Introduction to Formal Verification and Logic Synthesis

Yukio Miyasaka

Contents

Formal Verification

- Prove that a given logic circuit meets some given properties
- Often compared to random simulation, here are some tradeoffs:

	Formal Verification	Random Simulation
Coverage	100%	#test patterns / #all possible patterns
Time	~O(exp(circuit size))	O(#test patterns * circuit size)

Note: coverage has a different meaning for bounded model checking

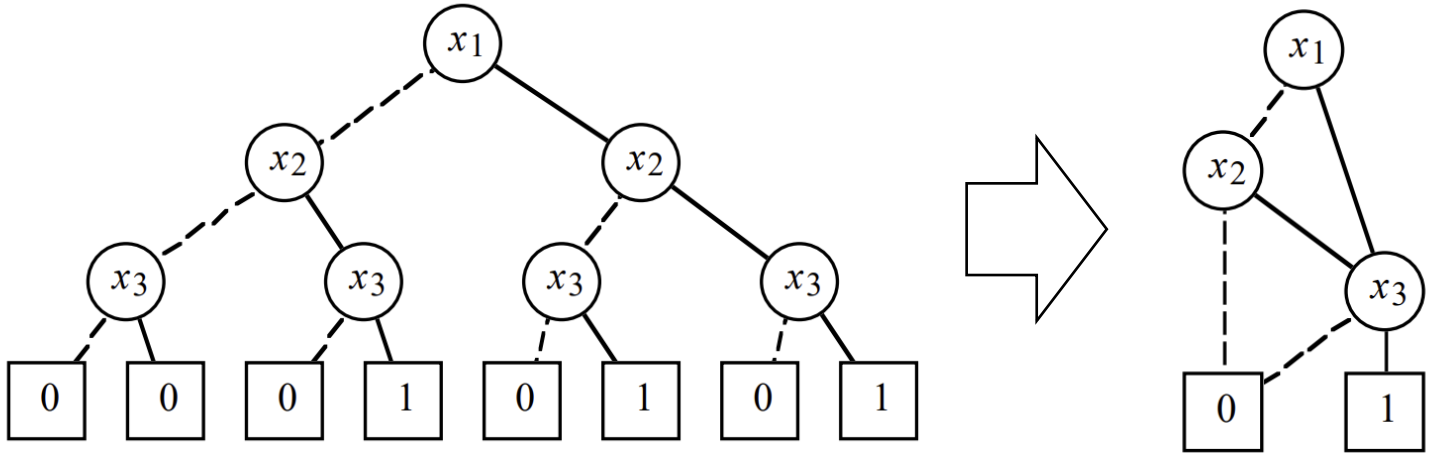
Equivalence Checking

- Given two implementations, prove they are functionally equivalent
  - For simplicity, assume the implementations are combinational logic circuits
  - Optimized implementation v.s. Golden model (Spec)
- Exhaustive simulation is enough to prove this property
- However, it takes an exponential amount of time
  - N inputs -> 2^N patterns to simulate
  - O(exp(#inputs))
- Can we do it better?

Binary Decision Diagram (BDD)

- Binary tree
- Redundant nodes (equivalent cofactors) are removed
- Each node has a unique function

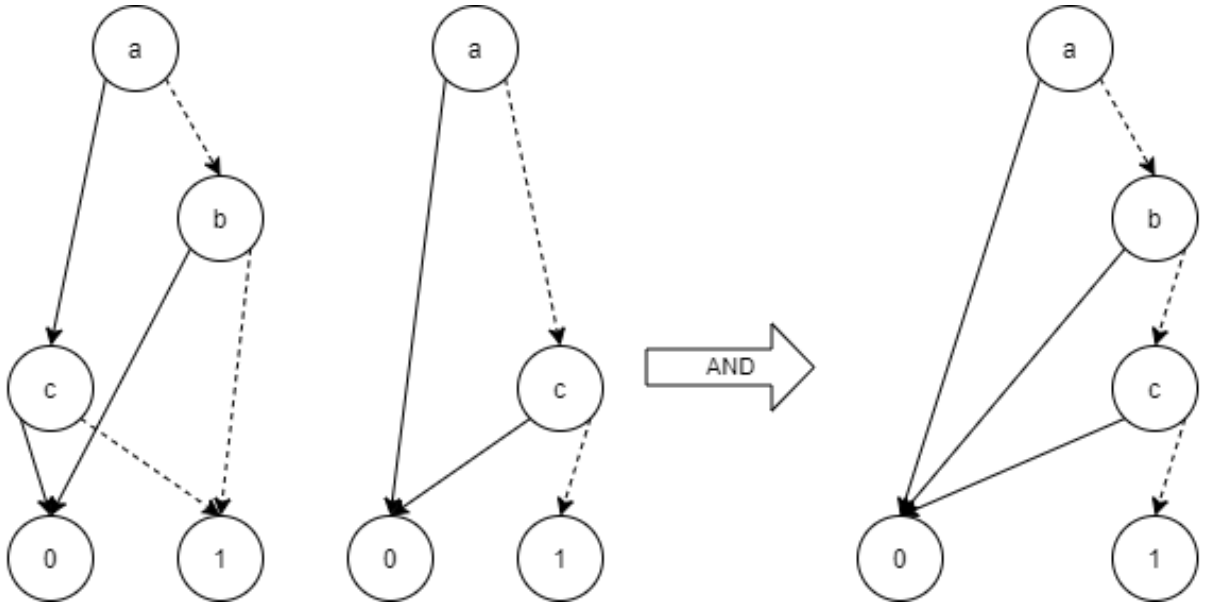
x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



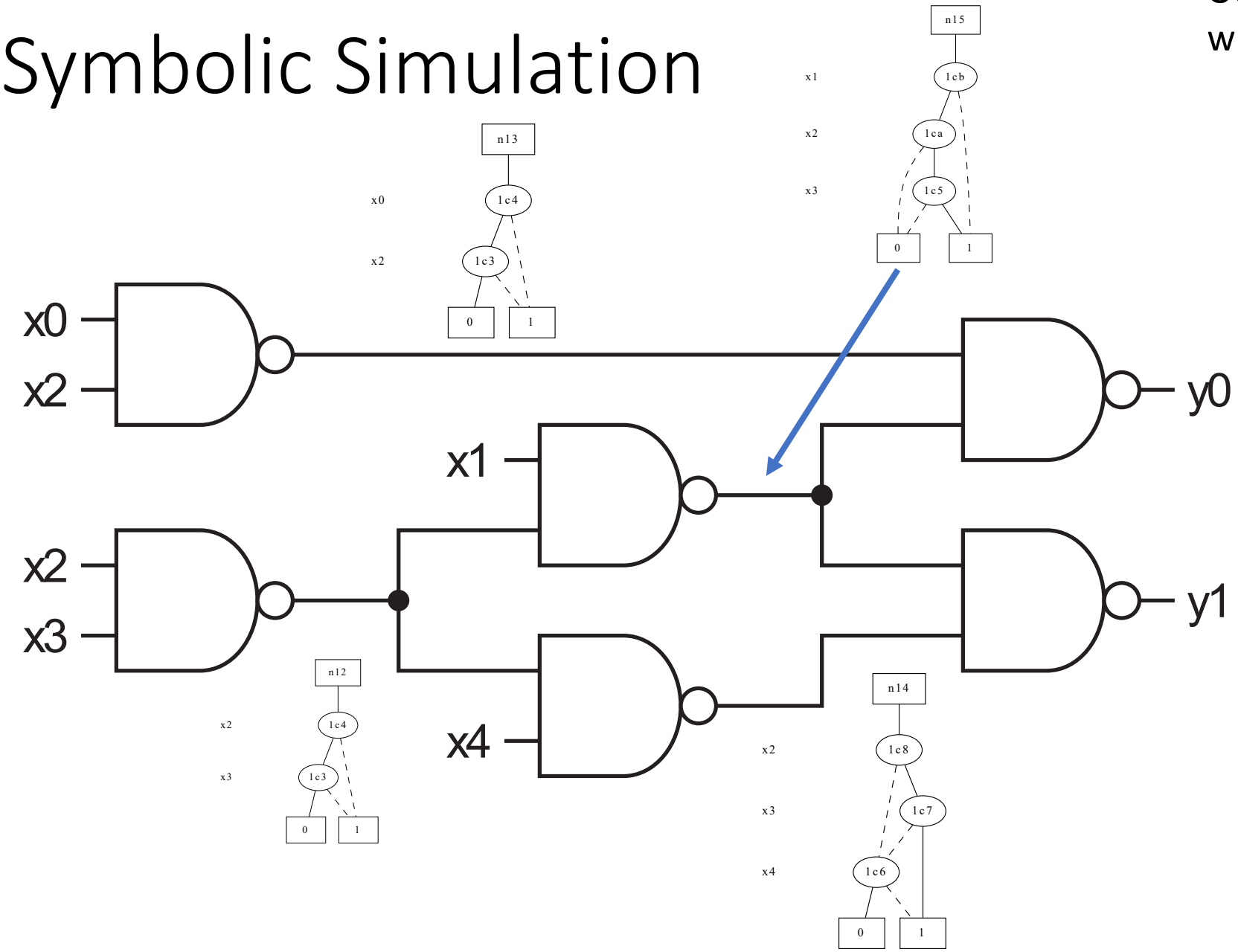
Randal E. Bryant, “Symbolic Boolean manipulation with ordered binary-decision diagrams,” in ACM Computing Surveys, vol. 24, no. 3, pp. 293–318, 1992.

BDD Operations

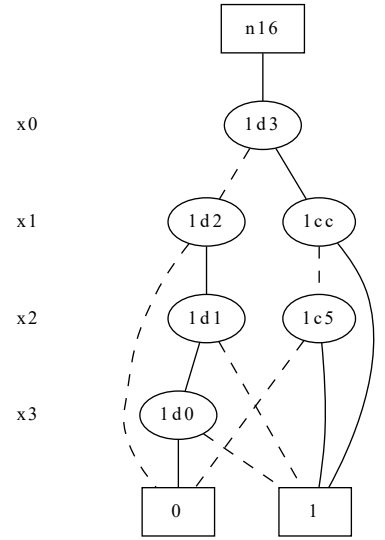
- We can construct a BDD for the result of operation by a recursive procedure
  - e.g. BDD z = AND(BDD x, BDD y);
- Runtime is bounded by O(#BDD nodes) while #BDD nodes is usually smaller than exp(#inputs)

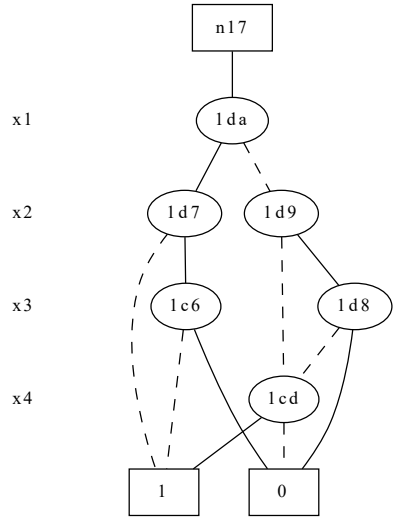


Symbolic Simulation



Output BDDs to be compared with the golden model

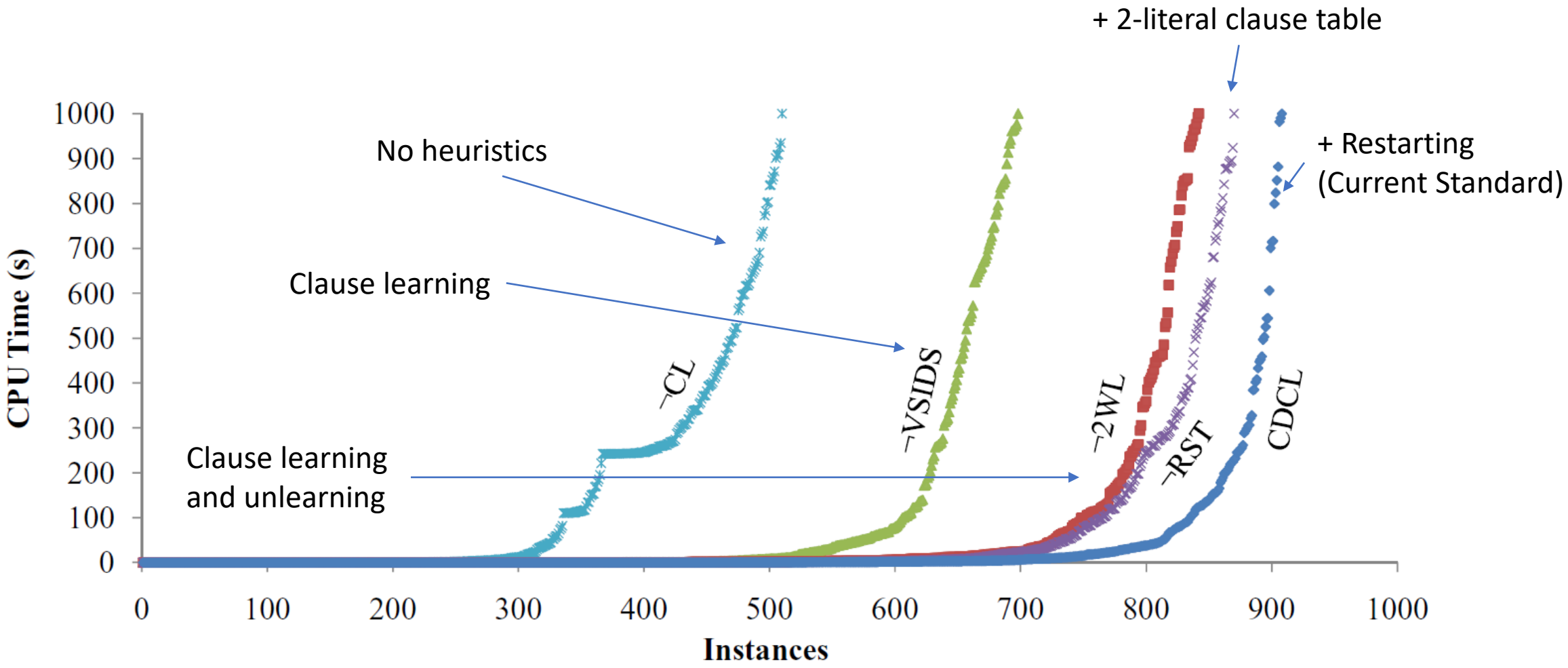




Boolean Satisfiability Problem

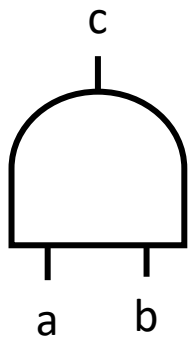
- Problem definition $\exists \vec{x}. f(\vec{x}) = 1.$
- One of the most famous NP-complete problems
- $f(\vec{x})$ is usually given as CNF (Conjunctive Normal Form) a.k.a. POS
  - Variable $x_1, x_2, \ldots$
  - Literal $x_1, x_1', x_2, x_2', \ldots$
  - Clause $x_1 + x_2' + x_3, \ldots$
  - CNF $(Cluase\ A) \cdot (Clause\ B) \cdot \ldots$
- Many heuristics have been proposed
- Can be solved much faster than the other NP-complete problems

History of SAT Solver Improvement

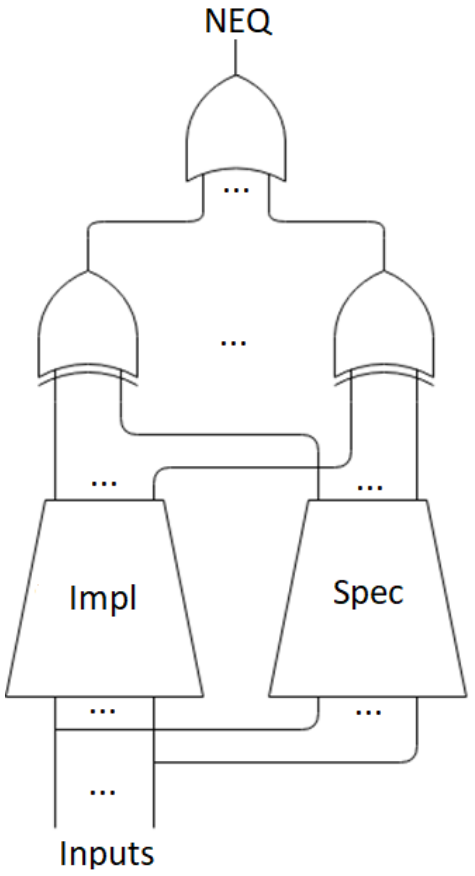


H. Katebi, K. A. Sakallah, and J. P. Marques-Silva, ``Empirical study of the anatomy of modern sat solvers," in Proc. 14th SAT, 2011, pp.343-356.

SAT-based Equivalence Checking



$$f_{AND} = (a' + b' + c)(a + c')(b + c')$$



MITER Circuit

Solve

$$\exists \overrightarrow{Inputs}, \overrightarrow{Internal}.$$
$$f_{MITER}\left(\overrightarrow{Inputs}, \overrightarrow{Internal}, NEQ\right) \cdot NEQ.$$

If satisfiable, there exists a pattern where two circuits output different values.

If unsatisfiable, two circuits are equivalent.

Performance Comparison

- For i10 benchmark (257 inputs, 224 outputs, about 2000 gates)
  - Exhaustive simulation: Never ends (2^257 > 10^25 patterns)
  - Symbolic simulation: 0.65 sec
  - SAT-based: 0.43 sec
- SAT-based method is usually faster than symbolic simulation
- Exceptions are arithmetic circuits like multipliers

Logic Synthesis

Logic Synthesis

- Generates a logic circuit from various kinds of descriptions such as truth table, Boolean expression, etc.
- Important metrics: area (#gates) and depth (#levels)

Two-Level Logic

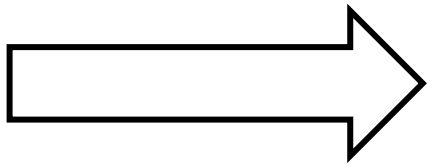
- AND gates in the first level, OR gates in the second level
- Direct representation of SOP
- The basic strategy is the same as Karnaugh map
- However, for more than one output, we have to care logic sharing
  - $x = a + a'bc, \quad y = b'c + a'bc$ is better than
  - $x = a + bc, \quad y = b'c + a'c$
- ESPRESSO heuristic logic minimizer:
  - Developed by Robert K. Brayton (emeritus professor at UC Berkeley)
  - https://ptolemy.berkeley.edu/projects/embedded/pubs/downloads/espresso/index.htm
  - Incompatible with modern C compilers, while you can find a patched version on GitHub or other websites

ESPRESSO: Example

test.pla

```
.i 3
.o 2
.p 8
000 00
001 01
010 00
011 11
100 10
101 11
110 10
111 10
.e
```

espresso test.pla > out.pla



out.pla

```
.i 3
.o 2
.p 3
-01 01
011 11
1-- 10
.e
```

Two-Level Logic to Multi-Level Logic

- We can reduce #gates by factoring
  - $y = b'c + a'c = (b' + a')c$
- **Fast Extract** is a greedy algorithm for factoring
  - For each pair of products, calculate divisors
    - $(abc, a'bc') \rightarrow divisors = \{ac + a'c', b\}$
  - For each divisor, count how many literals it can save
  - Factor out the divisor that can save the most
  - Repeat until no more factoring is possible
  - (Some details are omitted)

J. Rajski, J. Vasudevamurthy, “The test-preserving concurrent decomposition and factorization of Boolean expressions”, IEEE Trans. CAD, Vol.11 (6), June 1992, pp.778-793.

Fast Extract: Example

- 5-input majority function: $abc + abd + abe + acd + \cdots$
  - Factoring out a single variable can save 5 literals, while sum of two variables can save 10 literals
- Factor out $a + b$
  - $(a + b)(cd + ce + de) + abc + abd + abe + cde$
- Factor out ab
  - $(a + b)(cd + ce + de) + ab(c + d + e) + cde$
- Factor out $c + d$
  - $(a + b)(cd + (c + d)e) + ab\big((c + d) + e\big) + cde$
- After handling trivial cases where cd is shared, we get 12 gate implementation:
  - First level: $n_0 = a + b, n_1 = ab, n_2 = c + d, n_3 = cd$
  - Second level: $n_4 = n_2 e, n_5 = n_2 + e, n_6 = n_3 e$
  - Third level: $n_7 = n_3 + n_4, n_8 = n_1 n_5$
  - Forth level: $n_9 = n_0 n_7, n_{10} = n_8 + n_6$
  - Fifth level: $n_{11} = n_9 + n_{10}$

Multi-Level Logic Optimization

- Fast Extract is not optimal
  - Quality depends on the initial SOP
  - What if multiple divisors can save the same number of literals?
  - One of them might lead to a better result in the end
- Rewriting is one of the most popular optimization methods
  - Extract a subcircuit (4-5 inputs) iteratively
  - Replace it with an equivalent precomputed minimum circuit
  - How can we precompute the minimum circuit?

Exact Synthesis

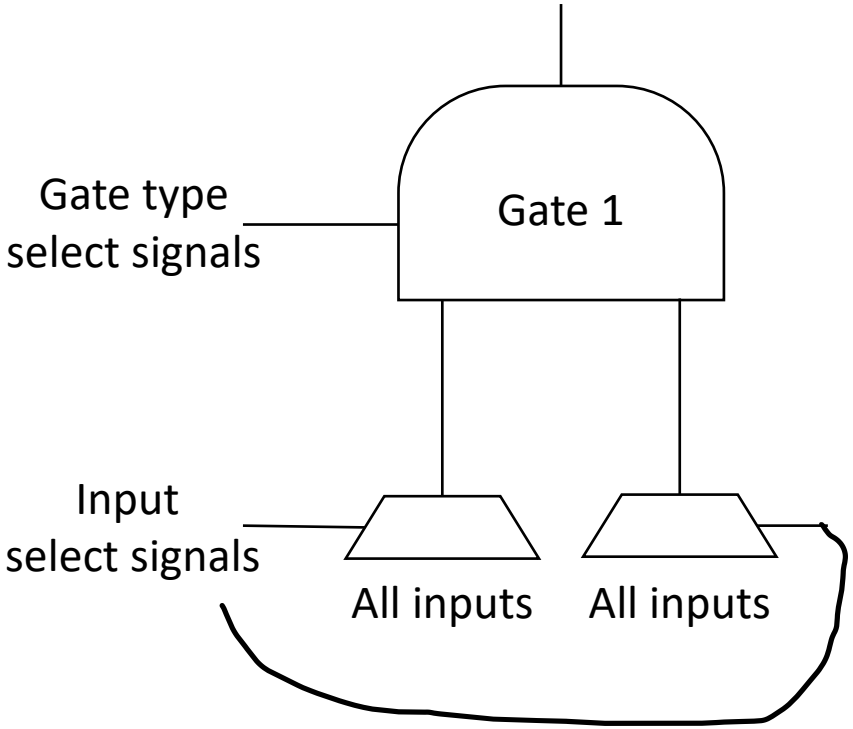
- Encode all possible circuits of *N* gates into a CNF
- Let a SAT solver find one that is equivalent to the specification

- If the solver cannot find one with *N-1* gates but one with *N* gates, the latter one is proven to be minimum

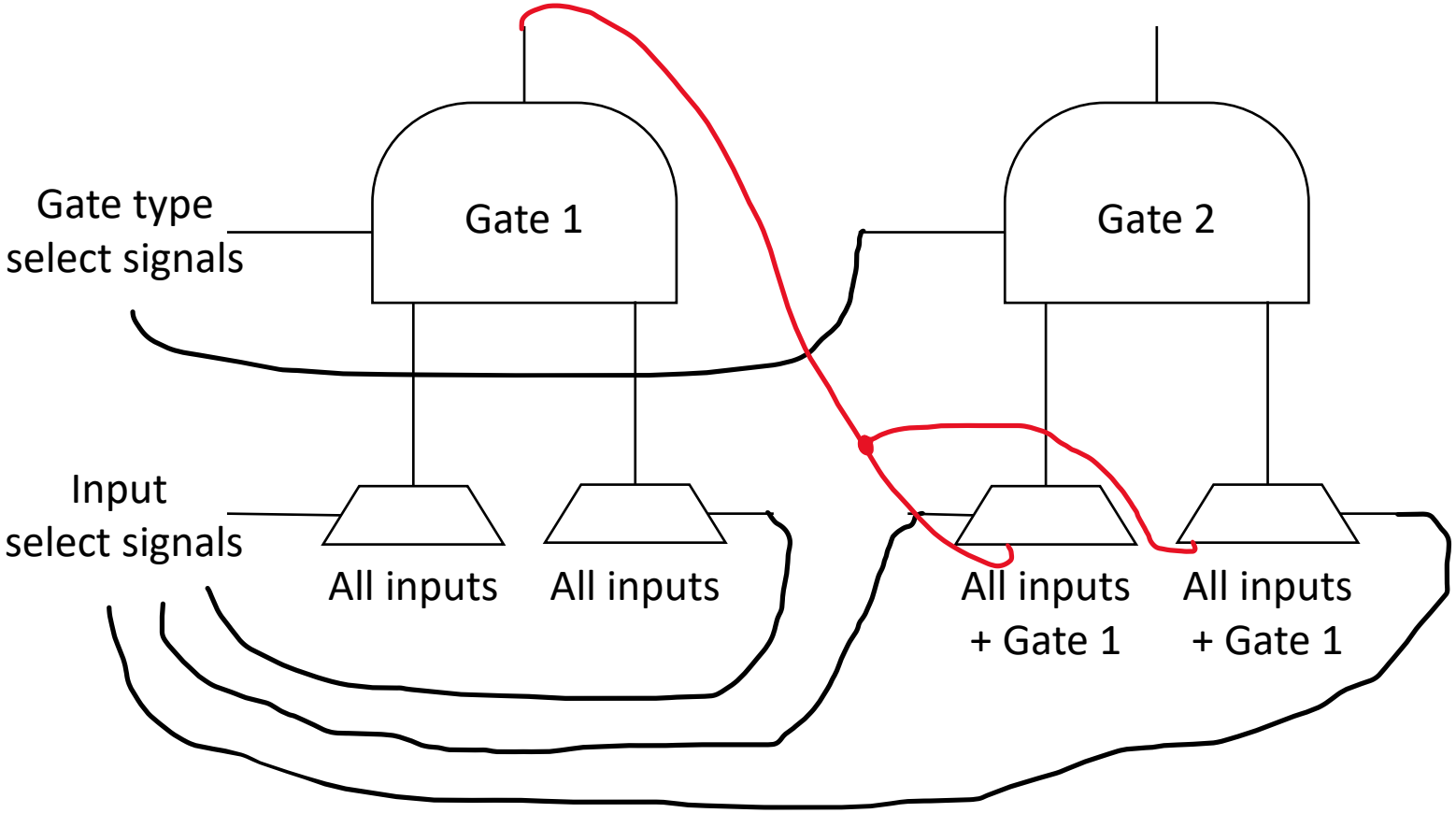
- There are many encoding methods

W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, "SAT-based exact synthesis: Encodings, topology families, and parallelism", IEEE Trans. CAD, Vol. 39(4), April 2020, pp. 871-884.

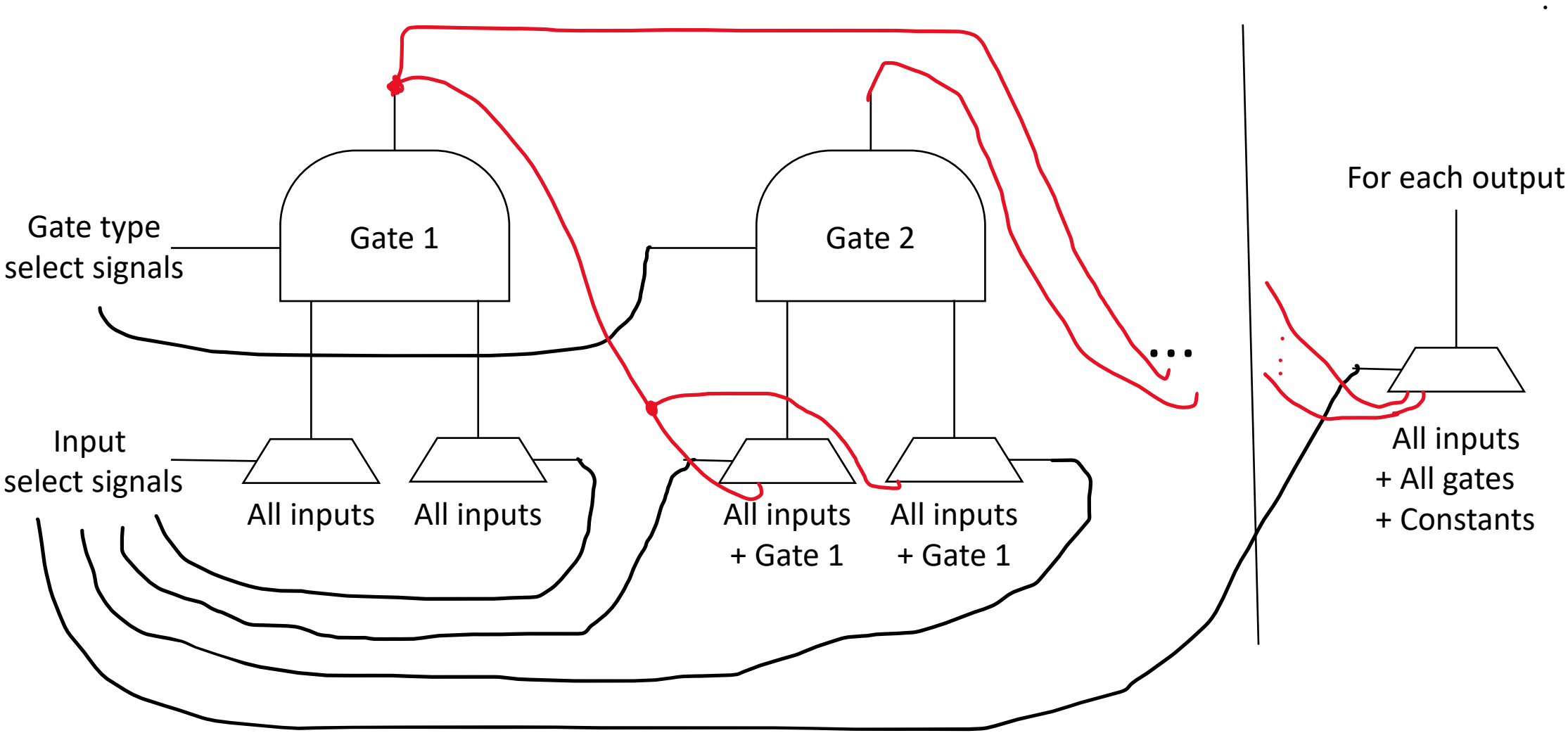
Exact Synthesis Encoding



Exact Synthesis Encoding



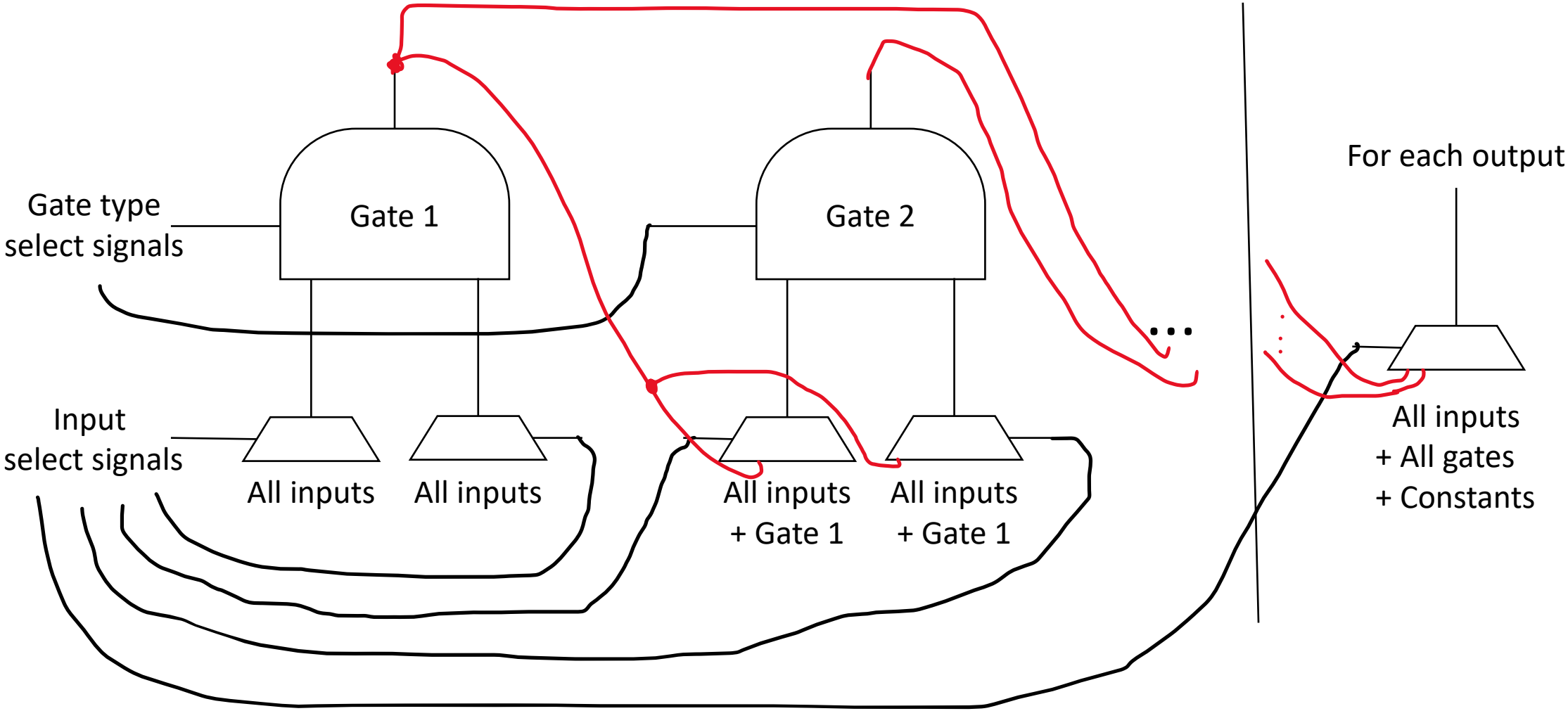
Exact Synthesis Encoding



Exact Synthesis Encoding

Solve

$$\exists \overrightarrow{Select\ signals}.\ \forall \overrightarrow{Inputs}.$$
$$f\left(\overrightarrow{Inputs}, \overrightarrow{Select\ signals}\right) = Spec\left(\overrightarrow{Inputs}\right).$$



More About Logic Optimization

- Exact synthesis works up to 5 gates.
- Rewriting is local optimization, does not necessarily lead to the global optimal
- There are many other optimization methods
  - Merge equivalent internal nodes
  - Substitute one node with a new subcircuit
  - Compute internal don’t-cares using BDD, and perform equivalent transformation
- Phase ordering problem
  - Once you apply one optimization, some other optimizations may be no longer effective
  - The effective order varies by circuits
  - Machine learning to find a good optimization order?