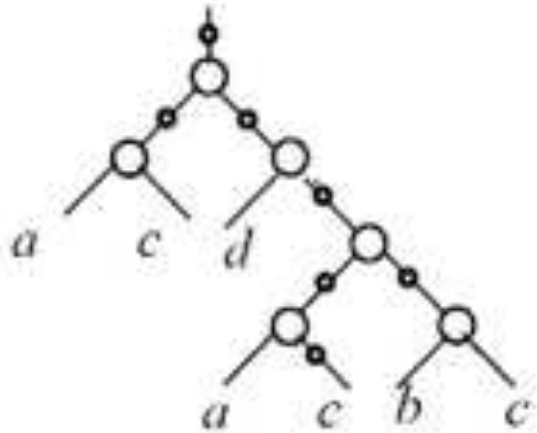


$$F(a,b,c,d) = \bar{d}((ab)\bar{c} + a(b\bar{c})) + d(a\bar{c} + bc)$$





$$F(a,b,c,d) = ac + d(a\bar{c} + bc)$$



6 nodes
4 levels

FRAIG: AIG re-write rules

- Simple rules, non canonical, but very quick AIG rewriting
- Swap inputs, merge nodes, look-up sub-structures

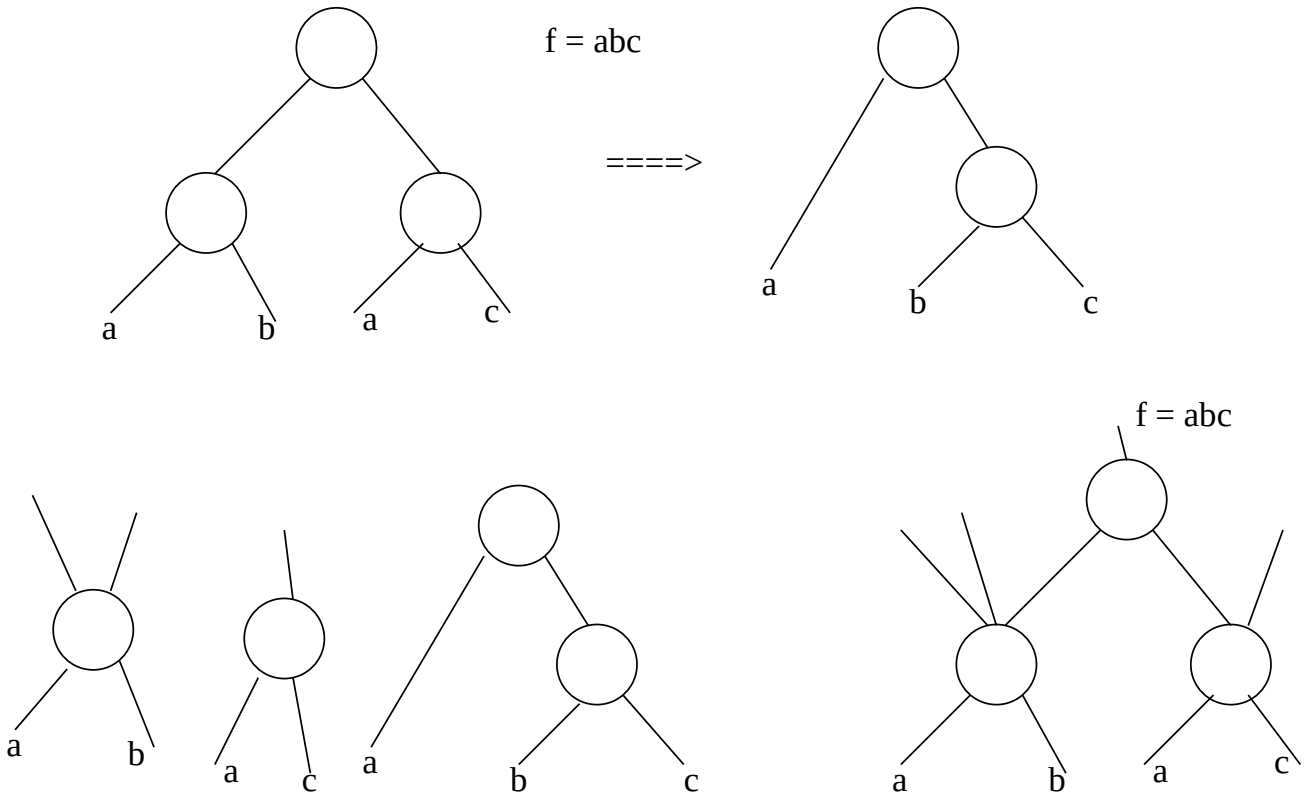


Figure: AIG rewrite examples

AIG-based CEC

- Construct FRAIGs, merge equivalent nodes
- Simulate for a few (say $I = 2^{16}$) inputs
- If nodes n_1, n_2 evaluate the same for I inputs
  - Miter sub-circuits at n_1 and n_2, solve sub-circuit CEC
  - If $n_1 = n_2$, simplify original miter: make $n_1 = n_2$ a primary input; continue until CEC solved.
- Very simple, yet very successful approach, used in industry
- AIGs can solve CEC for bit-level and synthesized designs

Word-Level Verification

- Imagine a Bit-Vector RTL description

Word-Level Verification

- Imagine a Bit-Vector RTL description
- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$

Word-Level Verification

- Imagine a Bit-Vector RTL description
- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$
- How will you solve SAT on this formula?

Word-Level Verification

- Imagine a Bit-Vector RTL description
- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$
- How will you solve SAT on this formula?
- Also, x, y, z are bit-vectors: $[31 : 0]$

Word-Level Verification

- Imagine a Bit-Vector RTL description
- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$
- How will you solve SAT on this formula?
- Also, x, y, z are bit-vectors: $[31 : 0]$
- $\underbrace{(x > y)}_{a} \vee \underbrace{(x < y)}_{b} \wedge (\underbrace{(2 * x < z)}_{c} \vee \neg(\underbrace{(x - y \geq z)}_{d} \wedge \underbrace{(z \leq y)}_{e}))$

Word-Level Verification

- Imagine a Bit-Vector RTL description
- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$
- How will you solve SAT on this formula?
- Also, x, y, z are bit-vectors: $[31 : 0]$
- $\underbrace{(x > y)}_{a} \vee \underbrace{(x < y)}_{b} \wedge (\underbrace{(2 * x < z)}_{c} \vee \neg(\underbrace{(x - y \geq z)}_{d} \wedge \underbrace{(z \leq y)}_{e}))$
- Solve SAT: $(a \vee b) \wedge (c \vee \neg(d \wedge e))$

Word-Level Verification

- Imagine a Bit-Vector RTL description
- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$
- How will you solve SAT on this formula?
- Also, x, y, z are bit-vectors: $[31:0]$
- $\underbrace{(x > y)}_{a} \vee \underbrace{(x < y)}_{b} \wedge (\underbrace{(2 * x < z)}_{c} \vee \neg(\underbrace{(x - y \geq z)}_{d} \wedge \underbrace{(z \leq y)}_{e}))$
- Solve SAT: $(a \vee b) \wedge (c \vee \neg(d \wedge e))$
- Solution: $a = b = c = d = e = 1$

Word-Level Verification

- Imagine a Bit-Vector RTL description
- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$
- How will you solve SAT on this formula?
- Also, x, y, z are bit-vectors: $[31 : 0]$
- $\underbrace{(x > y)}_{a} \vee \underbrace{(x < y)}_{b} \wedge (\underbrace{(2 * x < z)}_{c} \vee \neg(\underbrace{(x - y \geq z)}_{d} \wedge \underbrace{(z \leq y)}_{e}))$
- Solve SAT: $(a \vee b) \wedge (c \vee \neg(d \wedge e))$
- Solution: $a = b = c = d = e = 1$
- Combine "solvers" for different theories!

SAT Modulo Theories (SMT)

- A mechanism to combine many "theories" and solvers together
  - Theory of difference constraints and logic
  - Equality and uninterpreted functions
  - Quantifier-free bit-vector formulas
  - All combined with First order logic
- Approach: Use SAT as a base-solver, and propagate solutions to theory solvers
- Spurious solutions (ones disproved with theory solvers) are added as "lemma", and SAT is re-solved
- See example on next slide

SMT Solving Strategies

- $(x \neq y) \wedge ((2 * x < z) \vee \neg((x - y \geq z) \wedge (z \leq y)))$
- $\underbrace{(x > y)}_{a} \vee \underbrace{(x < y)}_{b} \wedge (\underbrace{(2 * x < z)}_{c} \vee \neg(\underbrace{(x - y \geq z)}_{d} \wedge \underbrace{(z \leq y)}_{e}))$
- Solve SAT $(a \vee b) \wedge (c \vee \neg(d \wedge e))$
- Solution: $a = b = c = d = e = 1$ creates a linear program
- If linear program infeasible, add $\neg(a \wedge b \wedge c \wedge d \wedge e)$ to the CNF, resolve SAT

Word-Level RTL CEC is still Challenging

- Multiplication is hard to solve (no one knows how to solve it!)
- SMT relies on "bit-blasting", and gives a huge problem to SAT

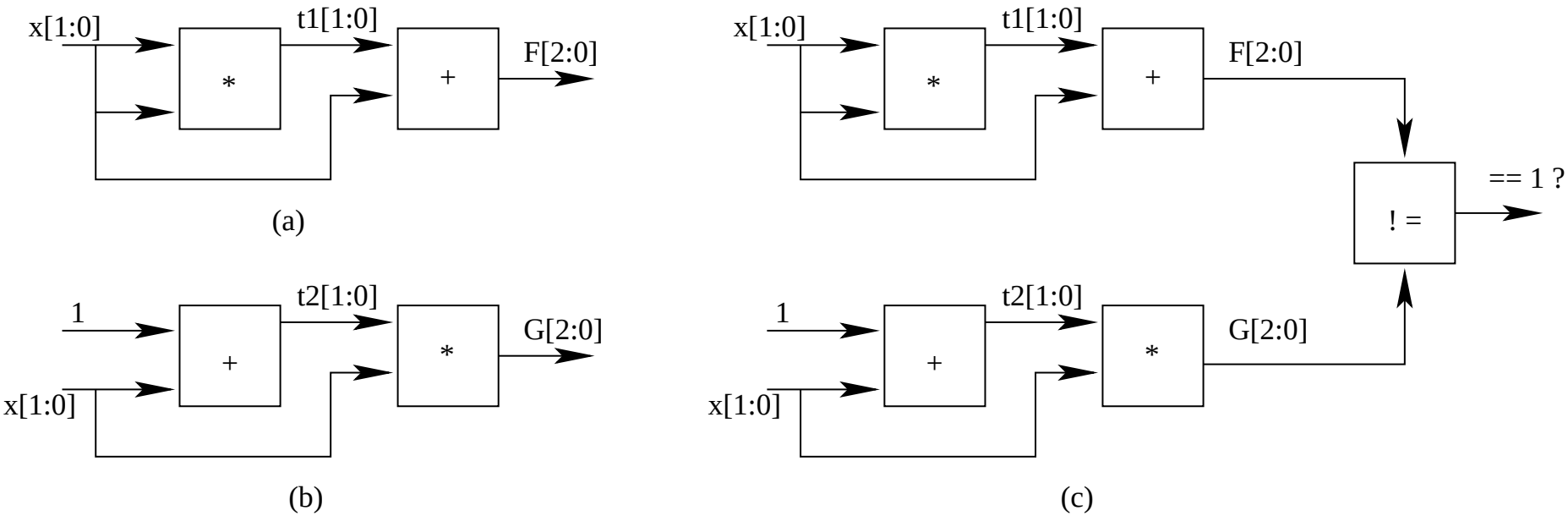


Figure: $x^2 + x \equiv x(x+1)$

Motivation for Algebraic Computation

- Modeling for bit-precise algebraic computation
  - Arithmetic RTLs: functions over k-bit-vectors
  - k-bit-vector $\mapsto$ integers $\pmod{2^k} = \mathbb{Z}_{2^k}$
  - k-bit-vector $\mapsto$ Galois (Finite) field $\mathbb{F}_{2^k}$
- For many of these applications SAT/SMT fail miserably!
- Computer Algebra and Algebraic Geometry + SAT/SMT
  - Model: Circuits as polynomial functions $f : \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^k}$, $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$
  - Apply symbolic and algebraic computing concepts for verification
  - And this topic is the core focus of this course