Equivalence Checking

Sean Weaver

Equivalence Checking

- Given two Boolean functions, prove whether or not two they are functionally equivalent
- This talk focuses specifically on the mechanics of checking the equivalence of pairs of combinational circuits

Types of Circuits

- Combinational Circuit
  - Digital circuit
  - No state-holding elements
  - No feedback loops
  - Output is a function of the current input
- Sequential Circuit
  - Can have state-holding elements
  - Can have feedback loops
  - Must transform (e.g. BMC) into a combinational circuit for equivalence checking

Circuit Equivalence Checking

- Checking the equivalence of a pair of circuits
  - For all possible input vectors ($2^{\#\text{input bits}}$), the outputs of the two circuits must be equivalent
  - Testing all possible input-output pairs is CoNP-Hard
  - However, the equivalence check of circuits with "similar" structure is easy [1]
  - So, we must be able to identify shared structure, and we need a tool that can efficiently solve NP-Complete problems (Satisfiability solver, BDDs, Gröbner Basis solver, etc.)

1. E. Goldberg, Y. Novikov. How good can a resolution based SAT-solver be? SAT-2003, LNCS 2919, pp. 35-52.

Equivalence Checking Uses

- Formal Verification
  - Prove whether a low level implementation matches a high level, or mathematical, specification
- Verifying Compiler
  - Maintain the functionality of generated code
- Version Control
  - Use previous implementations to maintain the correctness of future implementations
- Functional Inversion
  - Prove whether encode and decode functions are inverses of each other

Functional Verification of Hardware Design

A reasonable *functional specification* **of any 1-bit adder**:

$$(X \Leftrightarrow (A \wedge \bar{B} \wedge \bar{C}) \vee (\bar{A} \wedge B \wedge \bar{C}) \vee (\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge B \wedge C)) \wedge$$
$$(Y \Leftrightarrow (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)).$$

A proposed *implementation* **of a 1-bit adder**:

$$\begin{aligned}
&(u \Leftrightarrow (A \wedge \bar{B}) \vee (\bar{A} \wedge B)) \wedge \\
&(v \Leftrightarrow u \wedge C) \wedge \\
&(w \Leftrightarrow A \wedge B) \wedge \\
&(X \Leftrightarrow (u \wedge \bar{C}) \vee (\bar{u} \wedge C)) \wedge \\
&(Y \Leftrightarrow w \vee v).
\end{aligned}$$

Call these formulas $\psi_S(A, B, C, X, Y)$ and $\psi_I(A, B, C, X, Y, u, v, w)$.

The theorem we are trying to prove is:

$$\psi_S(A, B, C, X, Y) \Leftrightarrow \exists u, v, w : \psi_I(A, B, C, X, Y, u, v, w).$$

Functional Verification of Hardware Design

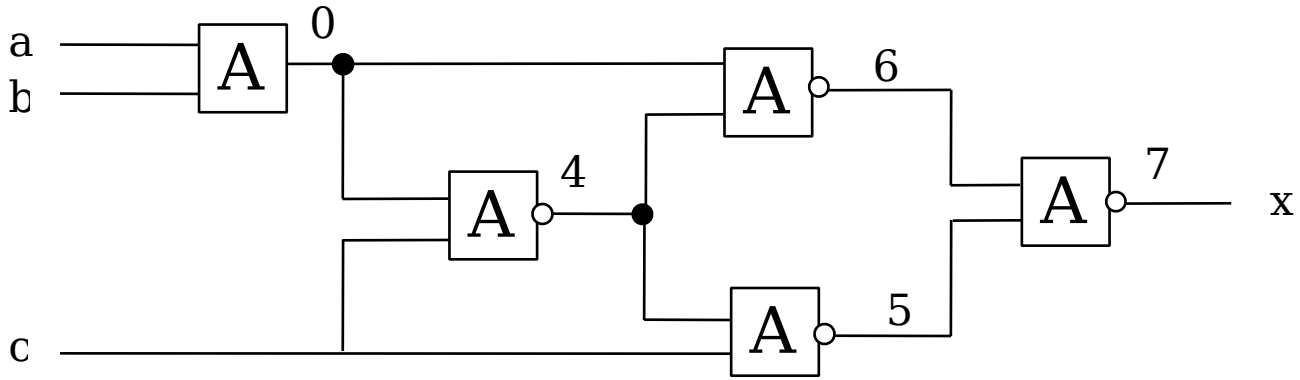
- Given that the input variables (A, B, C) are equivalent, verify output variables (X, Y) are equivalent.
  1. Conjoin specification and implementation formulas,
  2. Add the equivalence checking constraint.
- Result –

$$
\begin{aligned}
&(X \Leftrightarrow (A \wedge \bar{B} \wedge \bar{C}) \vee (\bar{A} \wedge B \wedge \bar{C}) \vee (\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge B \wedge C)) \wedge \\
&(Y \Leftrightarrow (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)) \wedge \\
&(u \Leftrightarrow (A \wedge \bar{B}) \vee (\bar{A} \wedge B)) \wedge \\
&(v \Leftrightarrow u \wedge C) \wedge \\
&(w \Leftrightarrow A \wedge B) \wedge \\
&(X' \Leftrightarrow (u \wedge \bar{C}) \vee (\bar{u} \wedge C)) \wedge \\
&(Y' \Leftrightarrow w \vee v) \wedge \\
&((X \oplus X') \vee (Y \oplus Y')).
\end{aligned}
$$

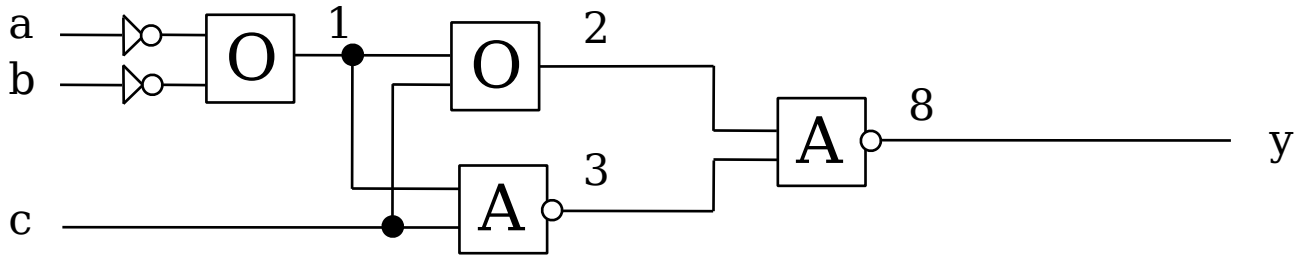
This is called a “miter” formula. If unsatisfiable, the specification and implementation are equivalent. A SAT solver can tell us this.

Example: Are These Circuits Equivalent?

#1



#2



Example: Outline

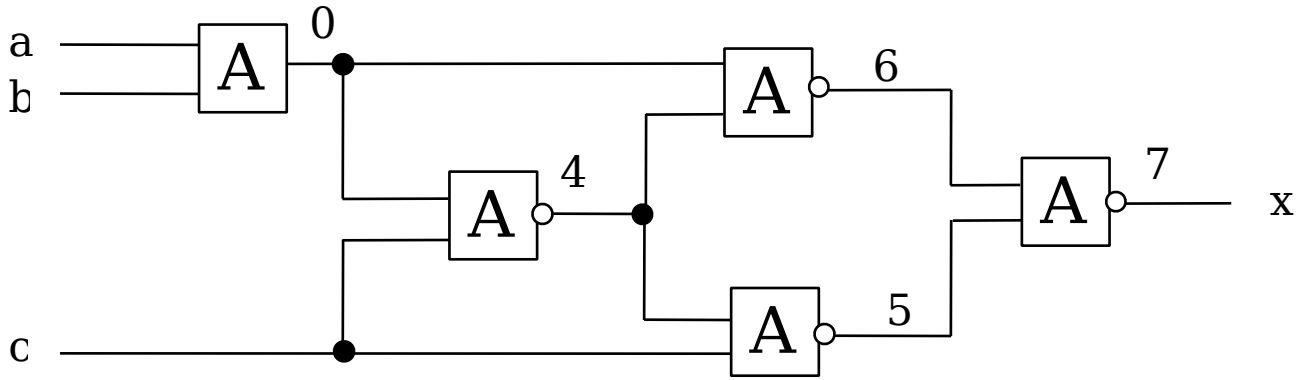
- Random Simulation -
  - Send random vectors through the two circuits, collecting pairs of candidate equivalent nodes
- And/Inverter Graph
  - Find more equivalent nodes by creating the AIG of the circuits
- SAT Sweeping
  - Use candidate equivalent nodes to guide SAT searches, merging AIG nodes which reduces the complexity of future SAT searches

Identifying Shared Structure

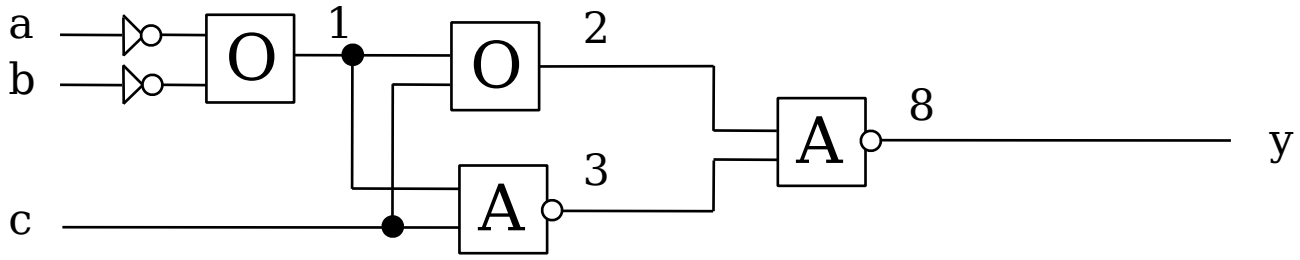
- An internal node in the first circuit may be equivalent to an internal node in the second circuit
- Detect by using random simulation
  - Percolate random vectors through both circuits (fast trick - use 64-bit words)
  - Partition nodes into equivalence classes
  - This can detect potentially many, high probability, candidate equivalent nodes

Random Simulation

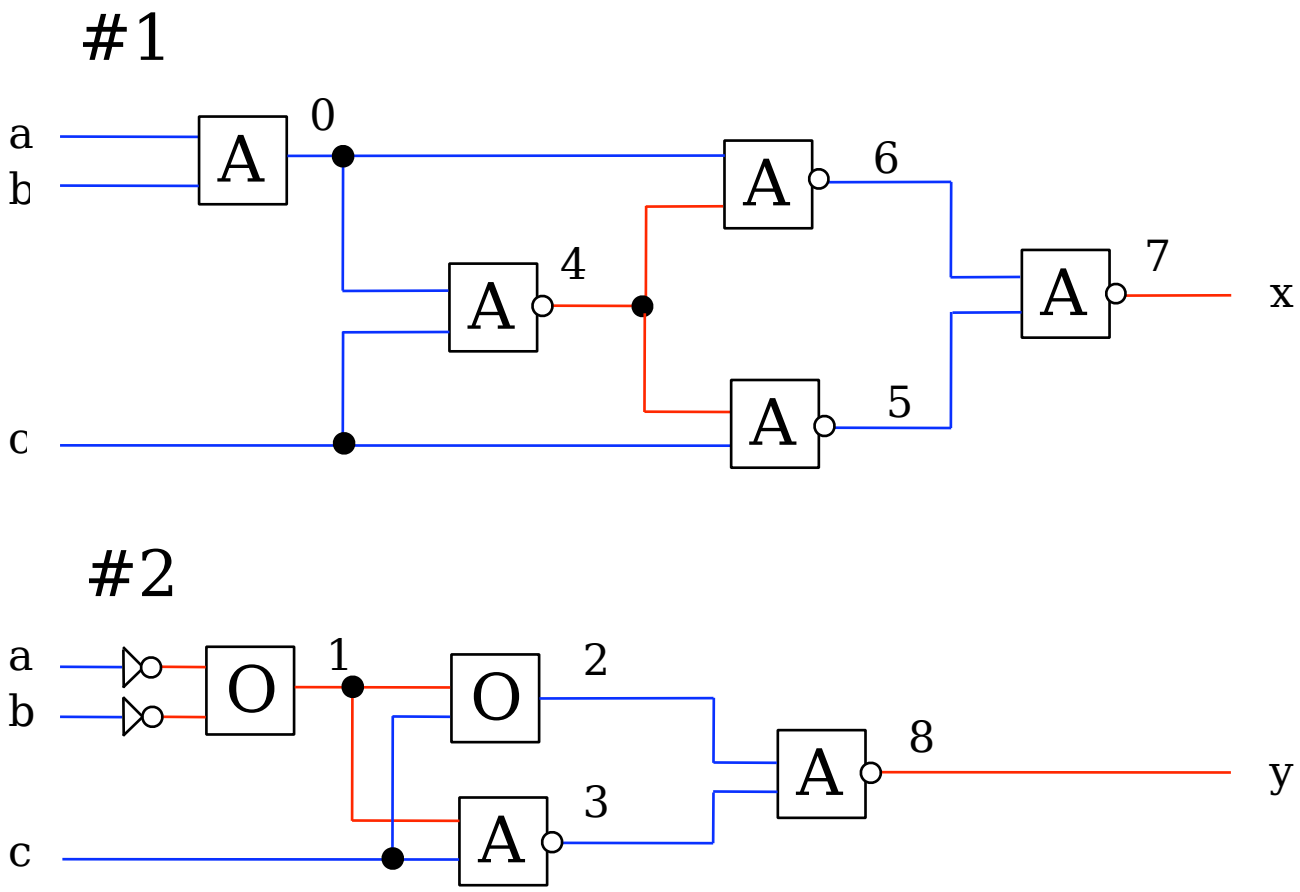
#1



#2



Random Simulation



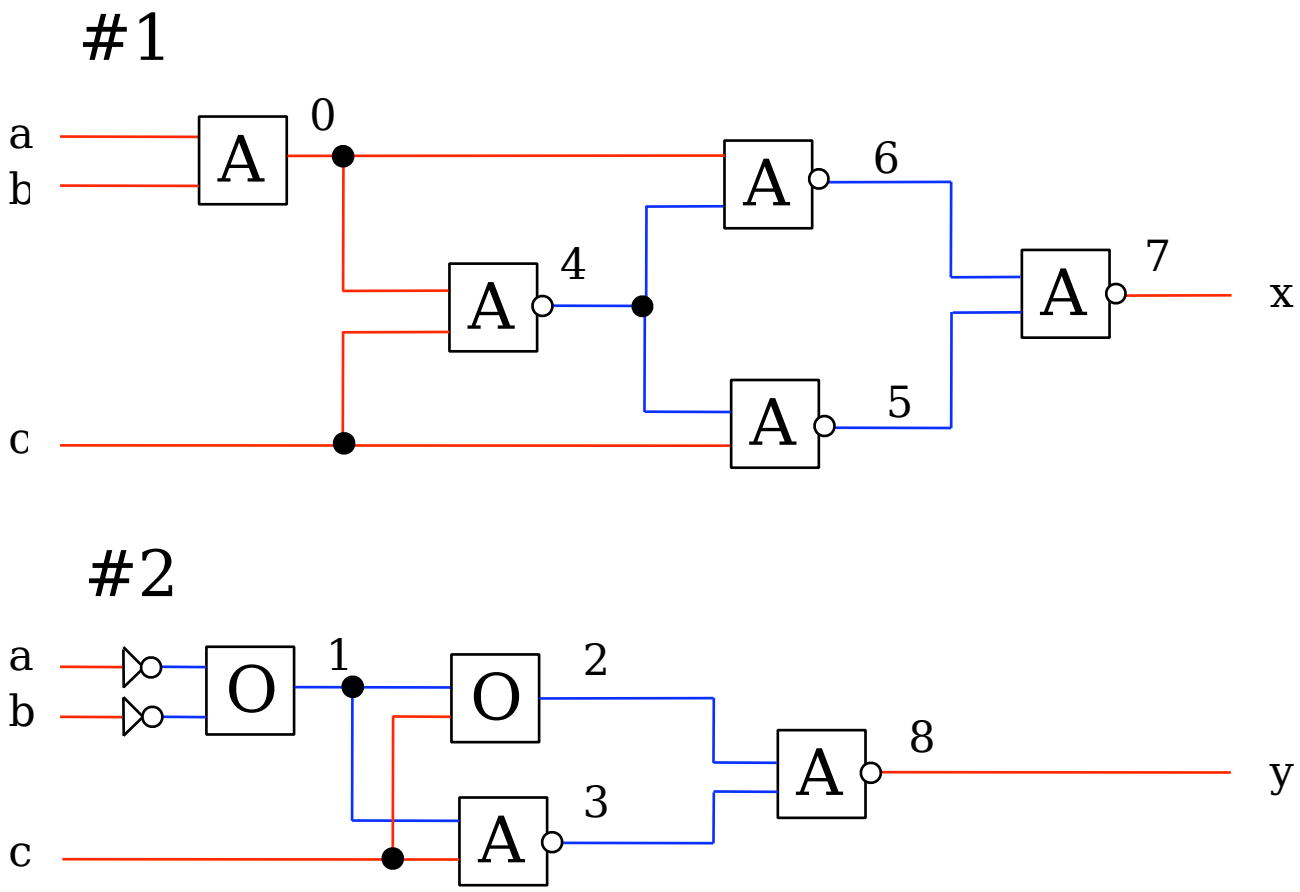
Buckets

1,4,7,8

0,2,3,5,6

Random Vector: {a=T, b=T, c=T}

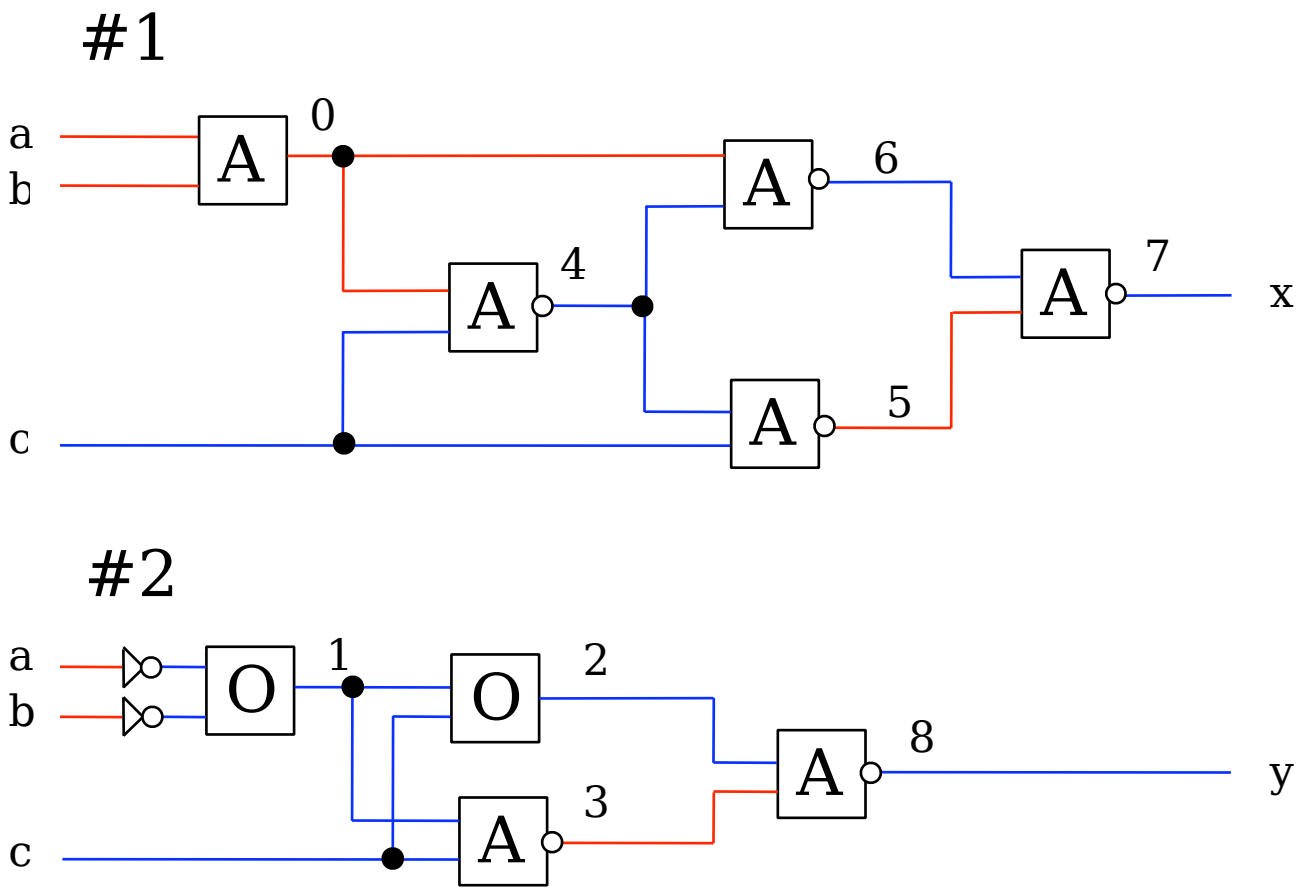
Random Simulation



Buckets
1,4
7,8
2,3,5,6
0

Random Vector: {a=F, b=F, c=F}

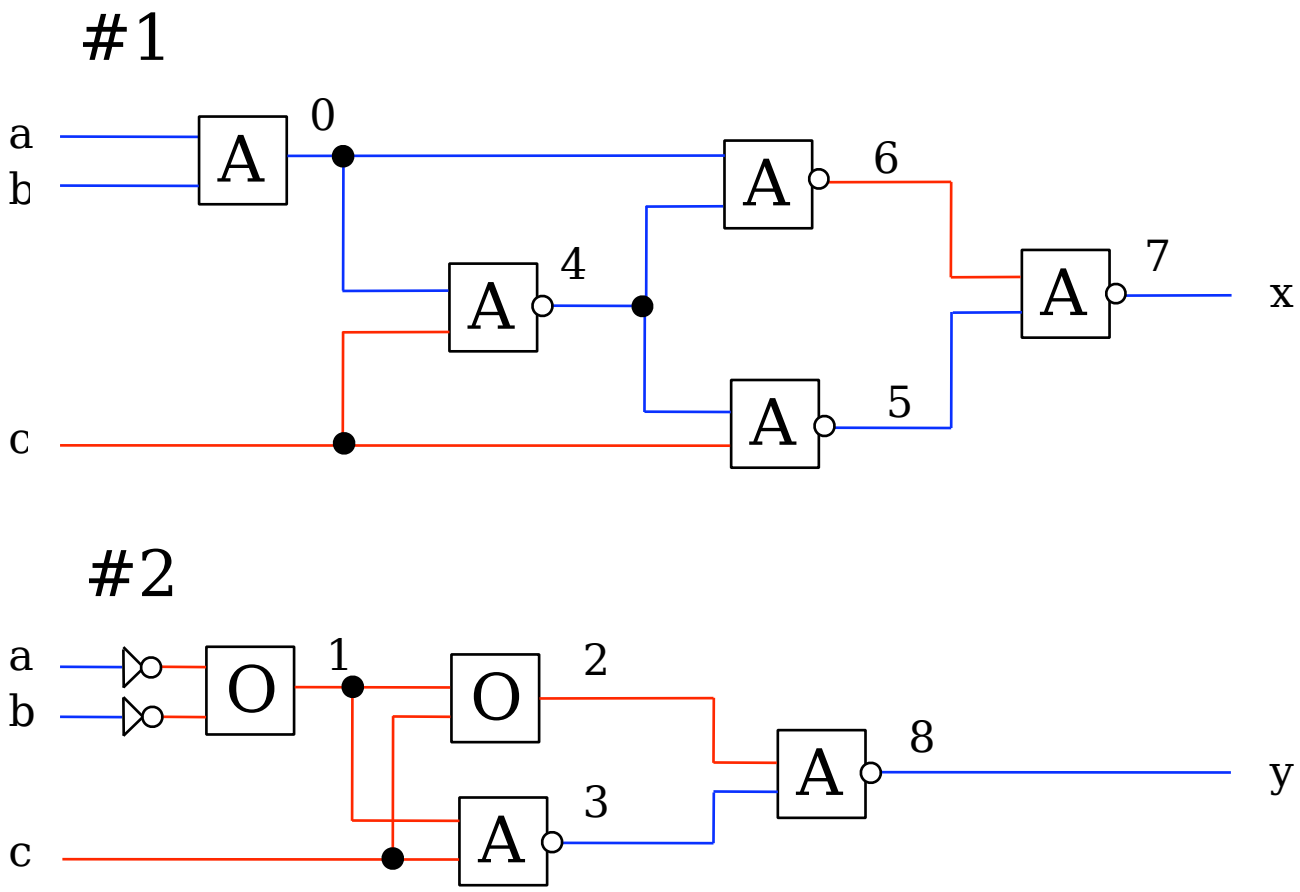
Random Simulation



Buckets
1,4
7,8
2,6
3,5

Random Vector: {a=F, b=F, c=T}

Random Simulation



Random Vector: {a=T, b=T, c=F}

Buckets

7,8

2,6

3,5

Example: Outline

- Random Simulation -
  - Send random vectors through the two circuits, collecting pairs of candidate equivalent nodes
- And/Inverter Graph
  - Find more equivalent nodes by creating the AIG of the circuits
- SAT Sweeping
  - Use candidate equivalent nodes to guide SAT searches, merging AIG nodes which reduces the complexity of future SAT searches

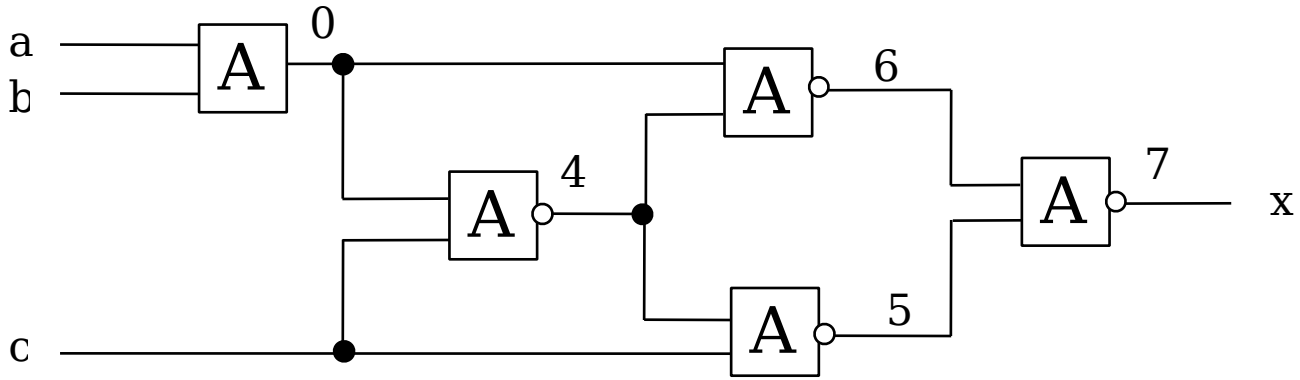
Identifying Shared Structure

- Random simulation is probabilistic
- And/Inverter Graph (AIG) [2]
  - Simple data structure used to represent combinational circuits
  - Operations are fast (add node, merge nodes)

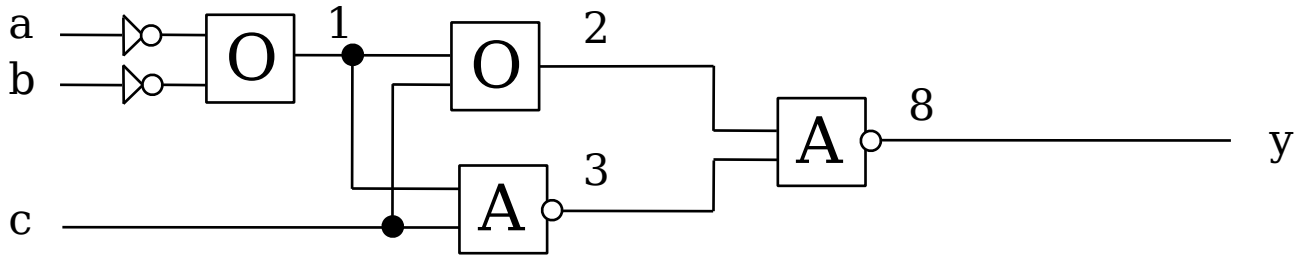
2. A. Kuehlmann, V. Paruthi, F. Krohm, and M.K. Ganai. Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. IEEE Trans. CAD, Vol. 21, No. 12, pp. 1377-1394 (2002)

And/Inverter Graph

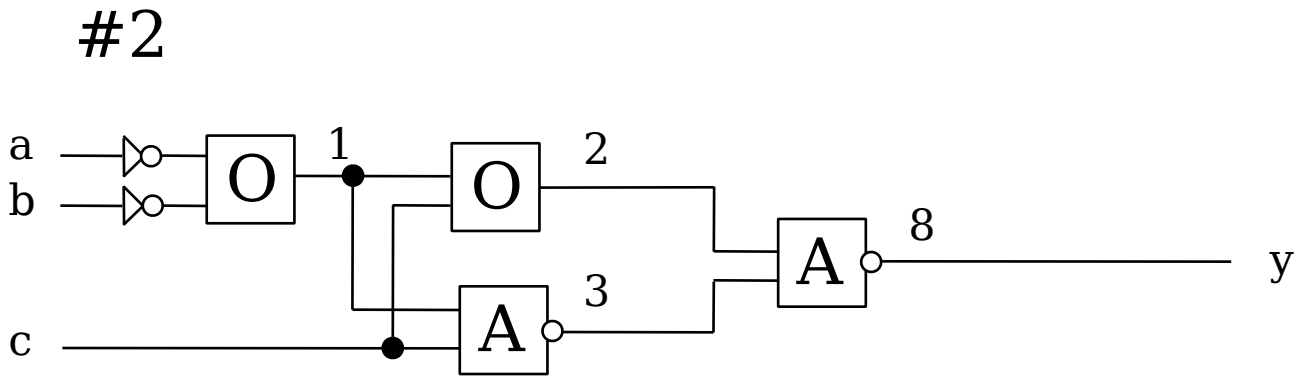
#1



#2

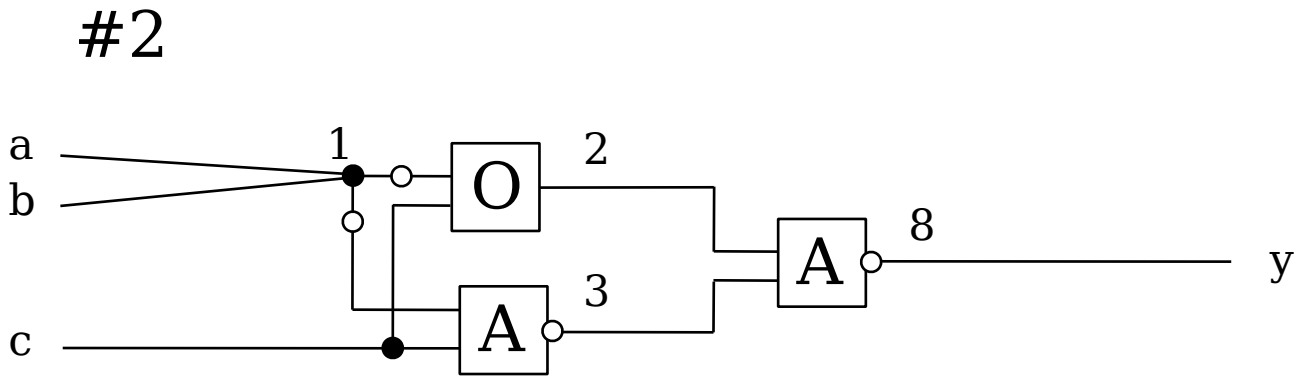


And/Inverter Graph



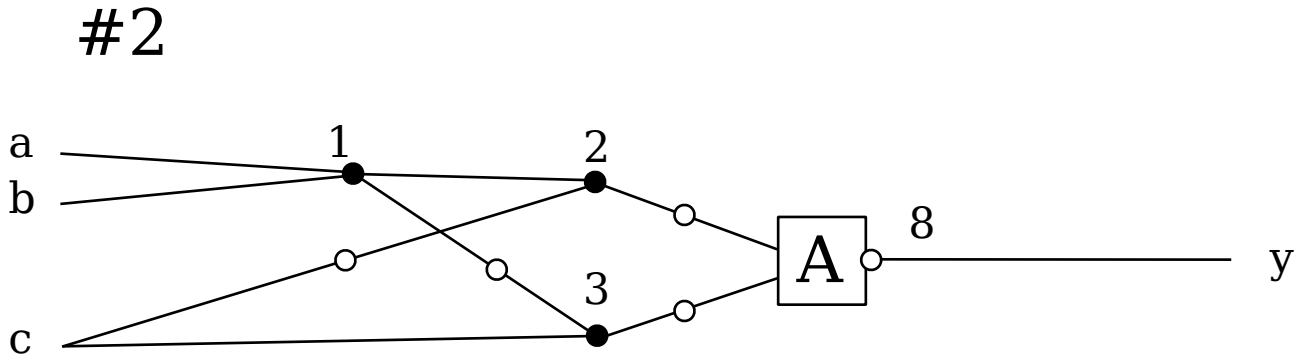
- Use the AIG data structure to store circuits
- AIG can quickly add nodes and merge equivalent nodes
- Structural hashing is used
- Merging a pair of equivalent nodes can cause other nodes to be merged automatically, without need for a SAT proof

And/Inverter Graph

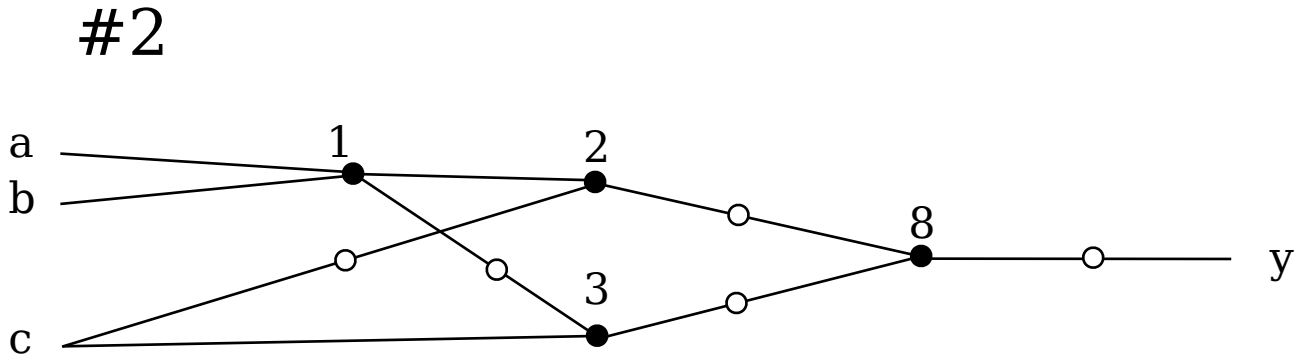


- Nodes represent AND gates
- Edges represent inputs to an AND gate
- Edges may be inverted
- OR gates must be converted to AND gates during AIG creation

And/Inverter Graph

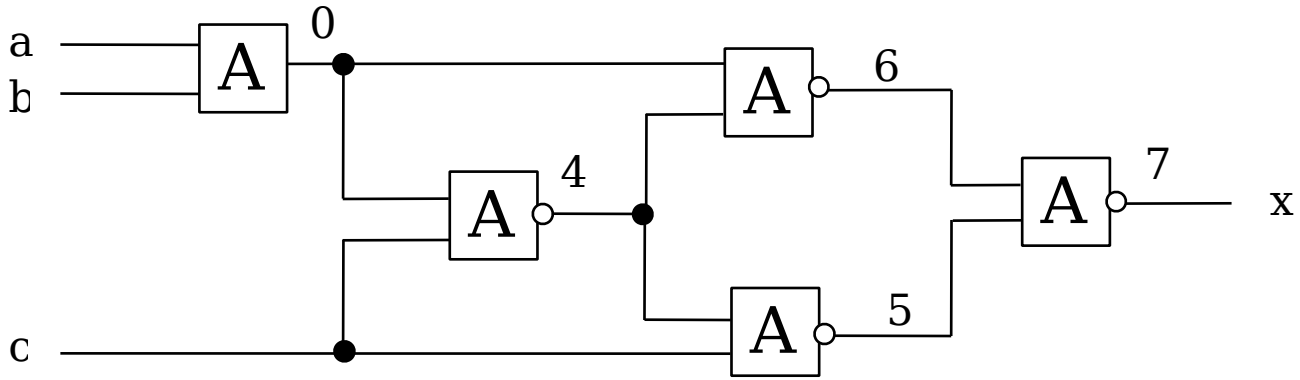


And/Inverter Graph

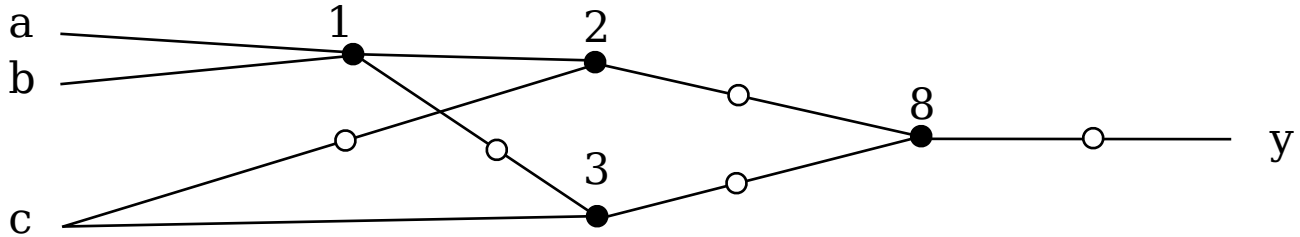


And/Inverter Graph

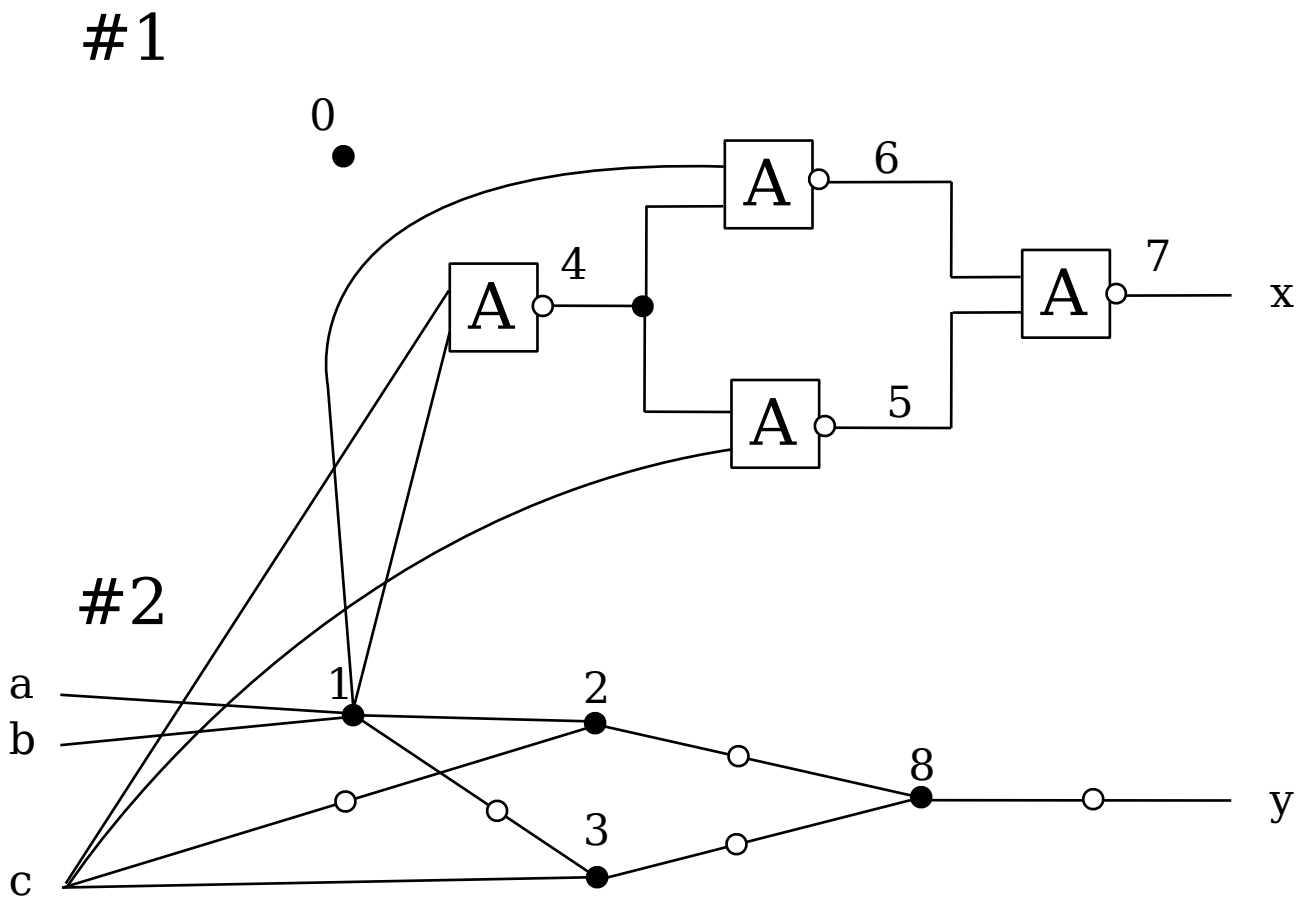
#1



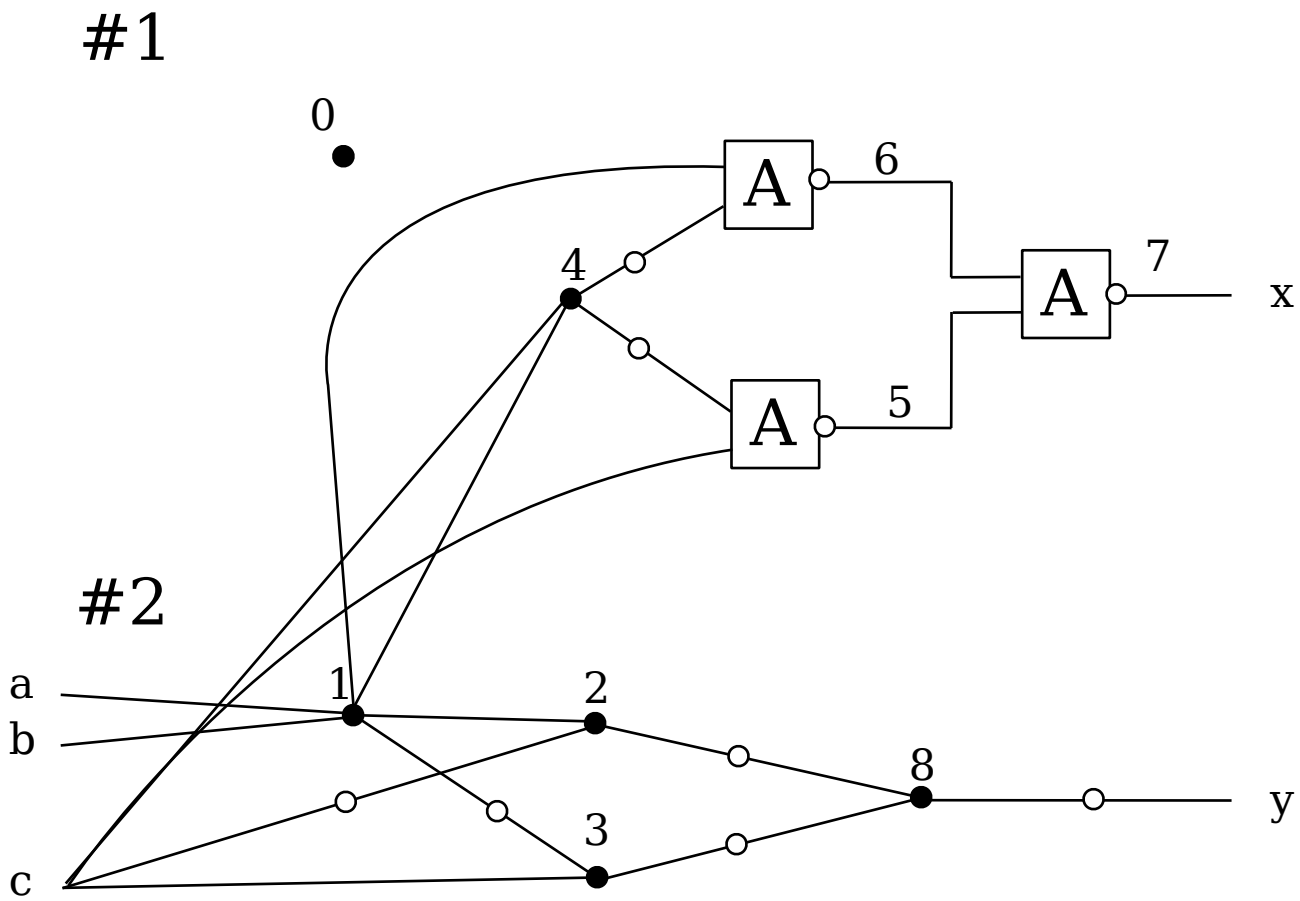
#2



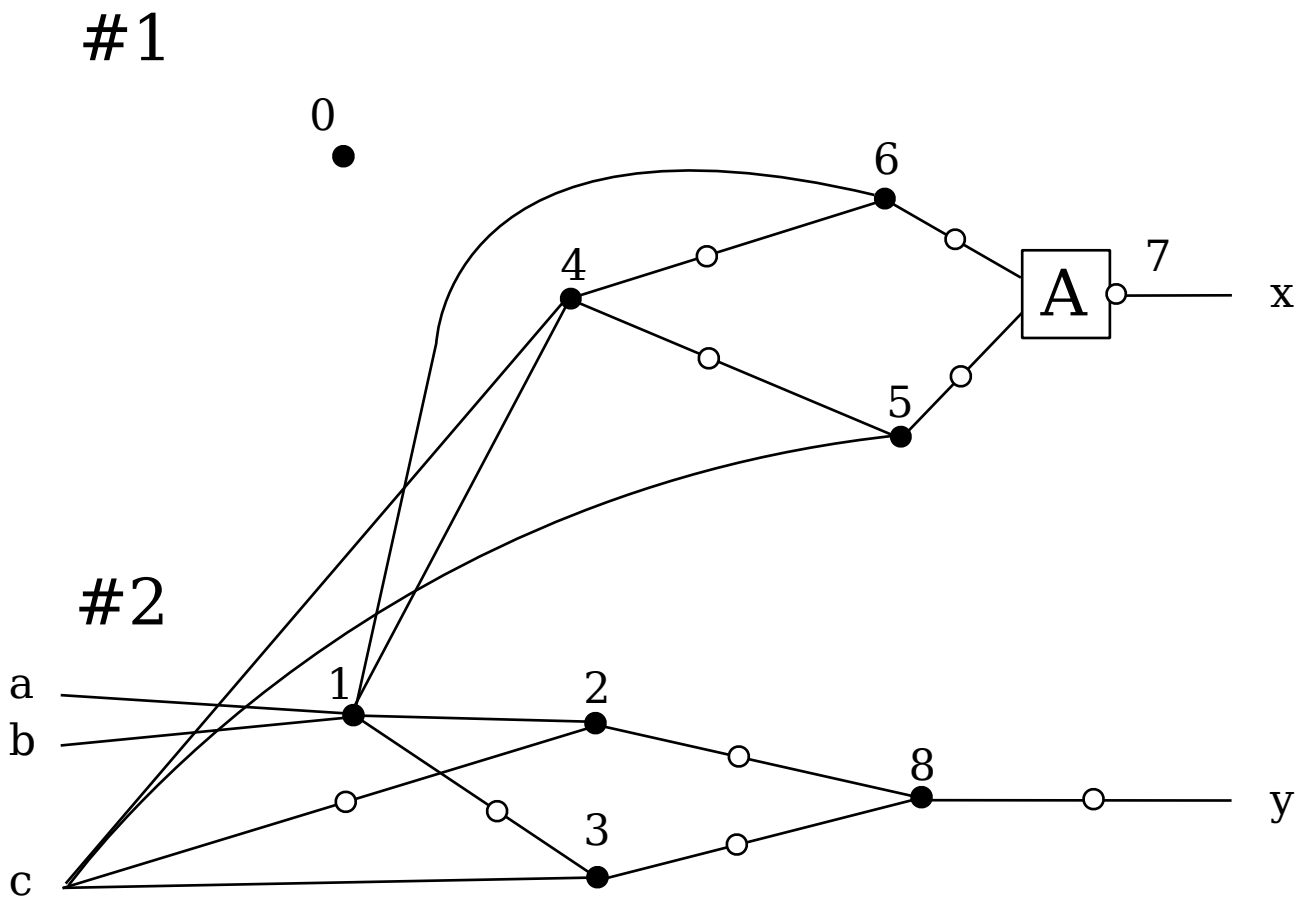
And/Inverter Graph



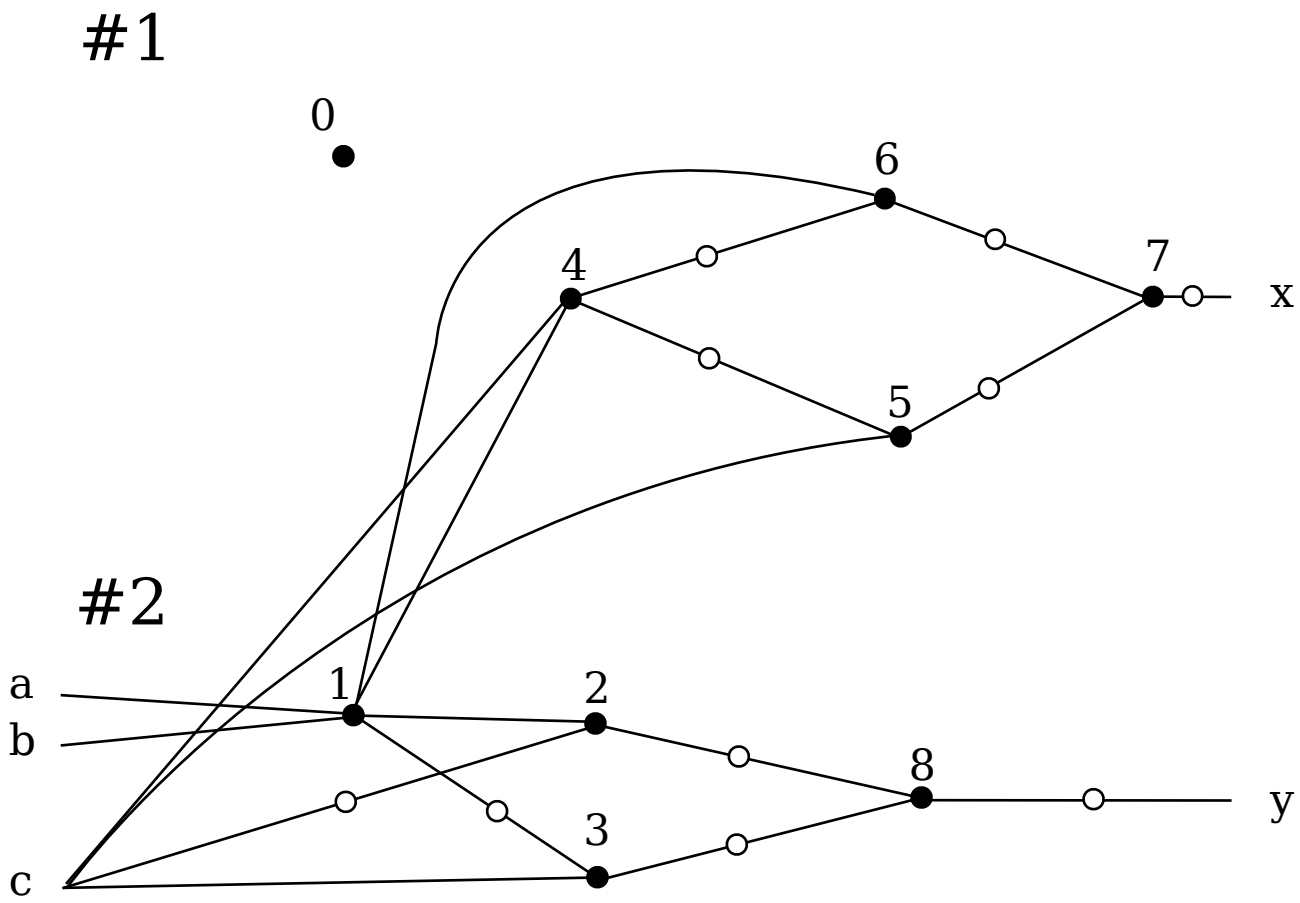
And/Inverter Graph



And/Inverter Graph



And/Inverter Graph



Example: Outline

- Random Simulation -
  - Send random vectors through the two circuits, collecting pairs of candidate equivalent nodes
- And/Inverter Graph
  - Find more equivalent nodes by creating the AIG of the circuits
- SAT Sweeping
  - Use candidate equivalent nodes to guide SAT searches, merging AIG nodes which reduces the complexity of future SAT searches

SAT Sweeping

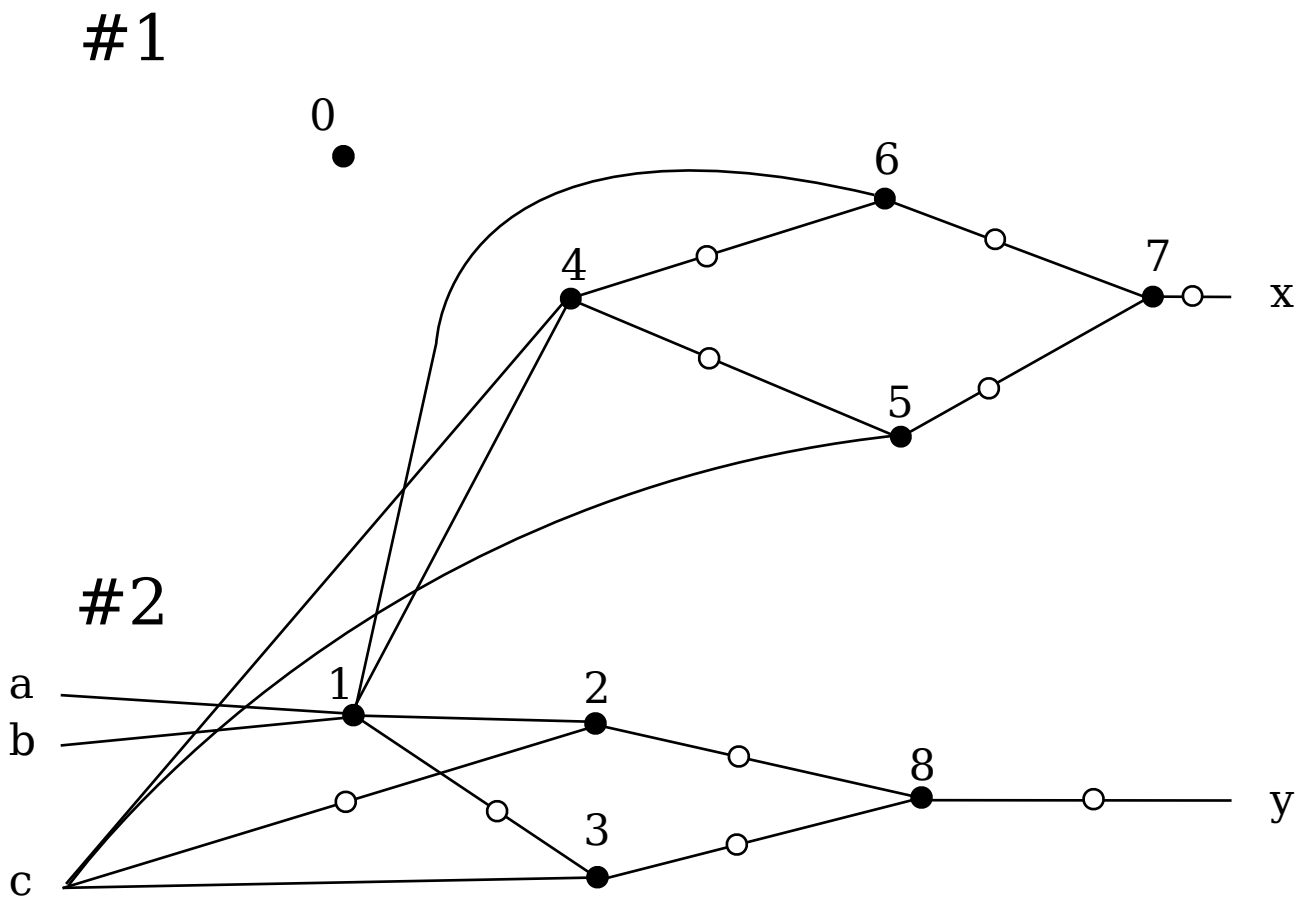
- Use SAT to prove whether or not the candidate equivalent nodes, from the random simulation phase, are equivalent
- The candidate equivalent nodes are used as cut points
- Generate SAT problems that are solved from inputs to outputs using the candidate equivalent nodes as a guide [3]

3. A. Kuehlmann. Dynamic Transition Relation Simplification for Bounded Property Checking. In ICCAD, 2004

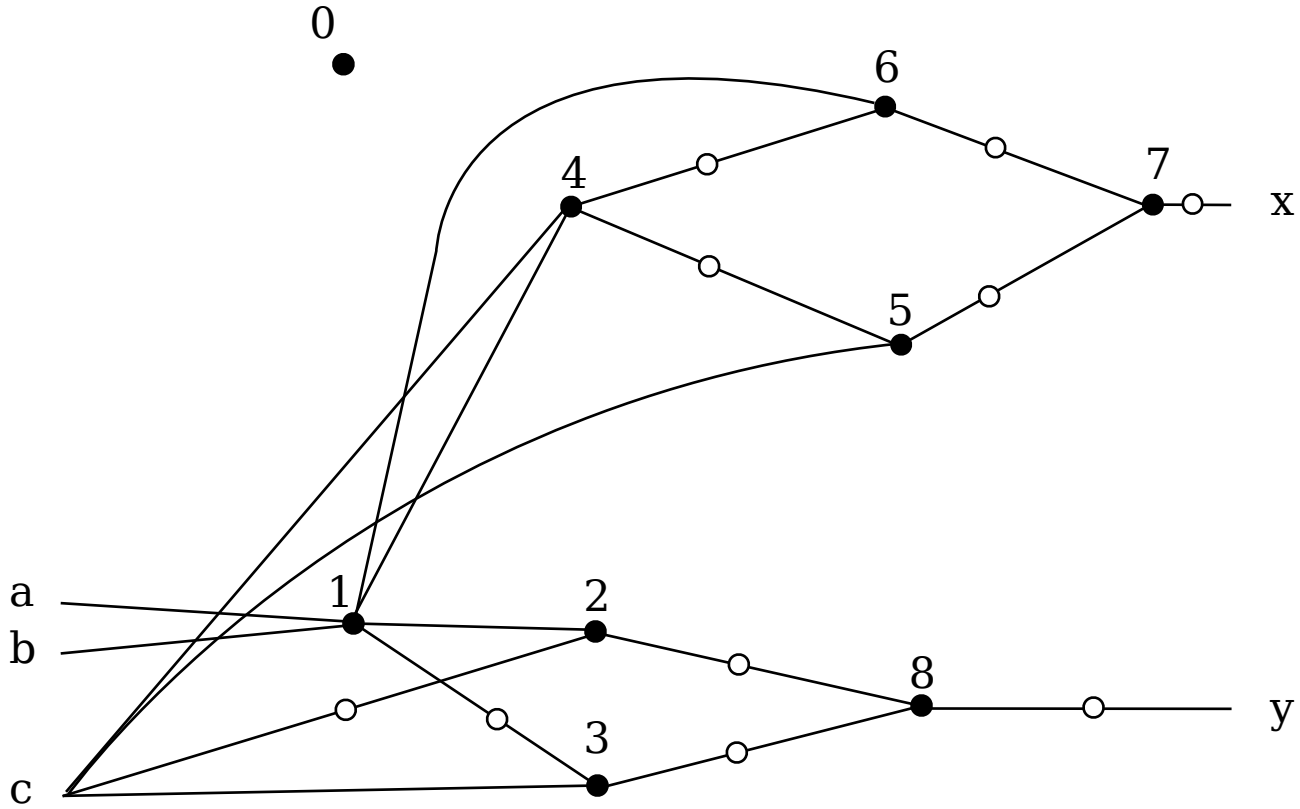
SAT Instances

- Create one SAT instance for one (or more) pair of candidate equivalent nodes
- A SAT instance encodes a miter circuit
- Each SAT search can result in the merger of equivalent AIG nodes, reducing the complexity of the AIG

And/Inverter Graph



Equivalences



Buckets

---

7 ?= 8

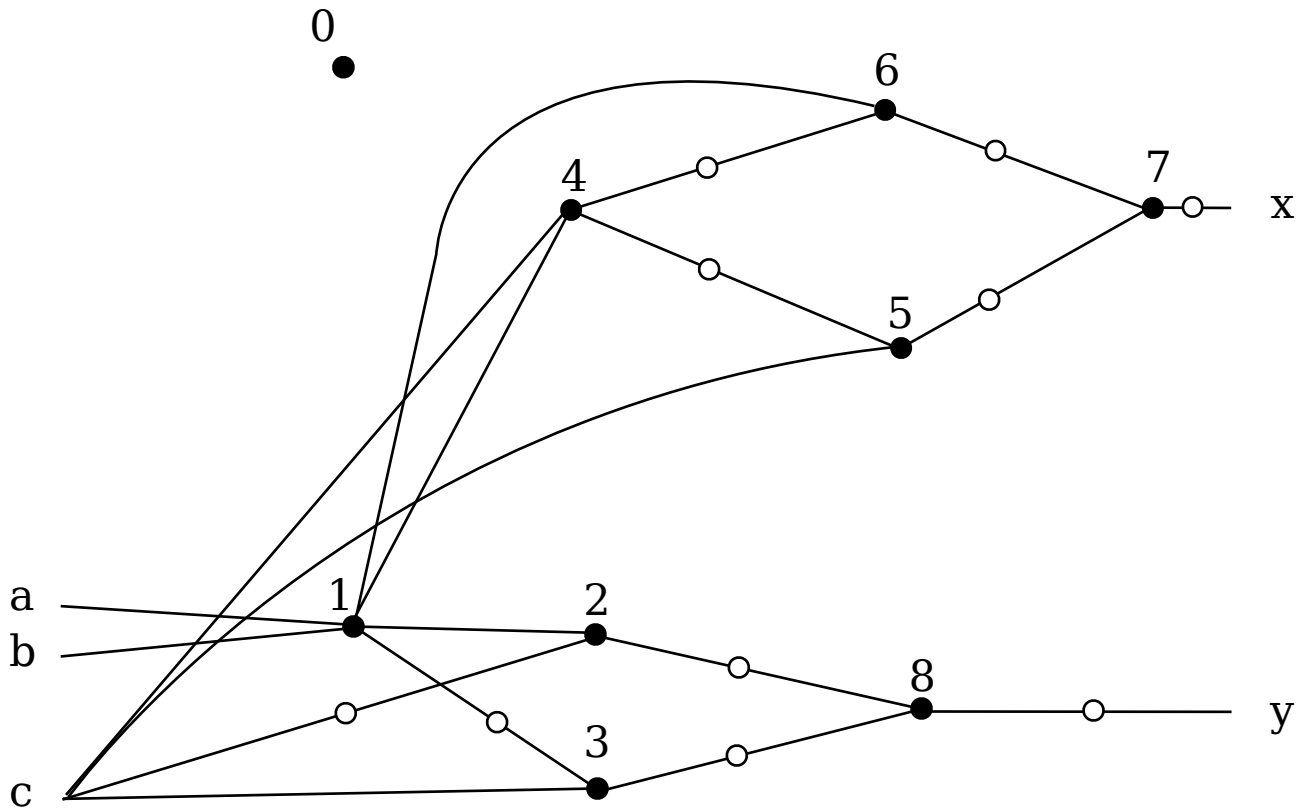
---

2 ?= 6

---

3 ?= 5

SAT Solver Says 3 = 5



Buckets

---

7 ?= 8

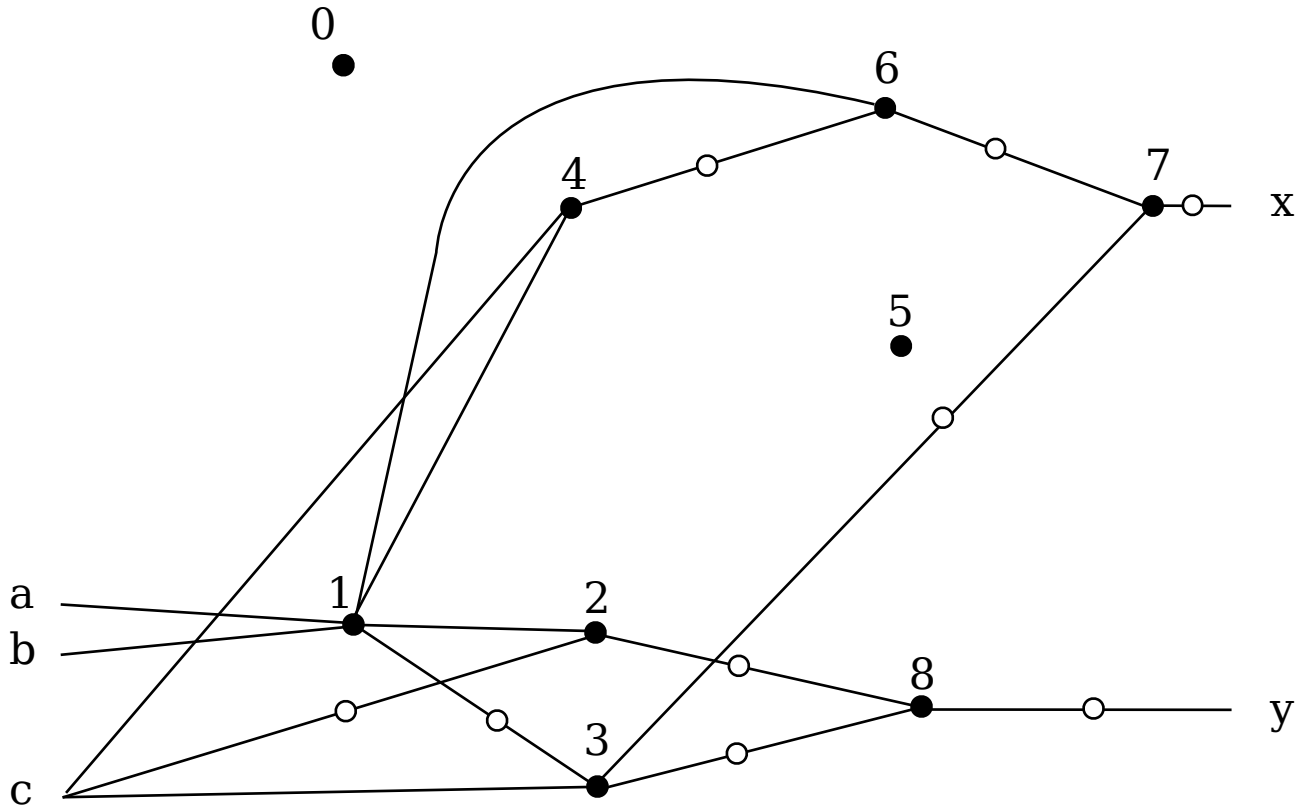
---

2 ?= 6

---

3 = 5

Merge 3 and 5



Buckets

---

7 ?= 8

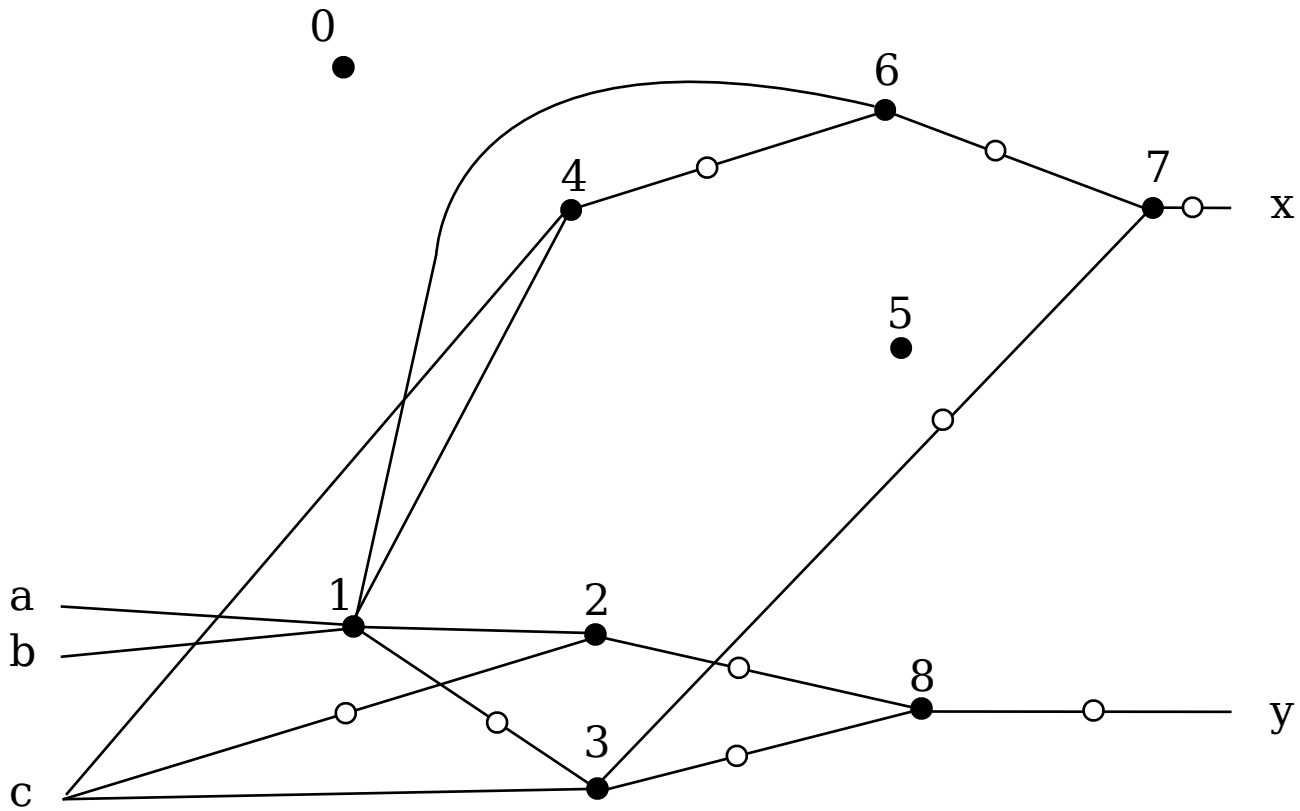
---

2 ?= 6

---

3 = 5

SAT Solver Says 2 = 6



Buckets

---

7 ?= 8

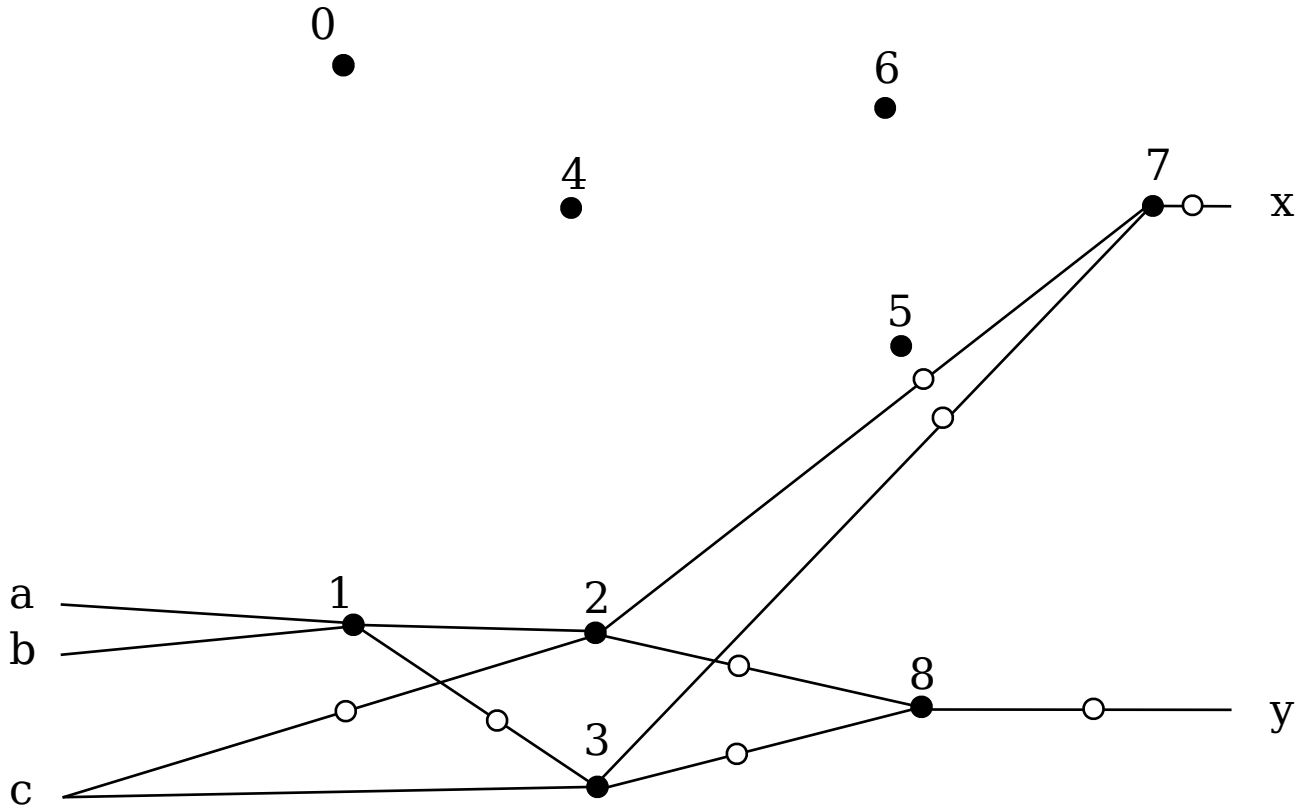
---

2 = 6

---

3 = 5

Merge 6 and 2



Buckets

---

7 ?= 8

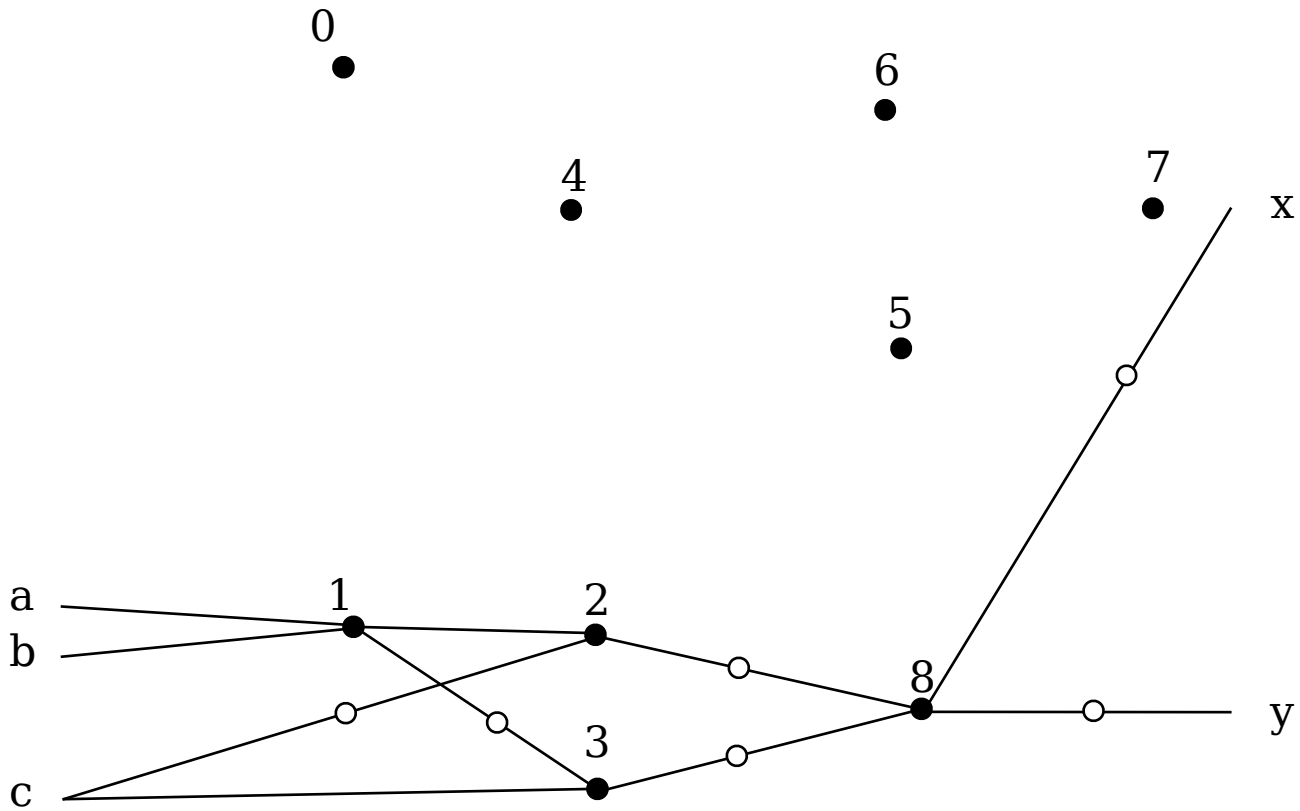
---

2 = 6

---

3 = 5

7 Structurally Hashes to 8



Buckets

---

7 = 8

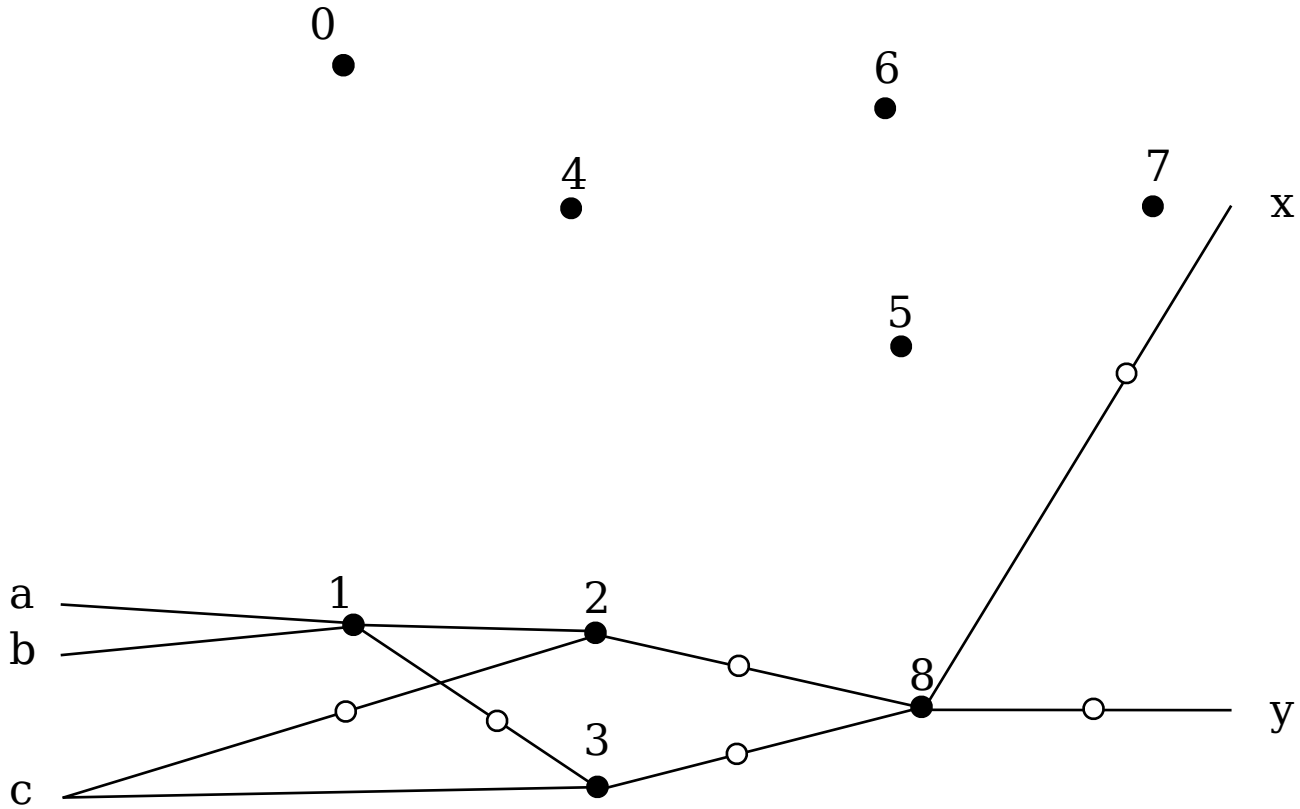
---

2 = 6

---

3 = 5

x and y Verified Equivalent



Buckets

---

7 = 8

---

2 = 6

---

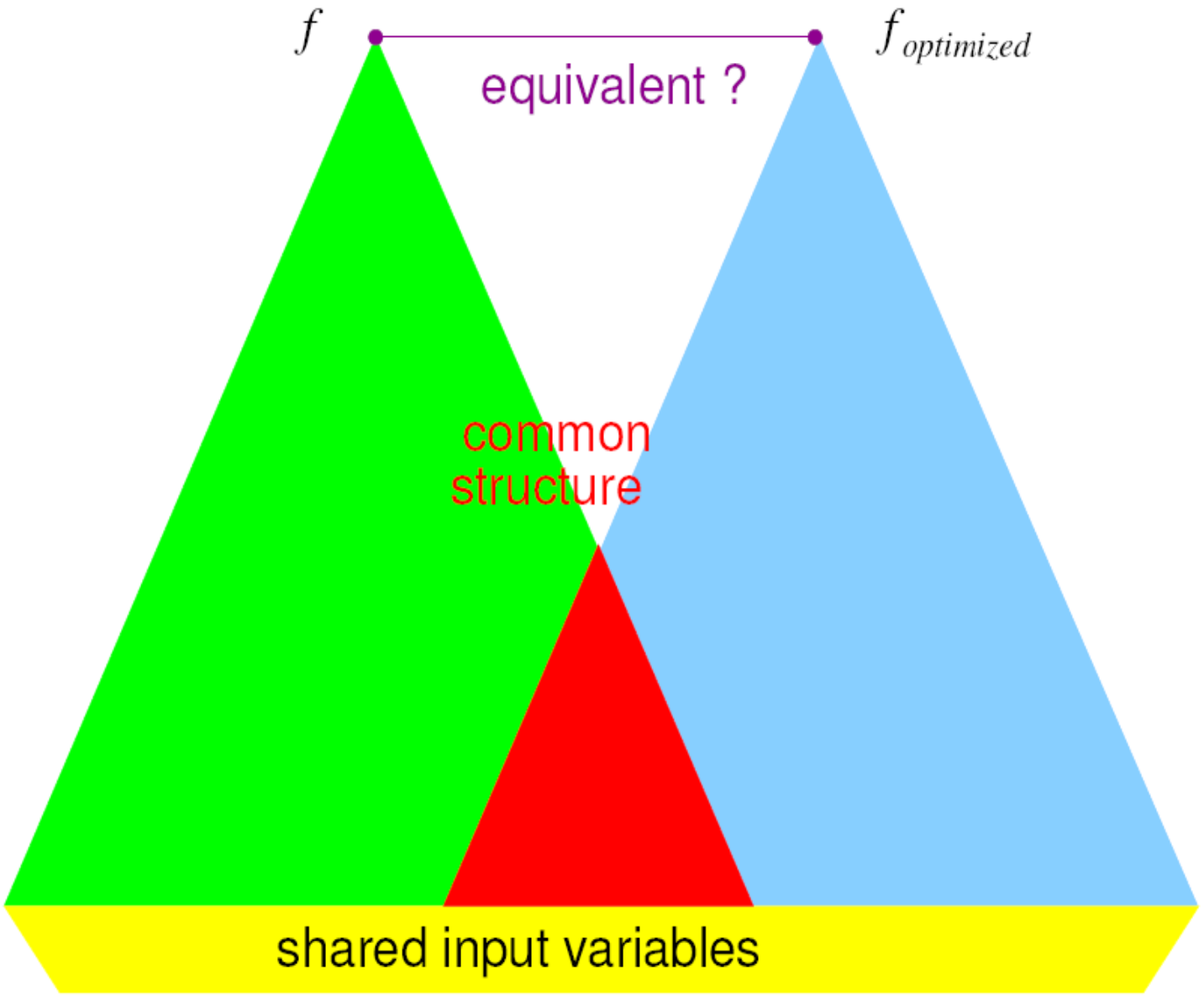
3 = 5

Example: Outline

- Random Simulation -
  - Send random vectors through the two circuits, collecting pairs of candidate equivalent nodes
- And/Inverter Graph
  - Find more equivalent nodes by creating the AIG of the circuits
- SAT Sweeping
  - Use candidate equivalent nodes to guide SAT searches, merging AIG nodes which reduces the complexity of future SAT searches

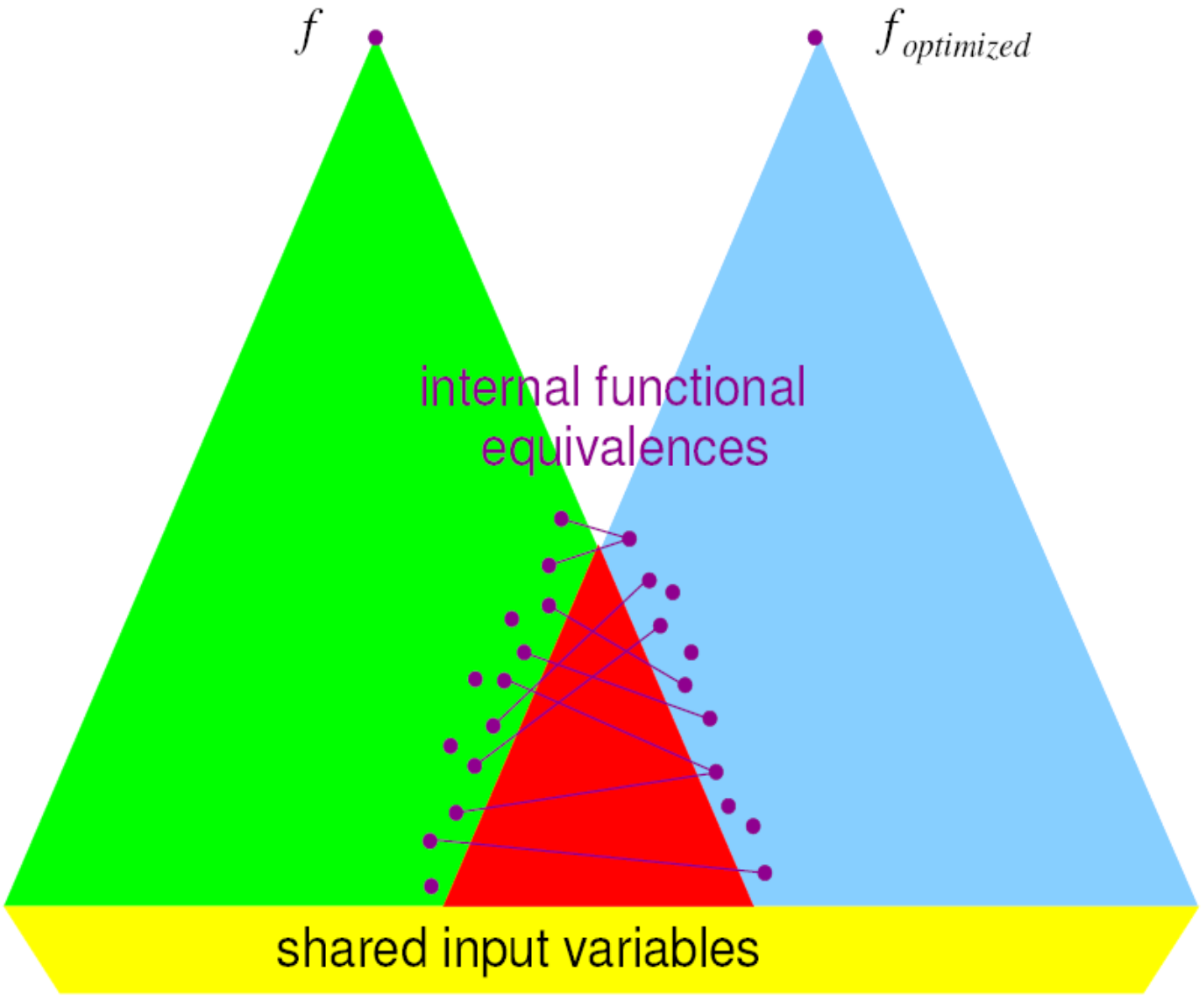
Slides taken from A. Biere. SAT in Formal Hardware Verification.

Equivalence Checking in the Large



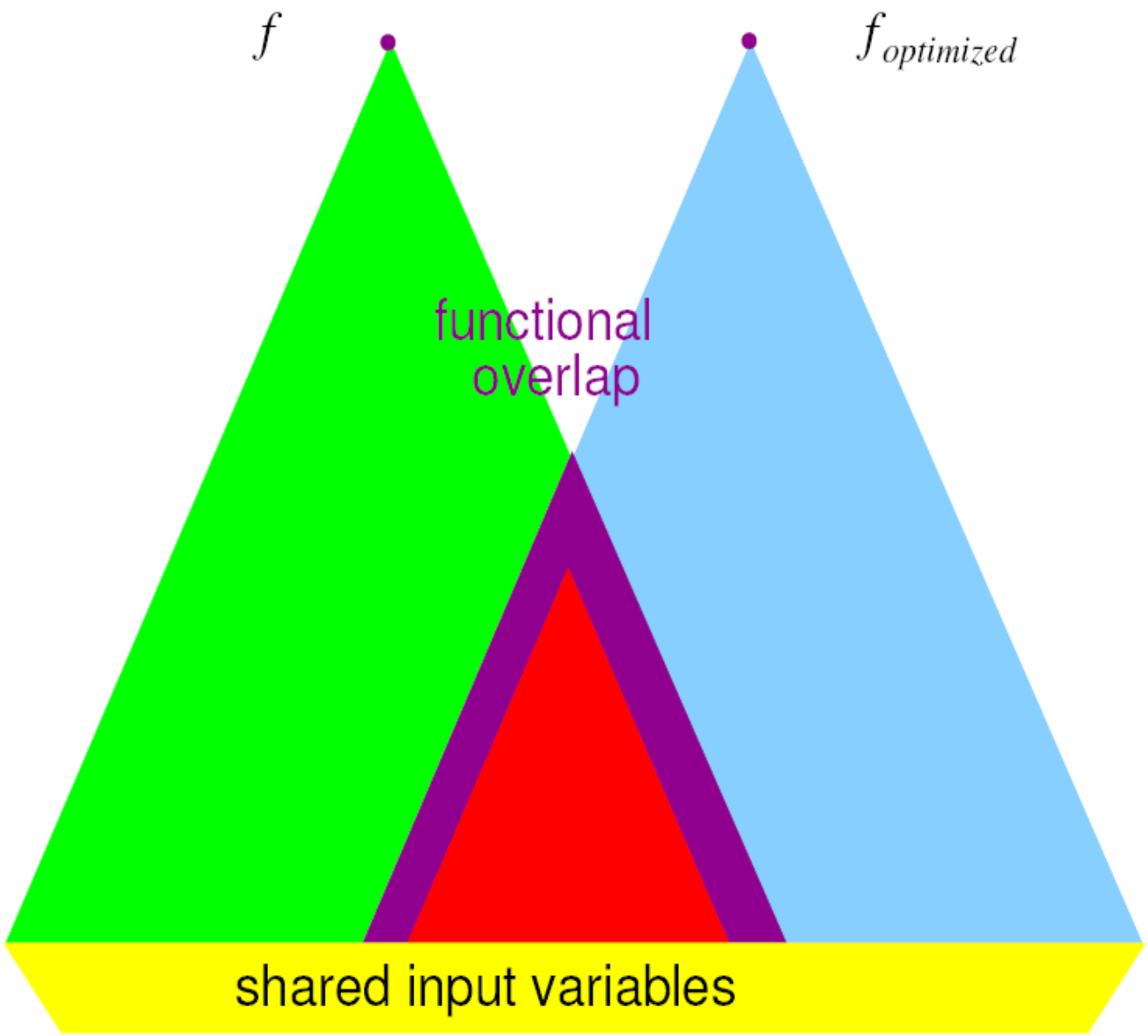
Slides taken from A. Biere. SAT in Formal Hardware Verification.

Equivalence Checking in the Large



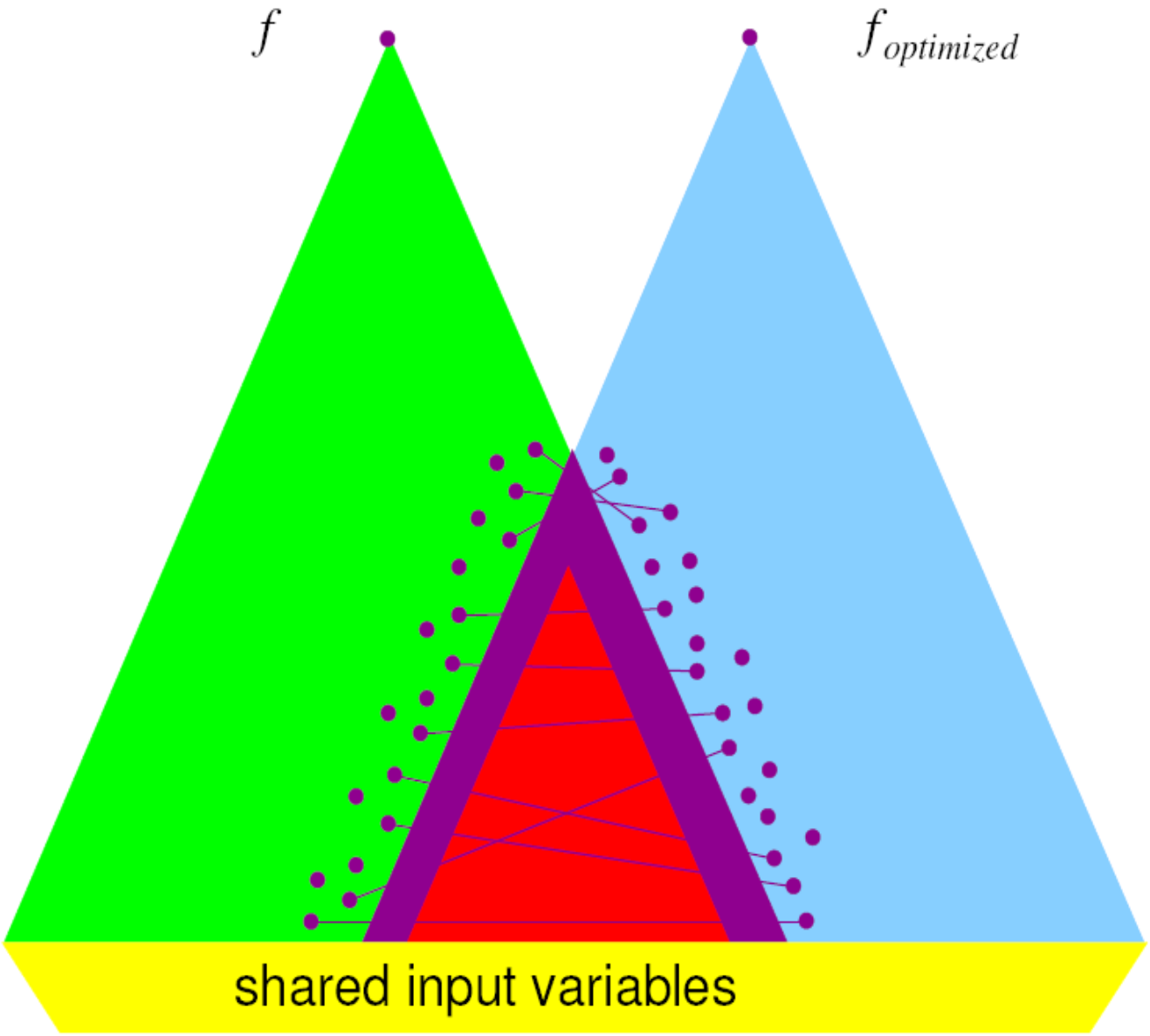
Slides taken from A. Biere. SAT in Formal Hardware Verification.

Equivalence Checking in the Large



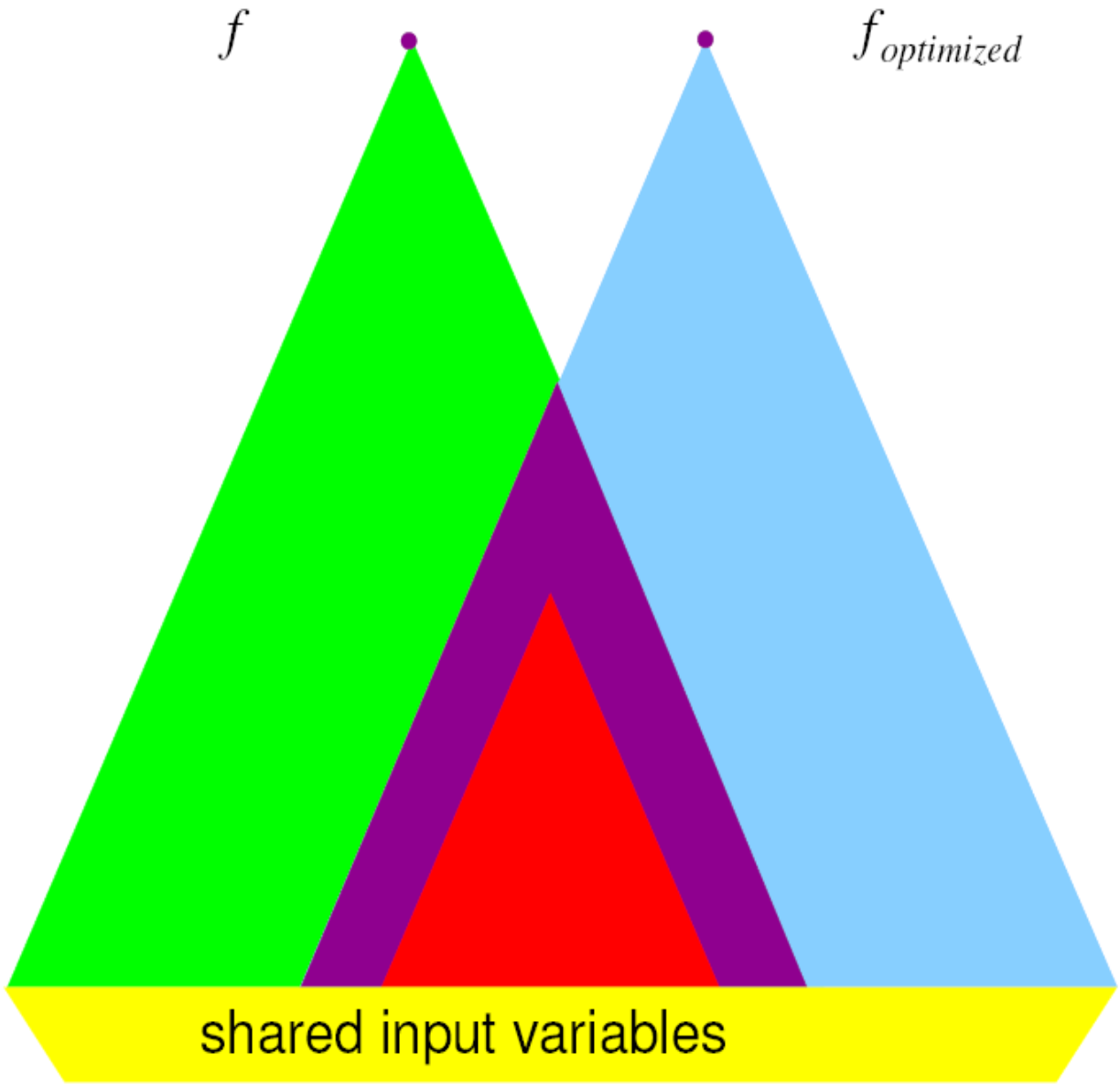
Slides taken from A. Biere. SAT in Formal Hardware Verification.

Equivalence Checking in the Large



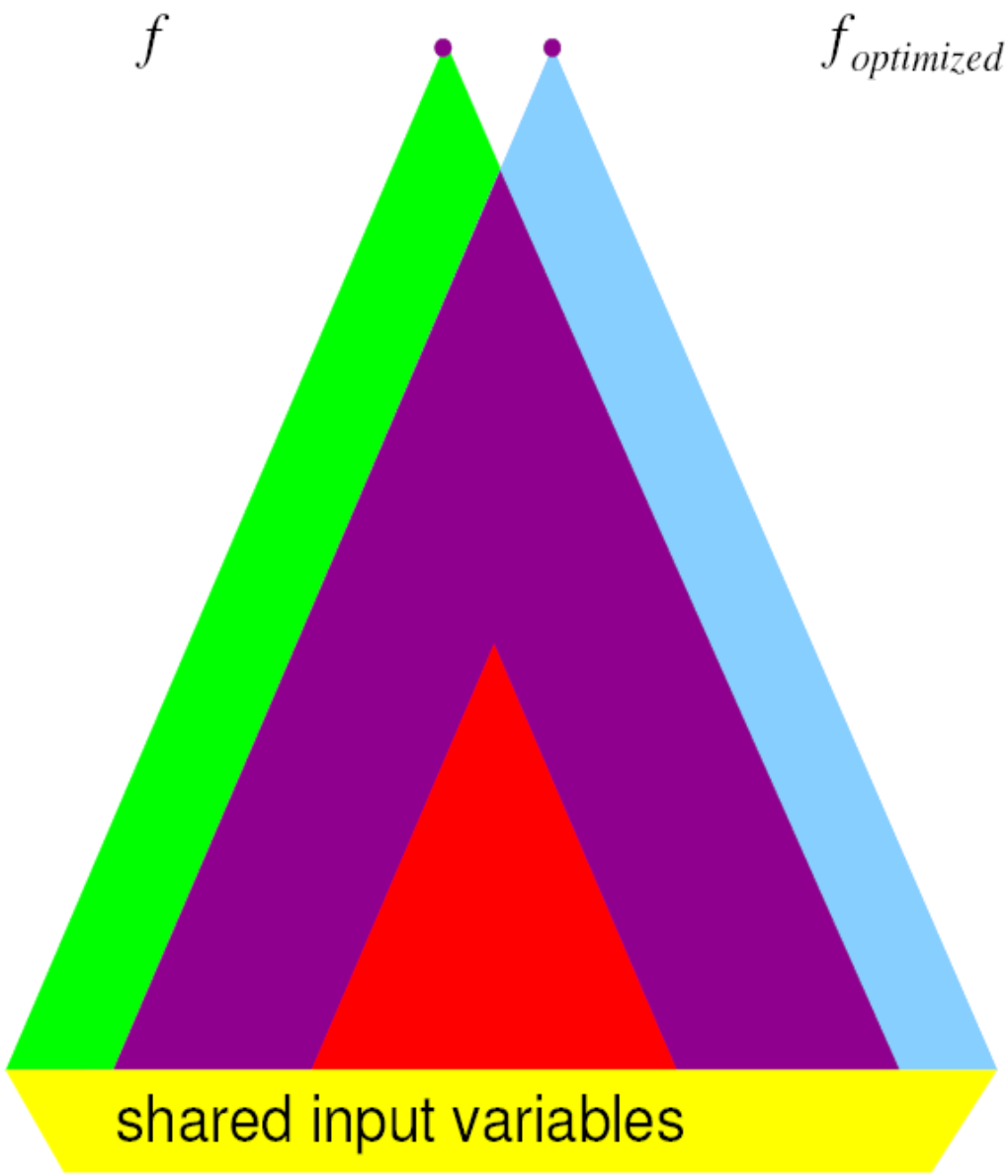
Slides taken from A. Biere. SAT in Formal Hardware Verification.

Equivalence Checking in the Large



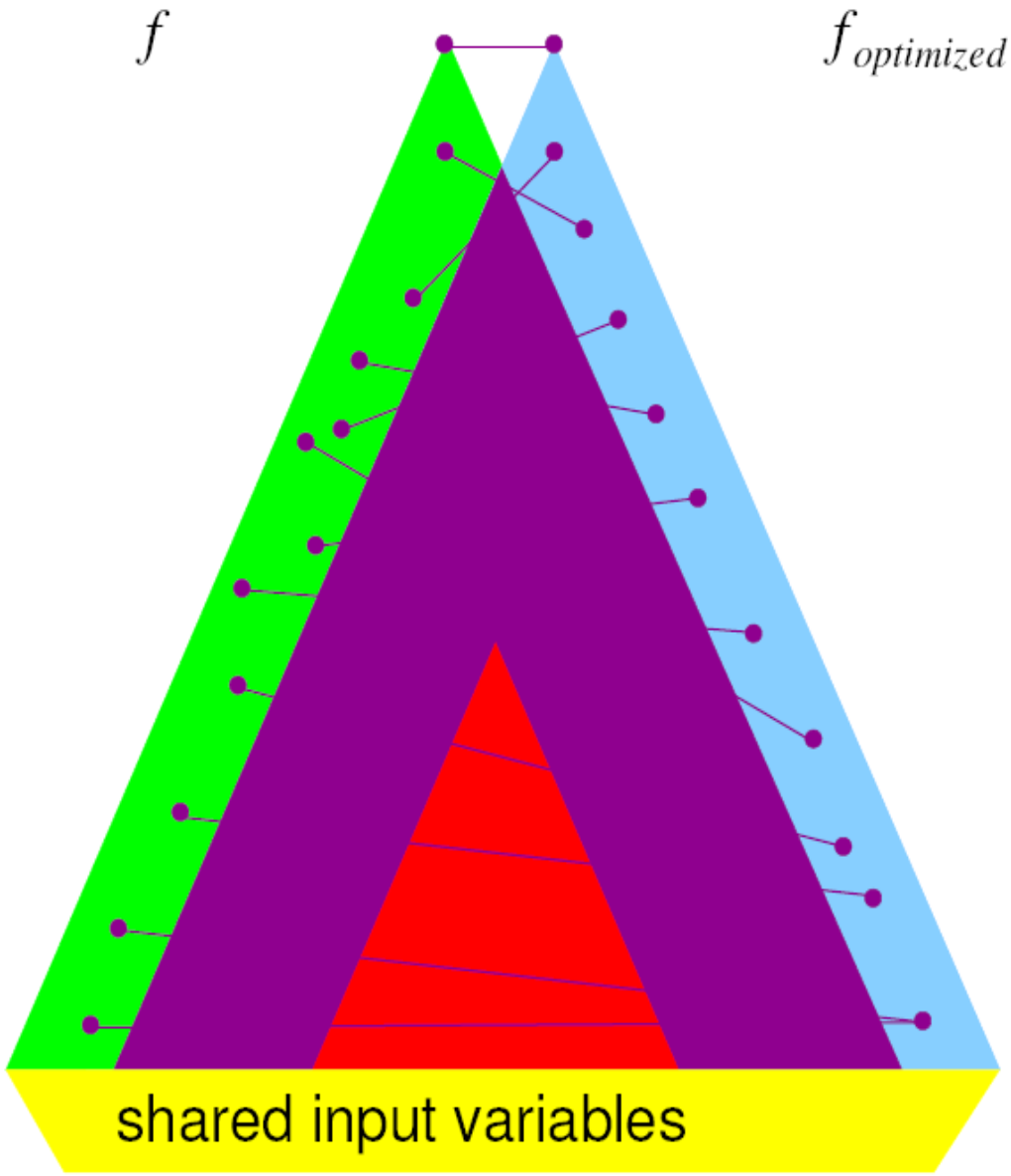
Slides taken from A. Biere. SAT in Formal Hardware Verification.

Equivalence Checking in the Large



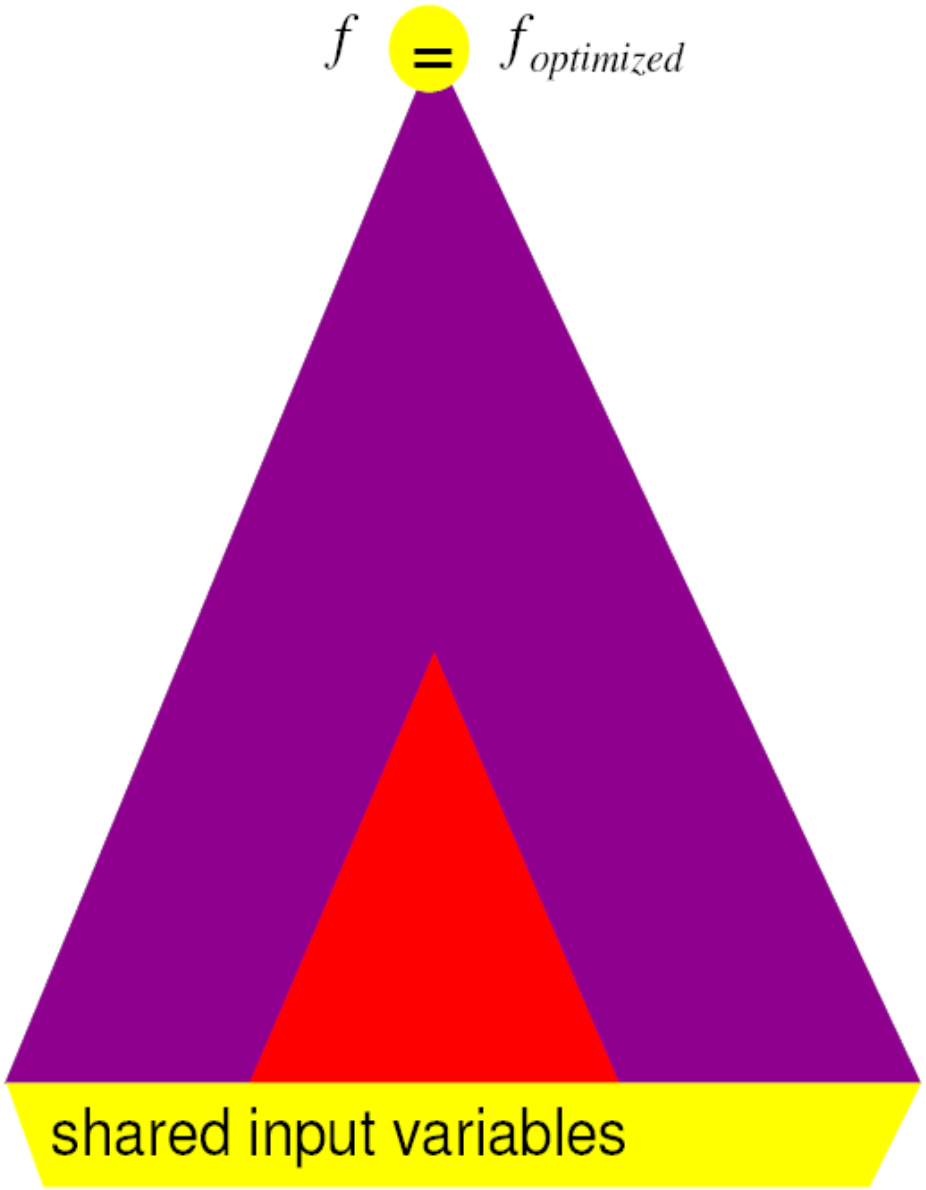
Slides taken from A. Biere. SAT in Formal Hardware Verification.

Equivalence Checking in the Large



Equivalence Checking in the Large

Slides taken from A. Biere. SAT in Formal Hardware Verification.



Cryptol

- Cryptol is a Haskell based specification language for writing crypto-algorithms
- Created by Galois Connections Inc. with support from NSA cryptographers
- Cryptol specifications can be transformed into AIGs
  - Cryptol also has a built in equivalence checker (jaig)
- Cryptol specifications can be used to verify various implementations
  - C code, VHDL, etc.

Results - AES

- Verified Cryptol specification of full rank AES-128 functionally equivalent to NIST competition optimized C-code
  - Cryptol-AES AIG has 934,000 nodes
  - NIST-AES AIG has 1,482,000 nodes
  - 190,000 equivalent nodes found
  - Using techniques described here plus special SAT heuristics
    - < 1 minute on 2 GHz Pentium III

Results - VdW

- Van der Waerden numbers W(k,r)=n
  - Place numbers 1 ... n into k buckets so that no arithmetic progression of length r exists in any bucket
  - Assertion by Dr. Michal Kouril
    - W(2,6) = 1132
  - This is quite a feat because now only 6 numbers are known and no new ones had been found since 1979

Results VdW

- Dr. Kouril's solver is written in VHDL, runs on a cluster of FPGAs at UC
- The solver has exhausted the search space
- How to give confidence that VHDL code is correct?
- Use equivalence checking!

Results VdW

- Wrote Cryptol specifications for the three main VHDL functions used
- Used Xilinx tools and Cryptol to generate AIGs from the VHDL code
- Used the Cryptol equivalence checker (jaig) to verify the VHDL code
  - Each function has 2^{240} possible inputs
  - Total time for all three checks < 30 minutes

References

A. Biere. Invited talk - SAT in Formal Hardware Verification. 8th Intl. Conf; on Theory and Applications of Satisfiability Testing (SAT'05), St. Andrews, Scotland, (2005).

D. Brand. Verification of Large Synthesized Designs. Proc. Intl Conf. Computer-Aided Design pp. 534-537 (1993).

E. Goldberg, Y. Novikov. How good can a resolution based SAT-solver be? SAT-2003, LNCS 2919, pp. 35-52.

F. Krohm, A. Kuehlmann, and A. Mets. The Use of Random Simulation in Formal Verification. Proc. of Int'l Conf. on Computer Design, Oct (1996).

A. Kuehlmann, F. Krohm. Equivalence Checking Using Cuts and Heaps. In Design Automation Conference (1997).

A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai. Robust Boolean Reasoning for Equivalence Checking and Function Property Verification. IEEE Trans. CAD, Vol. 21, No. 12, pp. 1377-1394 (2002).

A. Kuehlmann. Dynamic Transition Relation Simplification for Bounded Property Checking. In ICCAD (2004).

J. Lewis. Cryptol, A Domain Specific Language for Cryptography. http://www.cryptol.net/docs/CryptolPaper.pdf (2002).