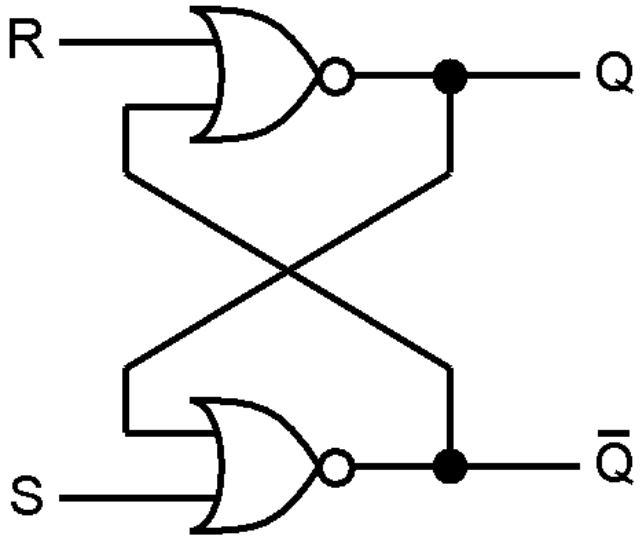
Fundamental Algorithms for System Modeling, Analysis, and Optimization

Edward A. Lee, Stavros Tripakis
UC Berkeley
EECS 144/244
Spring 2013

Non-Strict Synchronous Composition

Sequential Circuits

For sequential circuits, the outputs depend on previous inputs.



Sequential circuits have to have cycles.

Cyclic Combinational Circuits

Some circuits have cycles, but their behavior is combinational (output does not depend on previous inputs).

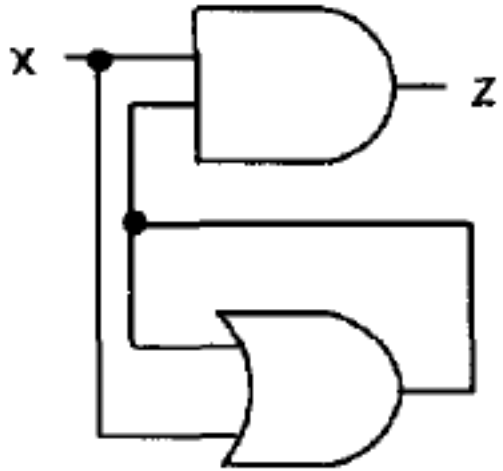


Fig. 1. Cyclic Combination Circuits: A Simple Example.

[Malik, Trans. on CAD, 1994]

Practical Cyclic Combinational Circuits

z = if (c) then shift(add(a, b), d) else add(shift(a, d), b)

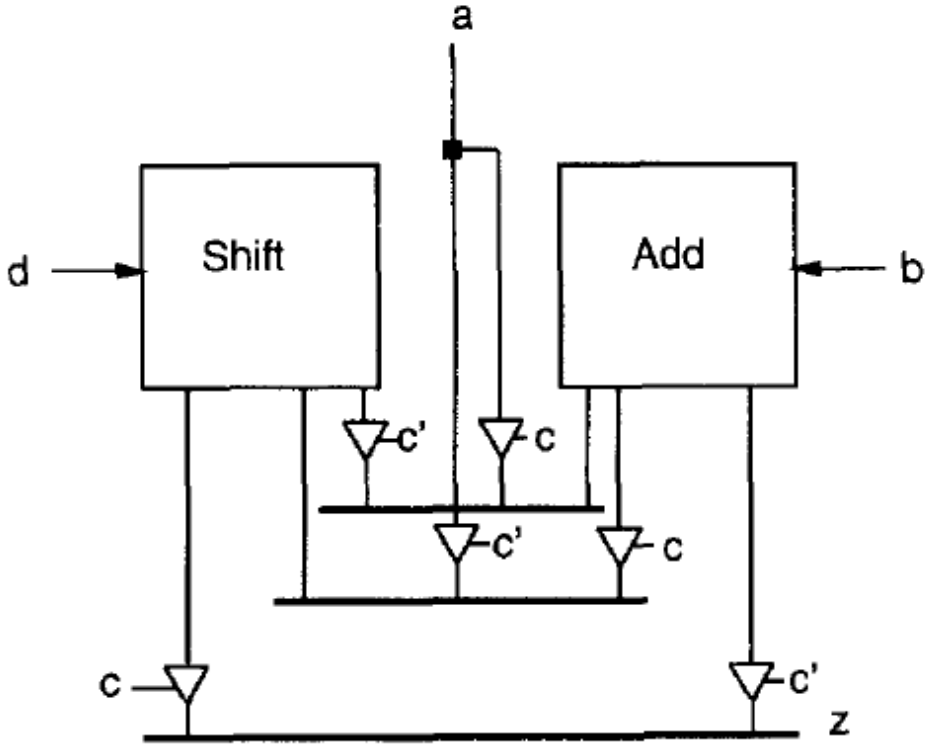


Fig. 3. Cyclic combinational circuits: practical example.

[Malik, Trans. on CAD, 1994]

Esterel:
A Synchronous/Reactive Programming Language

```
present I then
  present S then emit T end
else
  present T then emit S end
end
```

"There is a path from S to T and a path from T to S, hence a cycle. However, it is obvious from the source code that only one path can be used at a time, and, therefore, that the circuit is well-behaved."

[Shiple, Berry, and Touati, 1996]

Announcements

Class moving to 299 Cory starting Wed Feb 20

At request of department: want more time to set up video recording

Also: possible time change for Wed?

Homework 1: deadline extension

Now due Monday Feb 18 (holiday)

Cyclic Combinational Circuits with Sequential Parts

Notice that although the cycle does not affect the output, these circuits may be problematic:

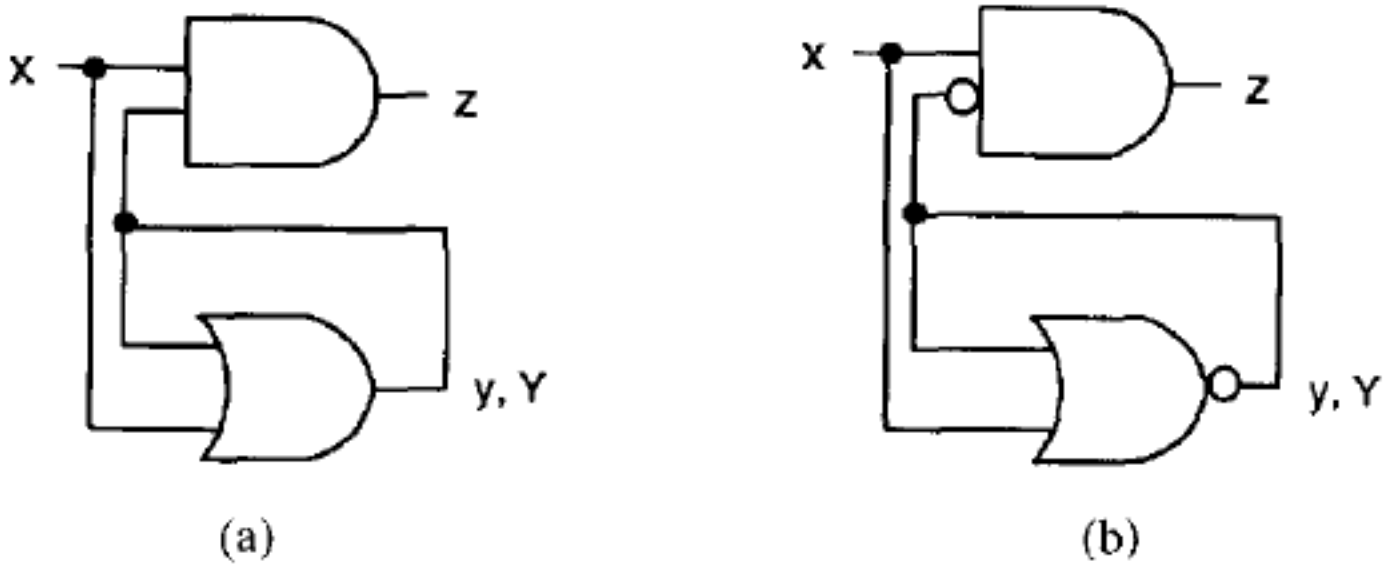


Fig. 4. Cyclic combinational circuits with sequential parts.

If x=0, circuit (b) may oscillate if there are gate delays.
[Malik, Trans. on CAD, 1994]

Problem: Ensuring well-behaved circuits (combinational and no oscillation)

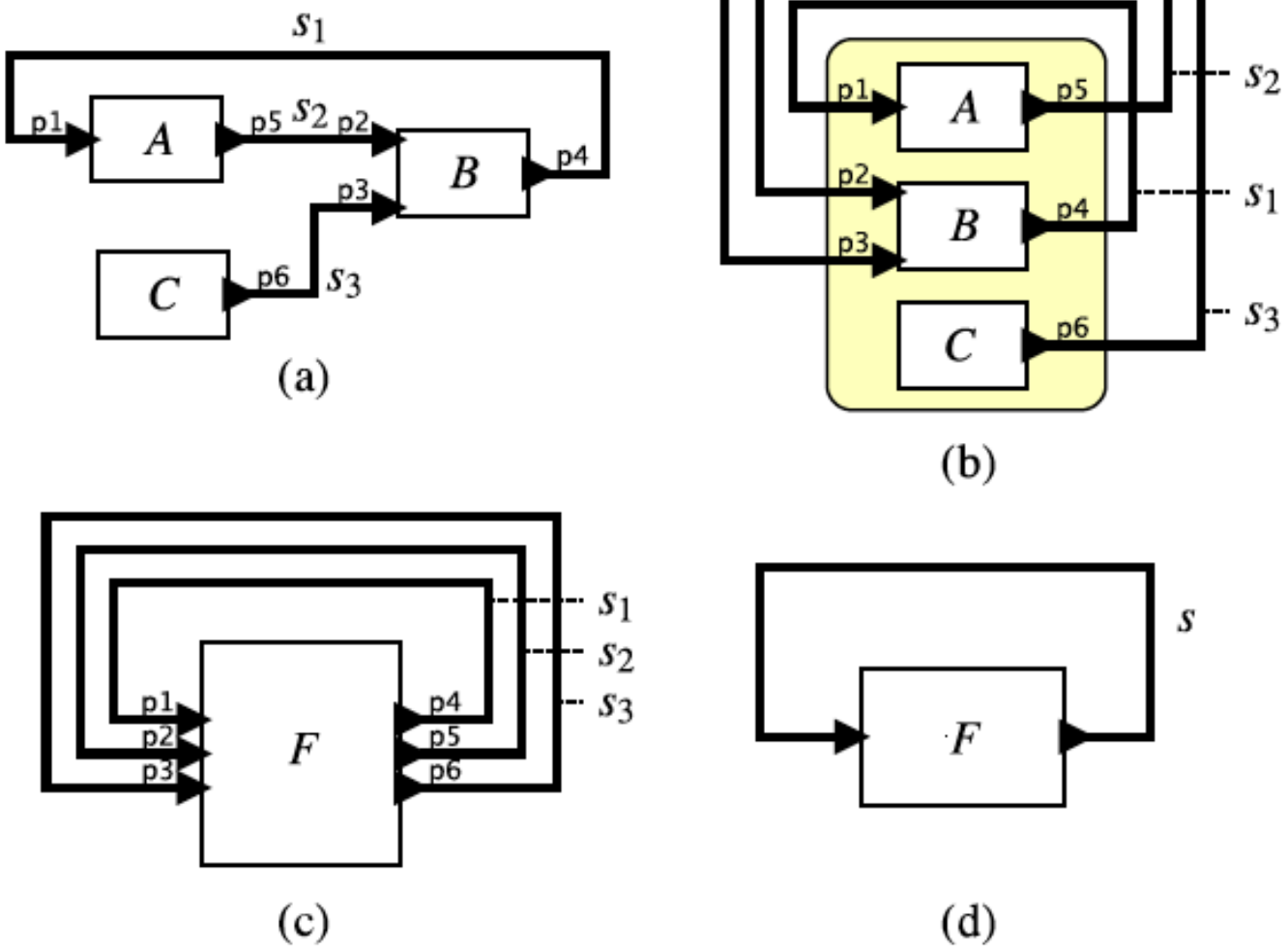
Solution: Constructive Fixed-Point Semantics

Two parts to this solution:

1. Fixed-point semantics
2. Constructive semantics

Fixed-Point Semantics

We will develop the solution first for a **closed** system (or equivalently, with known external inputs)



The Synchronous Abstraction

- Execution is a sequence of “ticks” of a global clock.
- All components in a network execute “simultaneously” and “instantaneously”
- The behavior at each tick is the solution to a fixed-point problem:

$$F_i(s) = s$$

Constructive Semantics

Solution procedure:

- Start with signals “unknown” at all nodes.
- Evaluate components (gates) in arbitrary order repeatedly until no further progress is made.
- If the result has all signals “known,” then declare it to be the constructive solution.
- Otherwise, reject model as non-constructive (buggy).

Applying the procedure

Initialize with input values and “unknown” on other nodes.

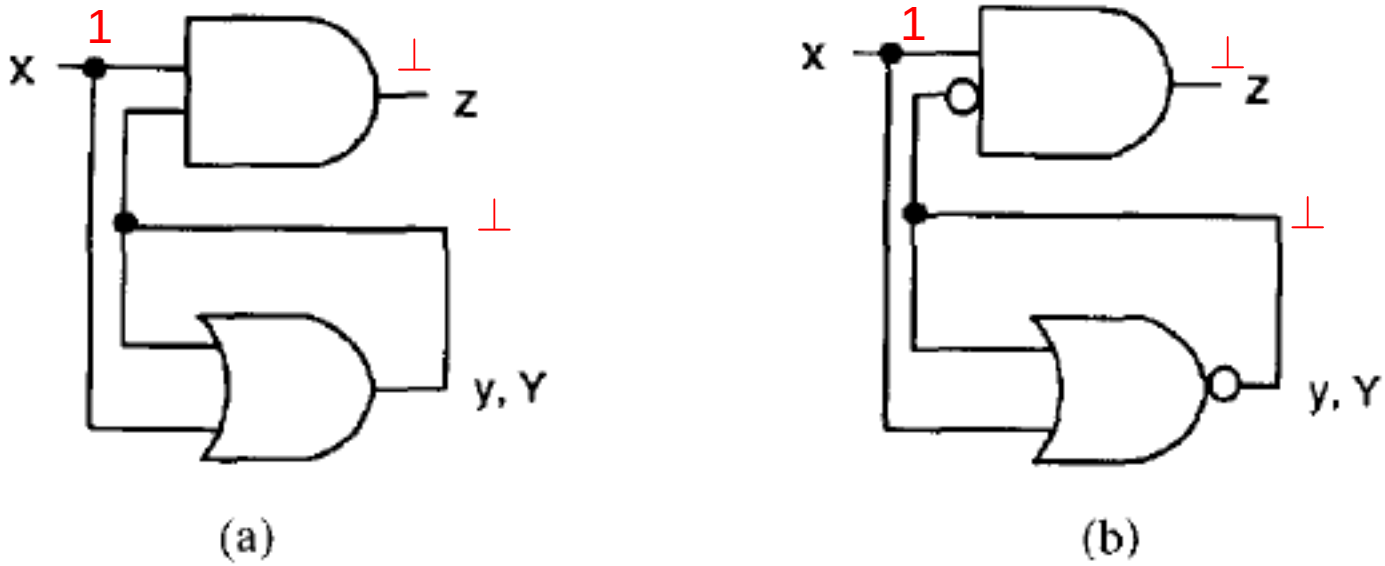


Fig. 4. Cyclic combinational circuits with sequential parts.

[Malik, Trans. on CAD, 1994]

Applying the procedure

Evaluate lower gates.

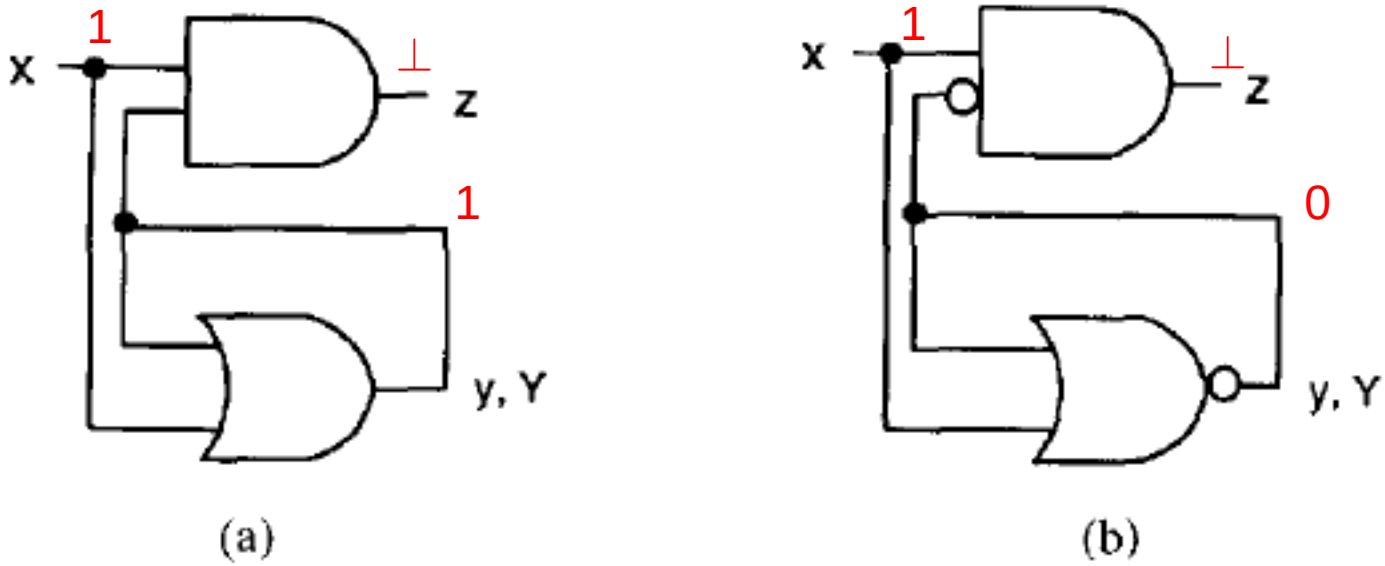


Fig. 4. Cyclic combinational circuits with sequential parts.

[Malik, Trans. on CAD, 1994]

Applying the procedure

Evaluate gates in arbitrary order until nothing changes.

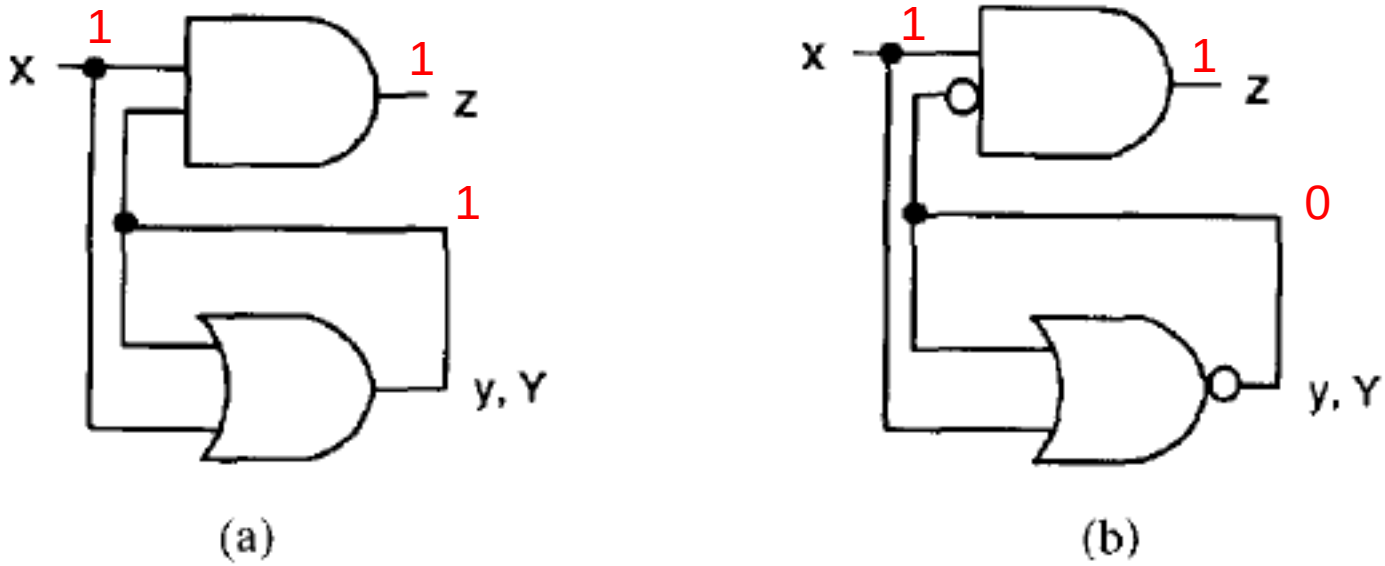


Fig. 4. Cyclic combinational circuits with sequential parts.

At this point, all nodes are known.

[Malik, Trans. on CAD, 1994]

Implicitly using Parallel (Non-Strict) Or

The non-strict or (often called the "parallel or") can produce a known output even if the input is not completely know. Here is a table showing the output as a function of two inputs:

input 2 \ input 1	⊥	F	T
⊥	⊥	⊥	T
F	⊥	F	T
T	T	T	T

Implicitly using Parallel (Non-Strict) And

The non-strict and (often called the "parallel and") can produce a known output even if the input is not completely know. Here is a table showing the output as a function of two inputs:

input 2 \ input 1	⊥	F	T
⊥	⊥	F	⊥
F	F	F	F
T	⊥	F	T

Applying the procedure with input 0

Initialize with input values and “unknown” on other nodes.

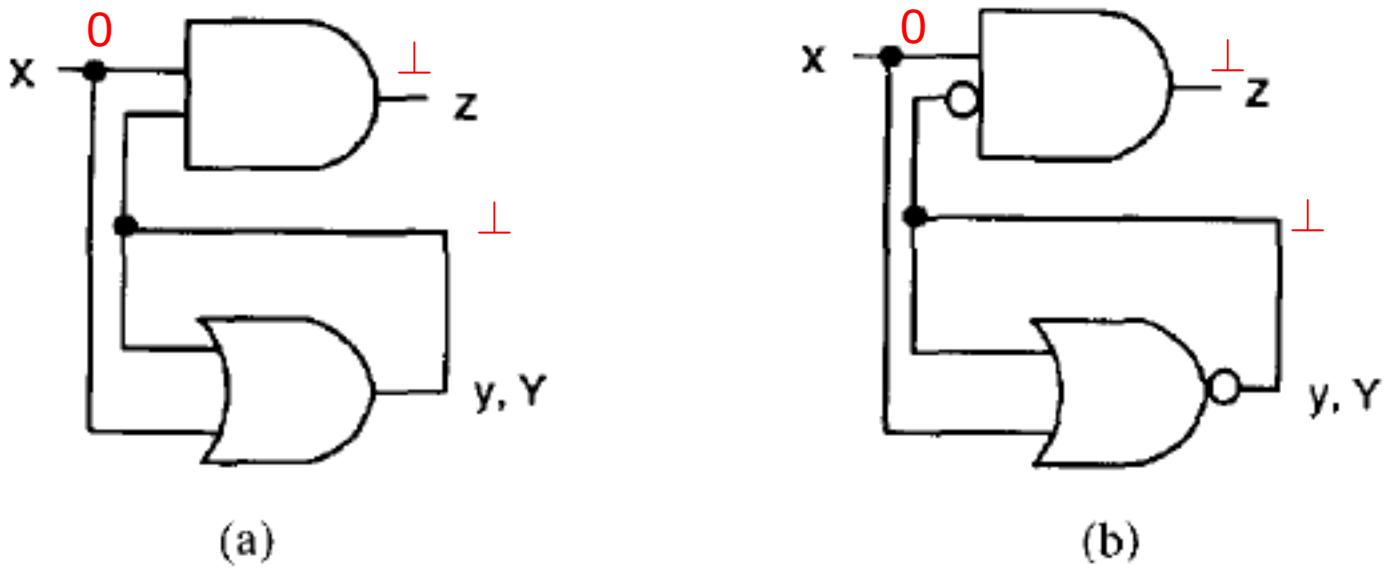


Fig. 4. Cyclic combinational circuits with sequential parts.

[Malik, Trans. on CAD, 1994]

Applying the procedure with input 0

Evaluate gates in arbitrary order until no progress is made.

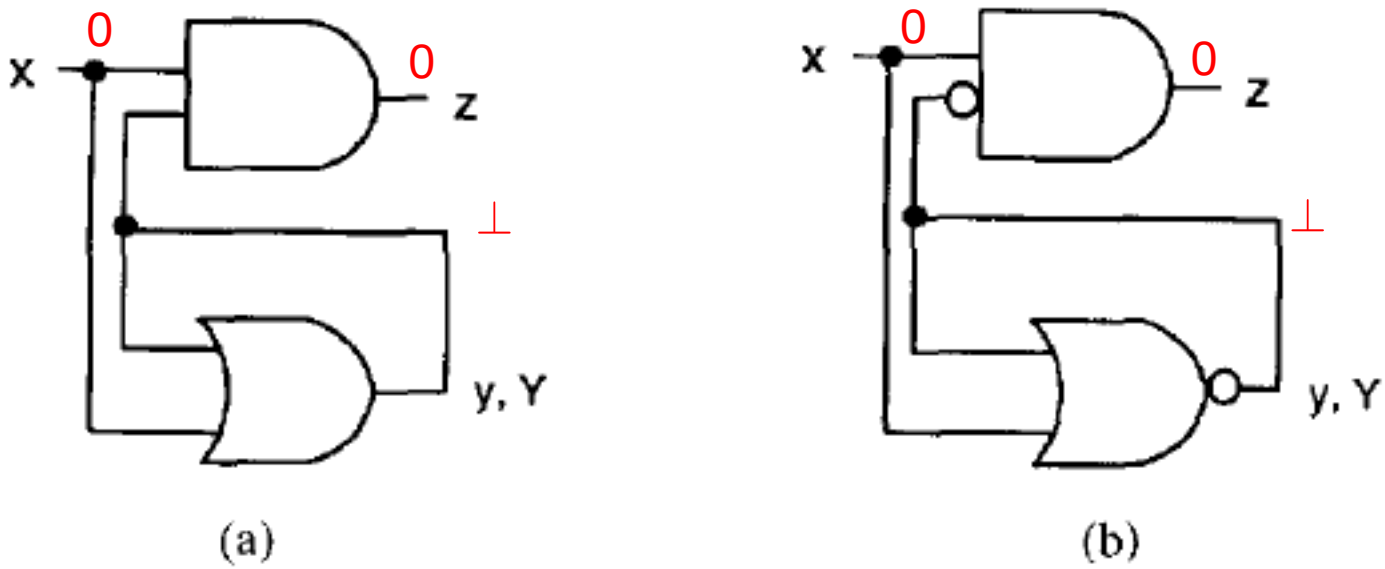


Fig. 4. Cyclic combinational circuits with sequential parts.

Unknown nodes remain. Constructive semantics rejects these circuits.

[Malik, Trans. on CAD, 1994]

Key Property

The procedure always converges to a unique solution for all nodes.

That solution is the least fixed point of a monotonic function on a complete partial order (CPO).

The Kleene fixed-point theorem assures that such a least fixed point exists, is unique, and can be found via this procedure.

Partial Orders

A *partial order* on the set A is a binary relation $\leq$ that is, for all $a, b, c \in A$,

- reflexive: $a \leq a$
- antisymmetric: $a \leq b$ and $b \leq a \Rightarrow a = b$
- transitive: $a \leq b$ and $b \leq c \Rightarrow a \leq c$

A *partially ordered set* (*poset*) is a set A and a binary relation $\leq$, written $(A, \leq)$.

Total Orders

Elements a and b of a poset $(A, \leq)$ are *comparable* if either $a \leq b$ or $b \leq a$. Otherwise they are *incomparable*.

A poset $(A, \leq)$ is *totally ordered* if every pair of elements is comparable.

Totally ordered sets are also called *linearly ordered sets* and *chains*.

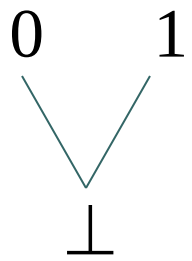
Examples

1. $0 < 1$
2. $1 < \infty$
3. child < parent
4. child > parent
5. 11,000/3,501 is a better approximation to π than 22/7
6. integer n is a divisor of integer m.
7. Set A is a subset of set B.

Which of these are partial orders? Total orders?

Which are the corresponding posets?

Flat Order



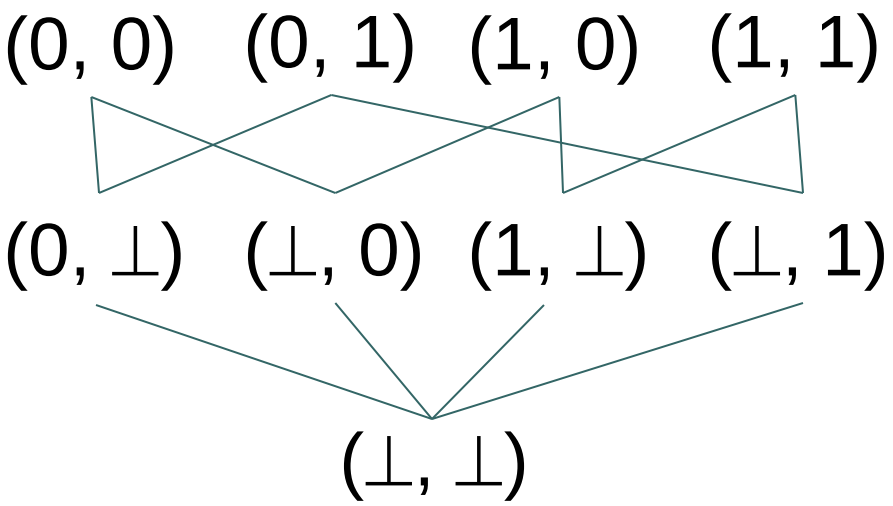
Hasse diagram

This means:

$$\perp \leq 0 \qquad \text{and} \qquad \perp \leq 1$$

The top layer can contain the elements of any data type, but since we are (for now) working with circuits, the set of binary digits is sufficient.

Flat Order on Tuples



Hasse diagram

This means:

$(\perp,0) \leq (1,0)$ $(\perp,1) \leq (1,1)$ …

This generalizes to arbitrary m-tuples.

Height is $m+1$

Join (Least Upper Bound)

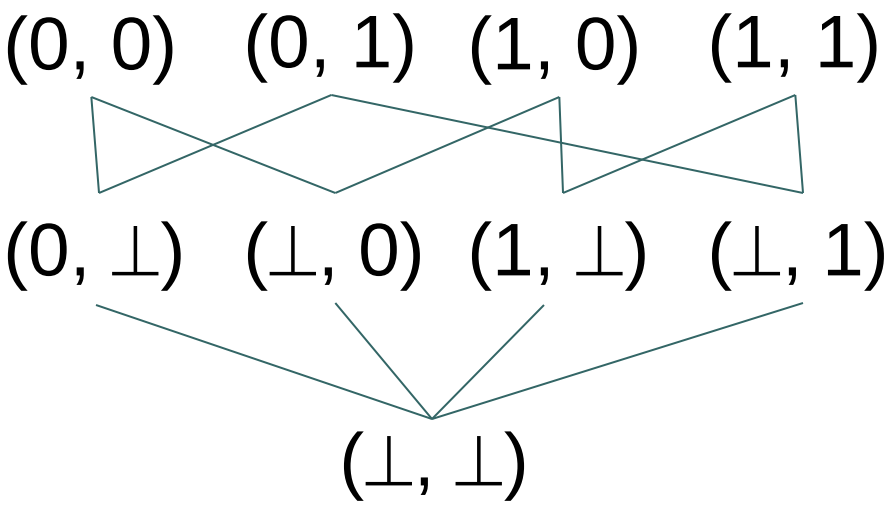
An *upper bound* of a subset $B \subseteq A$ of a poset $(A, \leq)$ is an element $a \in A$ such that for all $b \in B$ we have $b \leq a$.

A *least upper bound* (LUB) or *join* of B is an upper bound a such that for all other upper bounds a' we have $a \leq a'$.

The *join* of B is written $\vee B$.

When the join of B exists, then B is said to be *joinable*.

Examples



Hasse diagram

Does the upper bound exist for

$\{(0,\perp), (\perp, 0)\}$?

$\{(0,\perp), (\perp, 1)\}$?

Does the upper bound exist for: 0<1<2<… ?

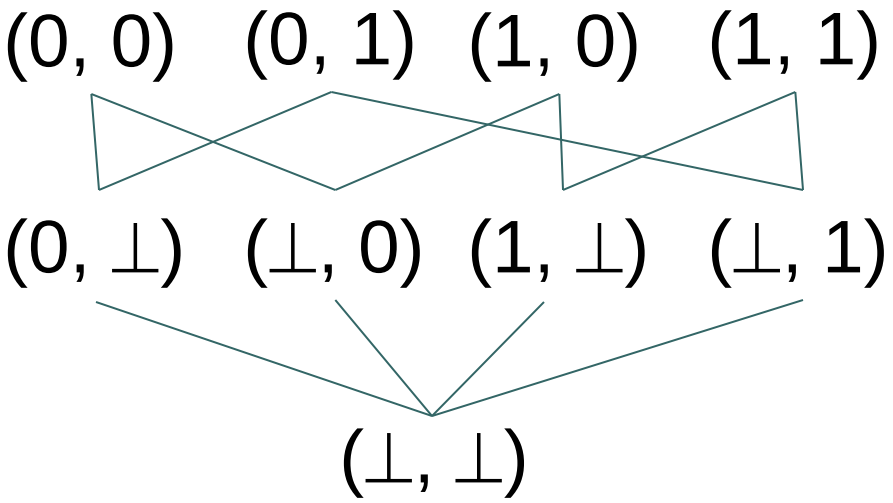
Monotonic (Order Preserving) Functions

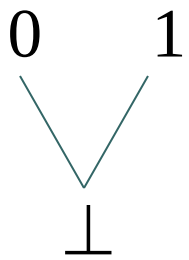
Let $(A, \leq)$ and $(B, \leq)$ be posets.

A function $f : A \rightarrow B$ is called *monotonic* if

$$a \leq a' \Rightarrow f(a) \leq f(a')$$

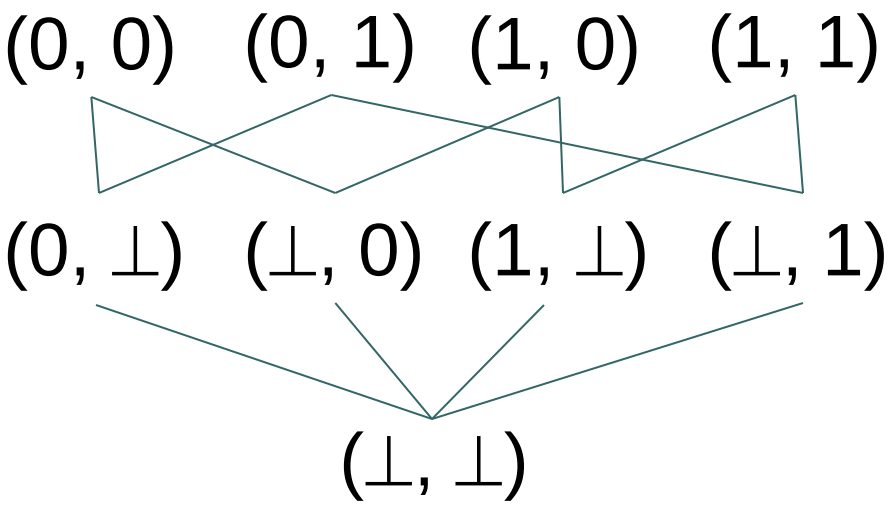
Parallel Or is Monotonic on the Flat Order

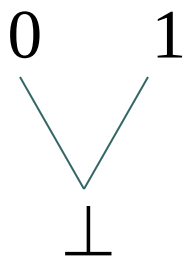




input 2 \ input 1	⊥	F	T
⊥	⊥	⊥	T
F	⊥	F	T
T	T	T	T

Parallel And is Monotonic on the Flat Order





input 1

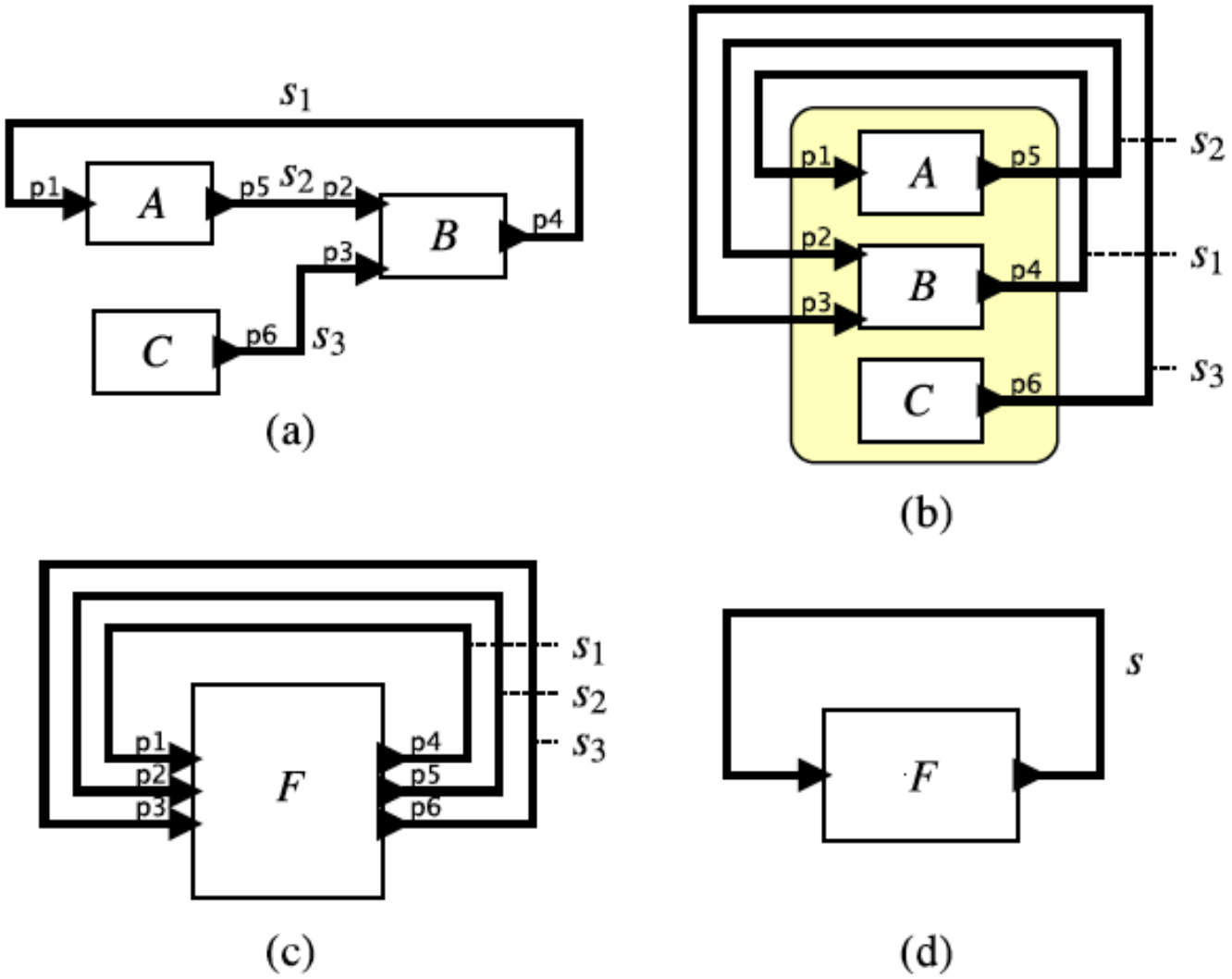
input 2	⊥	F	T
⊥	⊥	F	⊥
F	F	F	F
T	⊥	F	T

What about logical NOT ?

What does the extended truth table (with “uknown”) for NOT look like?

Is NOT monotonic?

A circuit (with inputs known) with m nodes can be represented as a function F:$\{0,1\}^m \rightarrow \{0,1\}^m$



Fixed Point Theorem

(a variant of the Kleene fixed-point theorem)

Let $(A, \leq)$ be a flat order on m-tuples

Let $f : A \rightarrow A$ be a monotonic function

Let $C = \{ f^n(\perp), n \in \{1, \ldots, m\} \}$

$\vee C = f^m(\perp)$ is the *least* fixed point of f

Intuition: The least fixed point of a monotonic function is obtained by applying the function first to unknown, then to the result, then to that result, etc.

Proof (part 1: is a fixed point)

Note that C is a chain in a finite poset:

$$\bot \leq f(\bot)$$

$$f(\bot) \leq f^2(\bot) \qquad \text{by monotonicity}$$

$$\ldots$$

$$f^{m\text{-}1}(\bot) \leq f^m(\bot)$$

Since the longest chain in the poset has length $m + 1$, this sequence has to stop increasing and settle to a fixed point.

Hence, $\vee C$ is a fixed point of f

Proof (part 2: is the least fixed point)

Let a be another fixed point: $f(a) = a$

Show that $\vee C$ is the least fixed point: $\vee C \leq a$

Since f is monotonic:

$\bot \leq a$

$f(\bot) \leq f(a) = a$

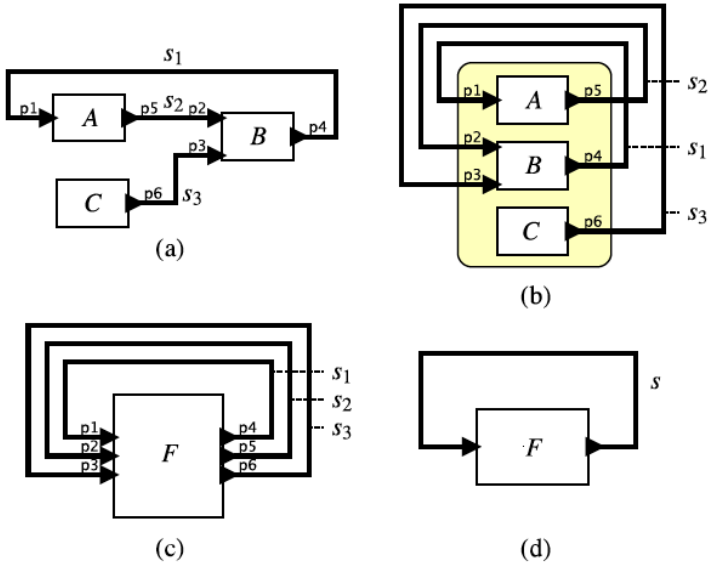
$\ldots$

$f^m(\bot) \leq f^m(a) = a$

So a is an upper bound of the chain C, hence $\vee C \leq a$.

Brute Force Application of the Theorem

- Start with signals “unknown” at all nodes of the circuit.
- Evaluate components (gates) in arbitrary order repeatedly until no further progress is made.
- If the result has all signals “known,” then declare it to be the constructive solution.
- Otherwise, reject model as non-constructive (buggy).



We can do better…

In the circuit below, evaluation order matters:

- 1, 2, 3, 4: requires three passes to converge.
- 3, 1, 4, 2: requires one pass to converge.

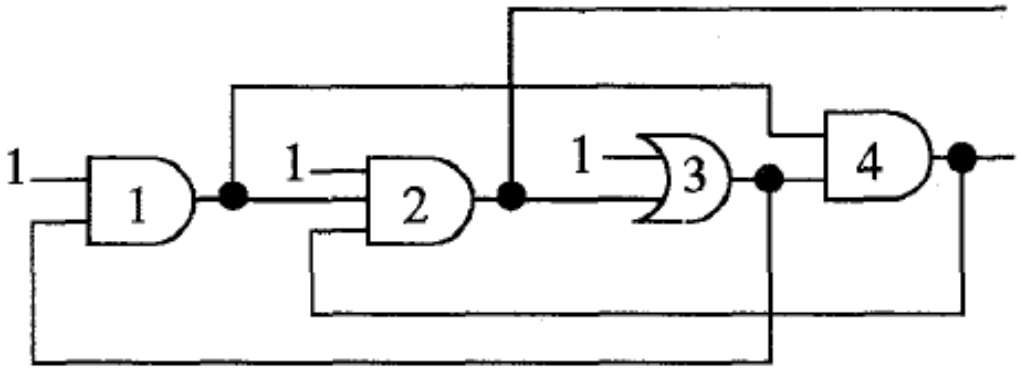


Figure 3: Number of gate evaluations depends on evaluation order.

[Shiple, Berry, and Touati, 1996]

Bourdoncle's Algorithm

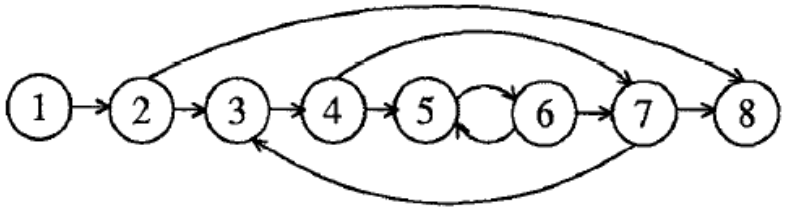


Figure 4: A directed graph.

Build a weak topological ordering (WTO) of a graph abstraction.

A WTO is a nesting of strongly-connected components (SCCs).

WTO for the above graph is (1, 2, (3, 4, (5, 6), 7), 8).

Execute as follows:

1. Start left to right.
2. Upon reaching a close parenthesis, iterate its SCC to find its fixed point.
3. Continue left to right.

So in the above example, we first evaluate 1,2,3,4,5,6. But then, instead of going to 7, we return to 5, and continue looping between 5 and 6 until there is no change. Then 7 is evaluated, and then we return to 3.

[Shiple, Berry, and Touati, 1996]

Complexity of Bourdoncle's Algorithm

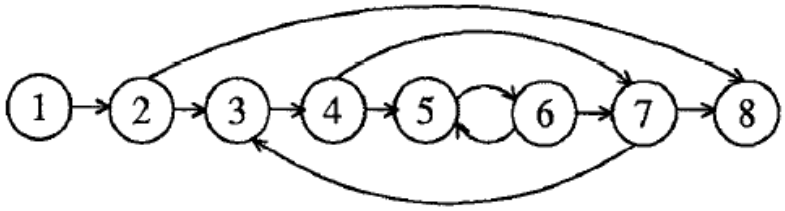


Figure 4: A directed graph.

WTO for the above graph is (1, 2, (3, 4, (5, 6), 7), 8).

The depth of an element is the number of nested components containing the element (e.g. element 3 has depth 1; 6 has depth 2).

Bourdoncle showed that the total number of evaluations is bounded by sum depth(v), where the sum is taken over all nodes. This contrasts to the brute force method, which is bounded by N^2, where N is the number of unknown wires.

[Shiple, Berry, and Touati, 1996]

Generalization of this algorithm uses branch-and-bound to find optimal schedules.

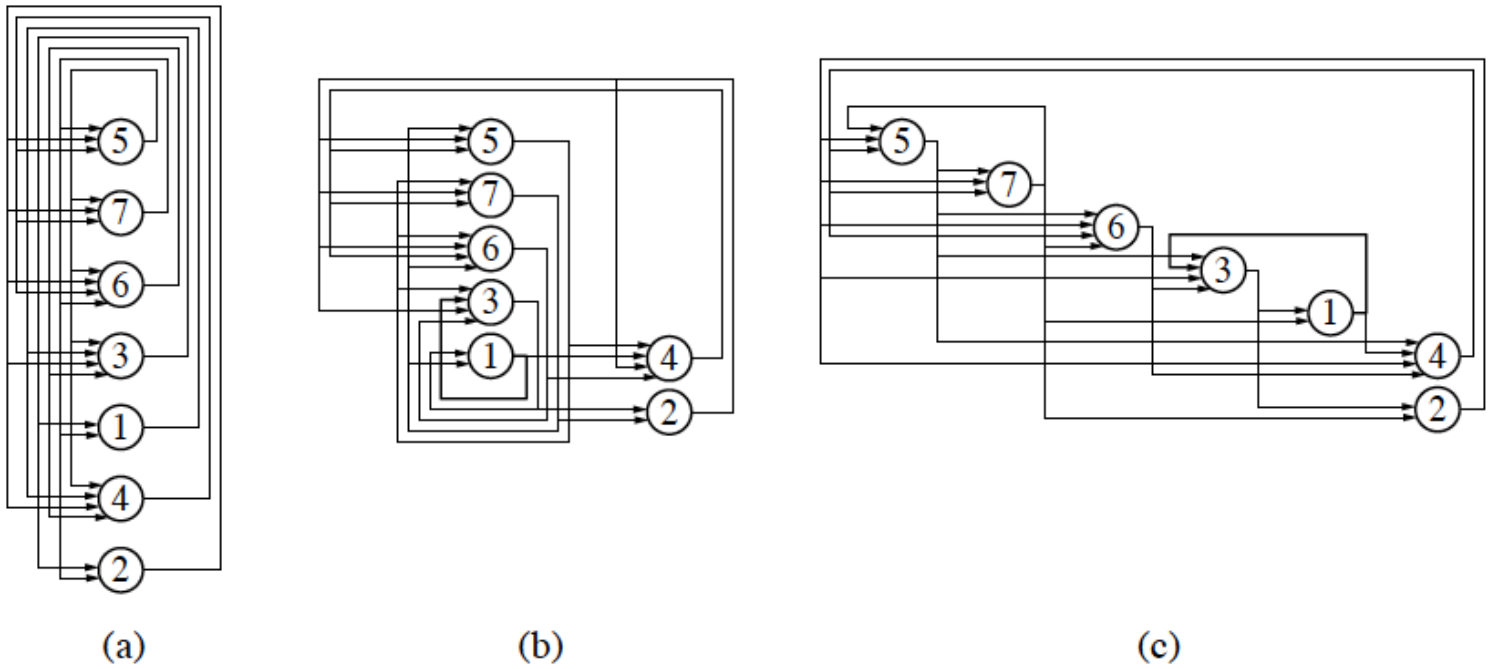
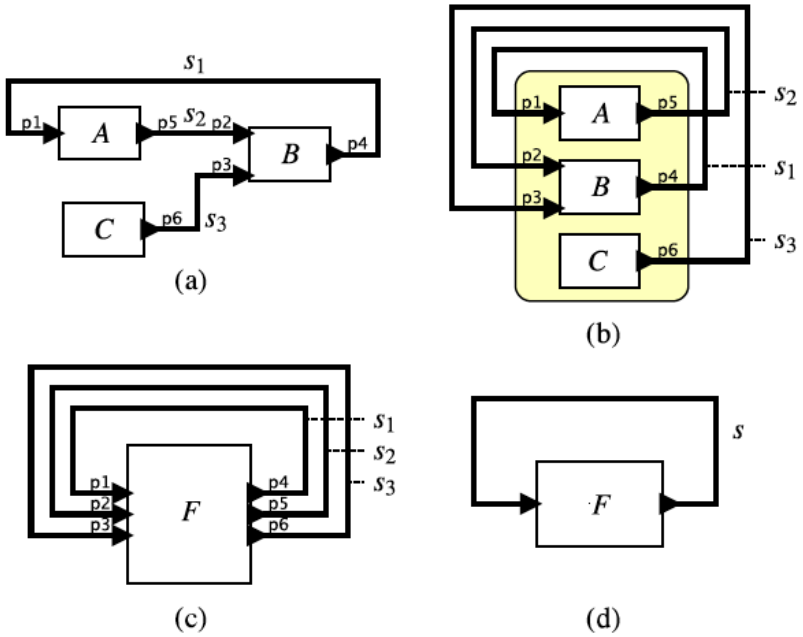


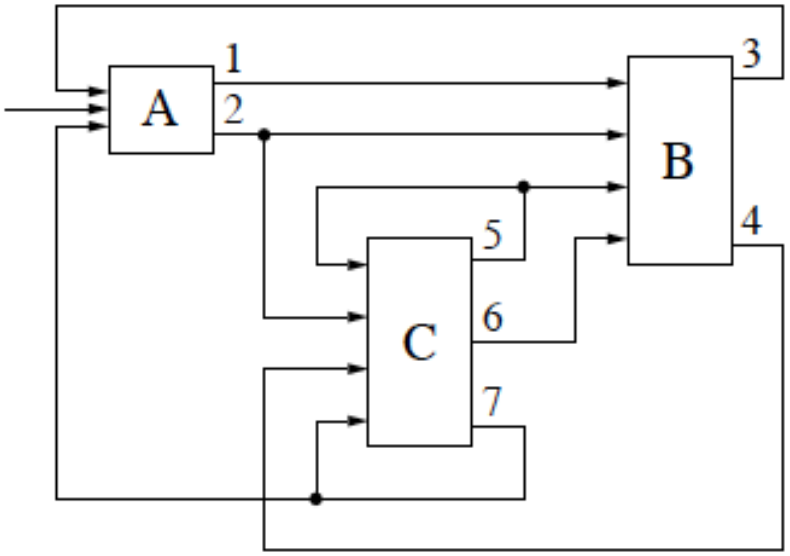
Fig. 4. Decomposing the dependency graph in Fig. 3b using Bekić's theorem. (a) A brute-force evaluation using Theorem 1 involves evaluating a feedback loop with seven wires. Cost: $7^2 = 49$, (b) splitting it into two using Bekić's theorem (the X function contains nodes 2 and 4, the others are part of the Y function) transforms the graph into an inner feedback loop with five wires and an outer loop with two. Cost: $2^2 + (2+1)5^2 = 79$, (c) Further decomposing the Y function transforms the five-element feedback loop into two loops (5 and 7, 3 and 1) of one wire each. Cost: $2^2 + (2+1)(3+1+3) = 25$.

[Edwards and Lee, Science of Computer Programming, 2003]

What if inputs are unknown?

We will extend the solution to **open** systems.

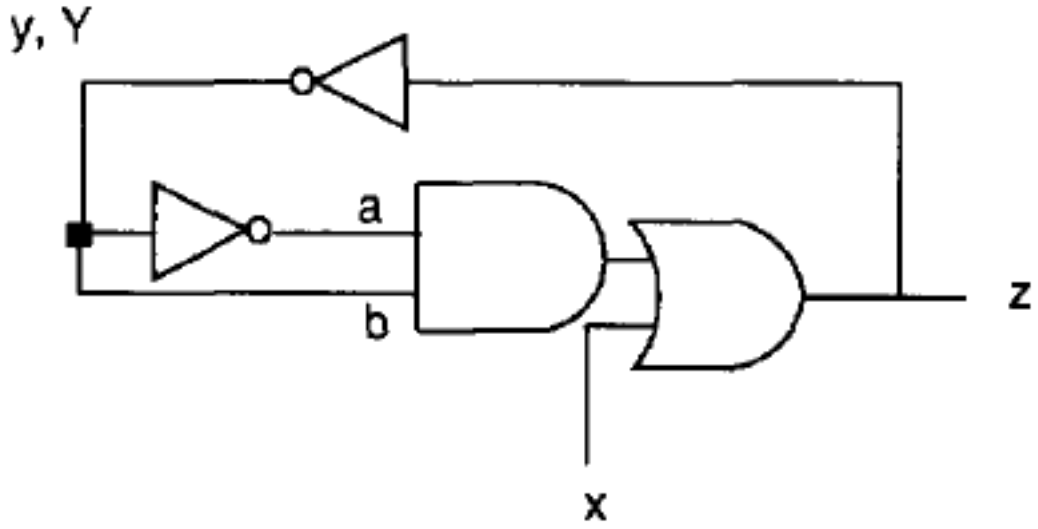




From closed …

… to open systems

Non-constructive, but seemingly combinational circuits



Logically, the output of the AND gate is 0. But if the input *x* is zero, and the inverters have delay, then this circuit will oscillate.

The problem is that, with methods presented so far, we have to test whether the circuit is constructive for all possible inputs.

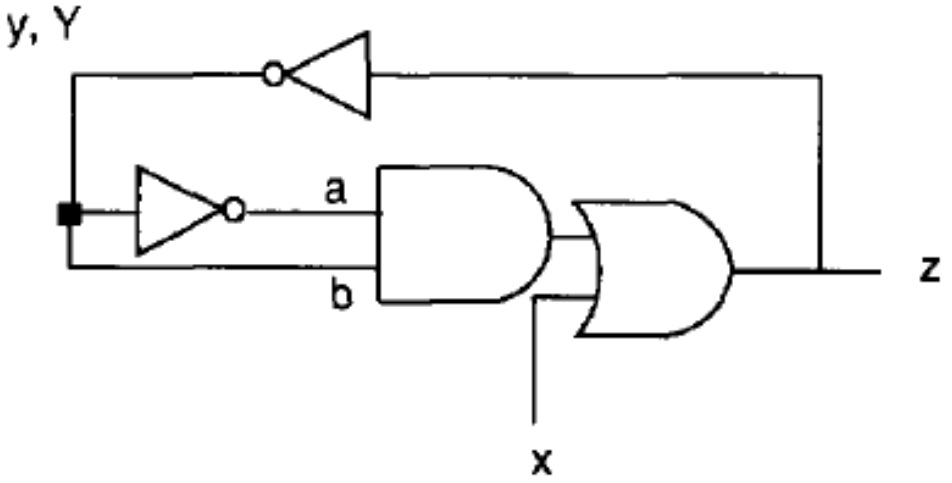
[Malik, Trans. on CAD, 1994]

Instead: use **Symbolic Execution**

Solution procedure:

- Start with unknown *function of the inputs* at all nodes except inputs.
- Update the functions in arbitrary order repeatedly until no further progress is made.
- If the result has all functions known, then declare the circuit constructive.

Symbolic Execution



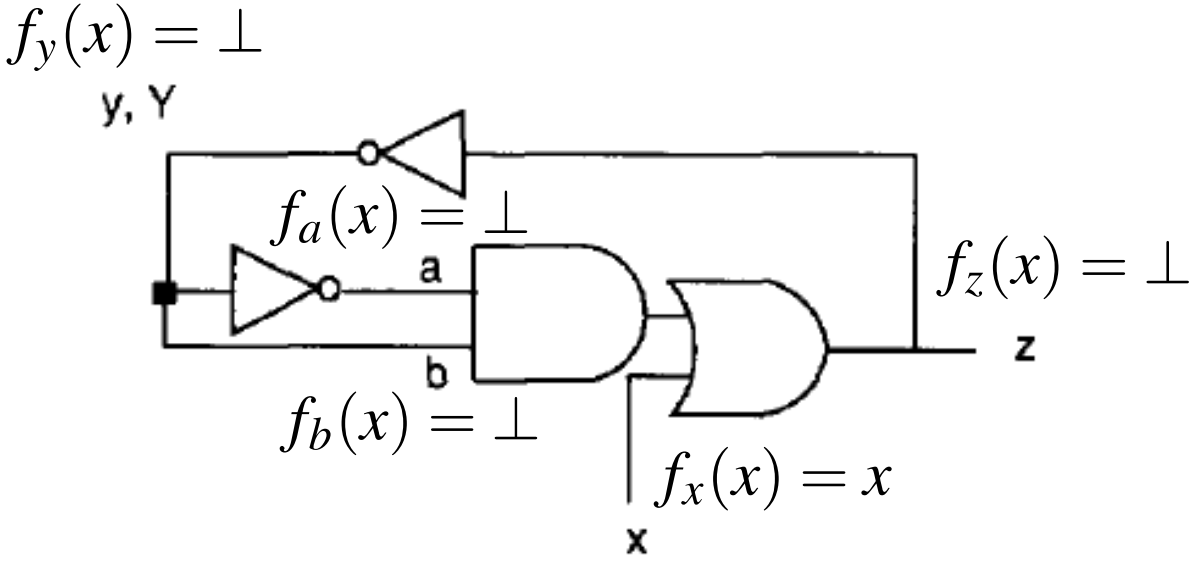
Assume a single binary input (for now). For each node a in the circuit, define a function from the input to the node value:

$$f_a : \{0,1\} \rightarrow \{\perp,0,1\}$$

These give the outputs as a function of x only.

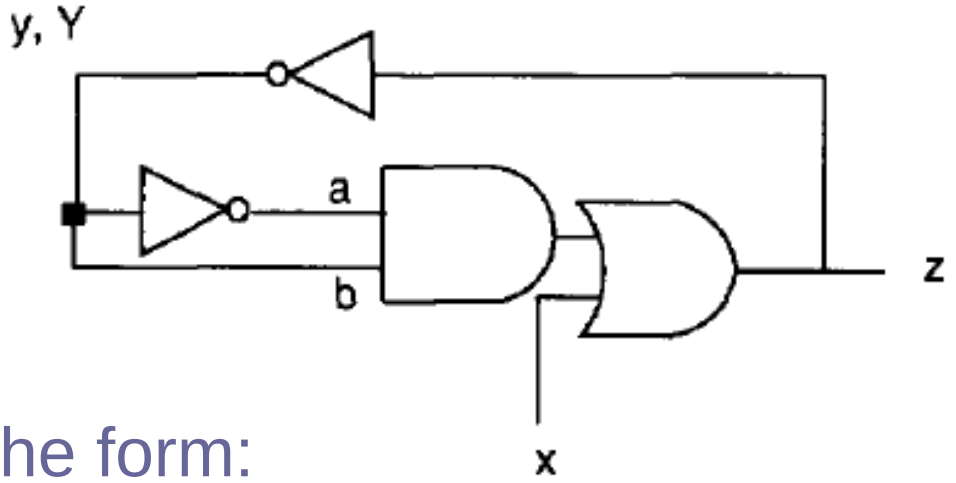
Symbolic Execution Strategy

Start with all nodes except inputs being given by the unknown function:



Then update these functions iteratively until convergence. But how to update the functions?

Idea



Represent each function of the form:

$$f_a : \{0,1\} \rightarrow \{\perp, 0, 1\}$$

using **two** *characteristic functions* of the form:

$$f_a^0 : \{0,1\} \rightarrow \{0,1\}$$
$$f_a^1 : \{0,1\} \rightarrow \{0,1\}$$

where

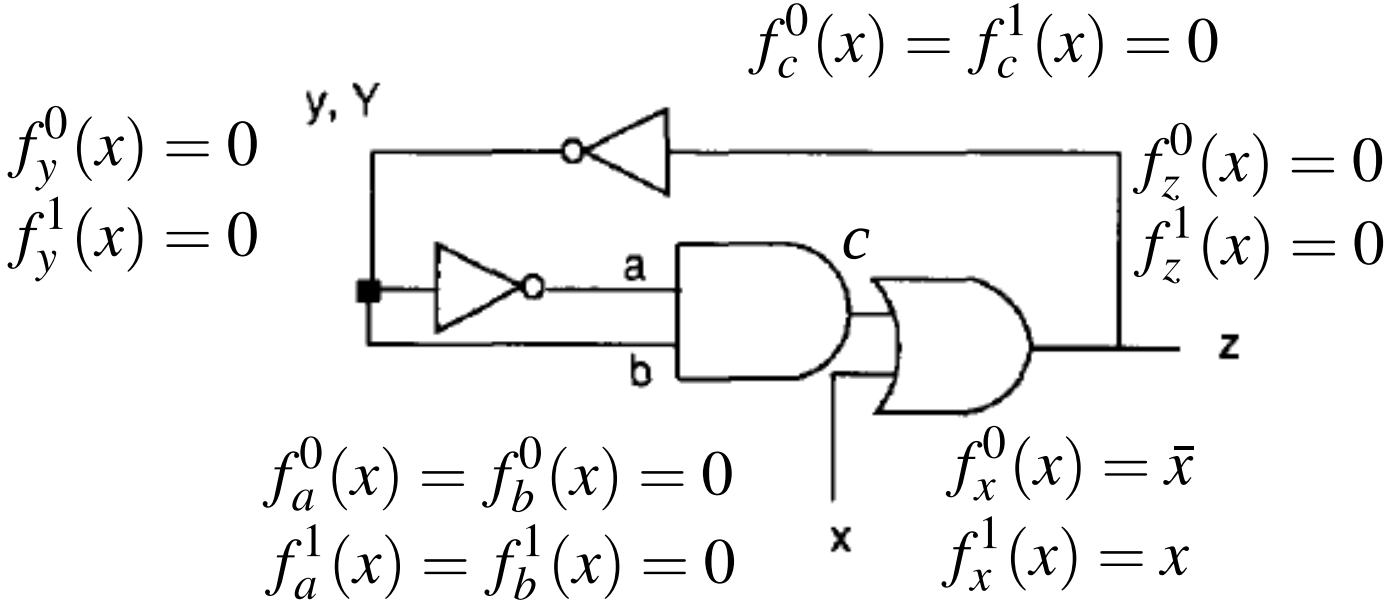
$$f_a^0(x) = \begin{cases} 1 & \text{if } f_a(x) = 0 \\ 0 & \text{otherwise} \end{cases} \qquad f_a^1(x) = \begin{cases} 1 & \text{if } f_a(x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

How can these be represented in practice?

Using **BDDs**!

Symbolic Execution Strategy using Characteristic Functions

Start with all nodes except inputs being given by the unknown function:



Then update these functions iteratively until convergence. But how to update the functions?

Operating on Characteristic Functions

Gates relate characteristic functions of the outputs with those of the inputs:

$$f_c^0(x) = f_a^1(x)$$
$$f_c^1(x) = f_a^0(x)$$

$$f_c^0(x) = f_a^0(x) + f_b^0(x)$$
$$f_c^1(x) = f_a^1(x) \cdot f_b^1(x)$$

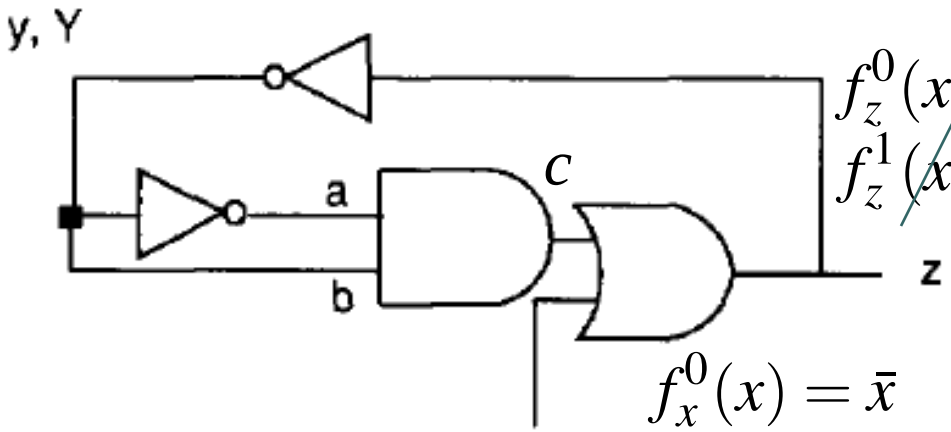
$$f_c^0(x) = f_a^0(x) \cdot f_b^0(x)$$
$$f_c^1(x) = f_a^1(x) + f_b^1(x)$$

Symbolic Execution Strategy using Characteristic Functions

Update nodes in arbitrary order:

$$f_z^0(x) = f_c^0(x) \cdot f_x^0(x) = 0 \cdot \bar{x} = 0$$
$$f_z^1(x) = f_c^1(x) + f_x^1(x) = 0 + x = x$$

$$f_y^0(x) = 0$$
$$f_y^1(x) = 0$$



$$f_z^0(x) = 0$$
$$f_z^1(x) = 0$$

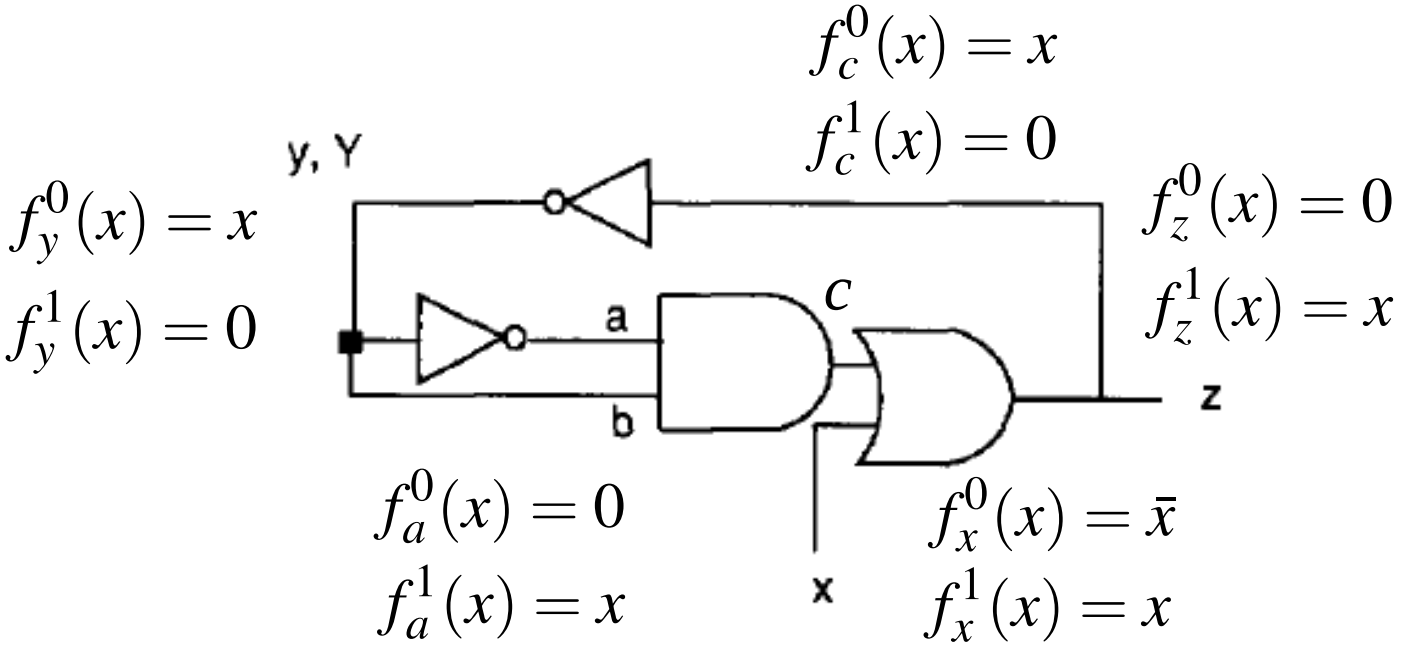
$$f_y^0(x) = f_z^1(x) = x$$
$$f_y^1(x) = f_z^1(x) = 0$$

$$f_x^0(x) = \bar{x}$$
$$f_x^1(x) = x$$

etc.

Symbolic Execution Strategy using Characteristic Functions Convergence

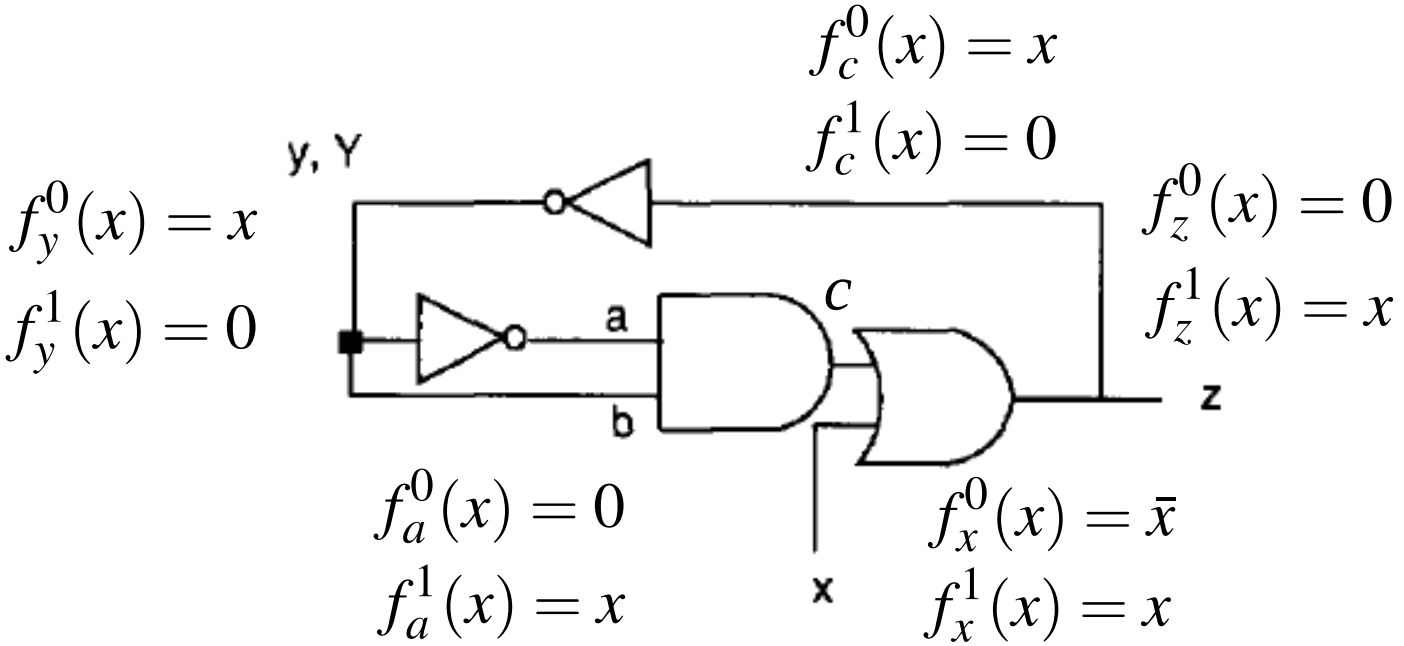
Quickly converge to these characteristic functions:



How do we know whether the circuit is constructive?

Symbolic Execution Strategy using Characteristic Functions Convergence

Quickly converge to these characteristic functions:



Circuit is constructive iff at all nodes *a* we have for all *x*

$$f_a^0(x) + f_a^1(x) = 1$$

i.e. the value is known! (Checking this is a SAT problem)

Does the procedure always converge? Is the answer unique?

Consider a poset {0, 1} where 0 < 1.

This induces a poset on the set of functions of form:

$$f_a^i : \{0,1\} \rightarrow \{0,1\}$$ How?

This poset has a bottom element: the function

$$f_\perp^i(x) = 0$$

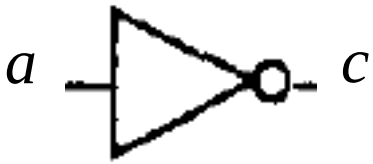
This poset is finite, with structure much like the flat order. The Kleene fixed-point theorem applies. Extends easily to tuples of functions.

Gate Operations on Characteristic Functions are Monotonic Functions!

These are monotonic in the sense that if you know more about the inputs, then you learn more about the outputs:

$$(f_a^0, f_a^1) \leq (g_a^0, g_a^1) \Rightarrow (f_c^0, f_c^1) \leq (g_c^0, g_c^1)$$

$$((f_a^0, f_b^0), (f_a^1, f_b^1)) \leq ((g_a^0, g_b^0), (g_a^1, g_b^1)) \Rightarrow (f_c^0, f_c^1) \leq (g_c^0, g_c^1)$$



$$f_c^0(x) = f_a^1(x)$$
$$f_c^1(x) = f_a^0(x)$$

$$f_c^0(x) = f_a^0(x) + f_b^0(x)$$
$$f_c^1(x) = f_a^1(x) \cdot f_b^1(x)$$

$$f_c^0(x) = f_a^0(x) \cdot f_b^0(x)$$
$$f_c^1(x) = f_a^1(x) + f_b^1(x)$$

Extension to sequential circuits

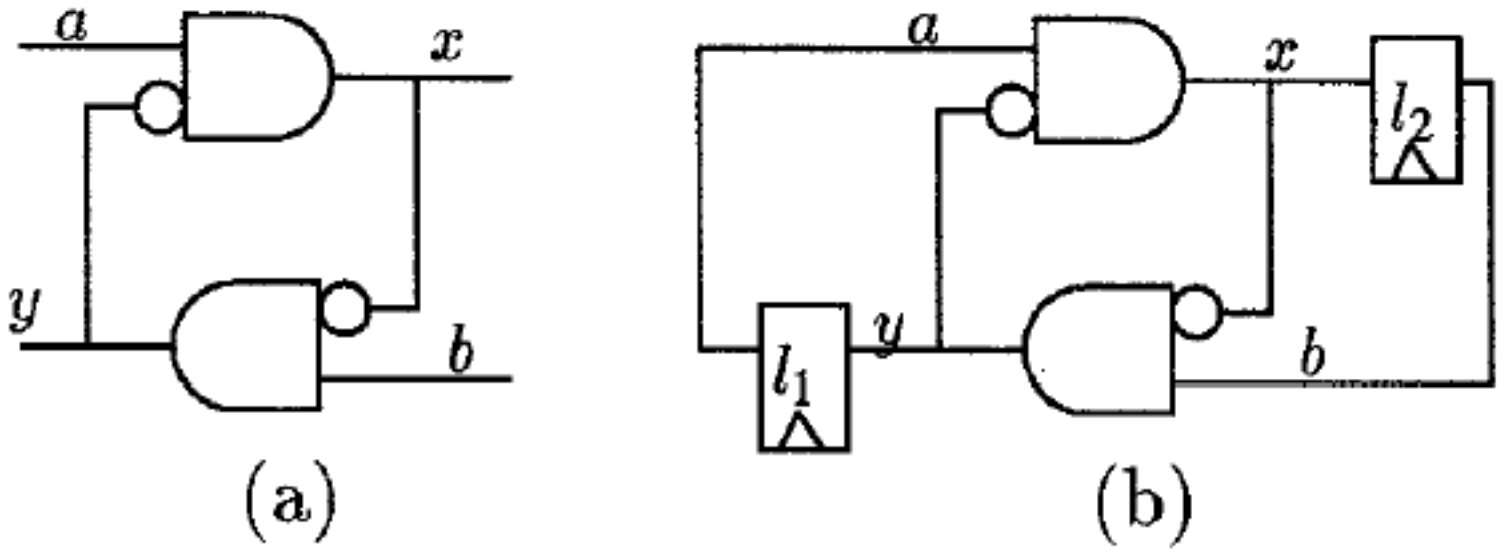


Figure 1: Circuits are well-behaved unless $a = b = 1$.

First need to find which inputs are problematic.

Then need to determine whether those inputs can occur (reachability analysis on a state machine)

[Shiple, Berry, and Touati, DATE, 1996]