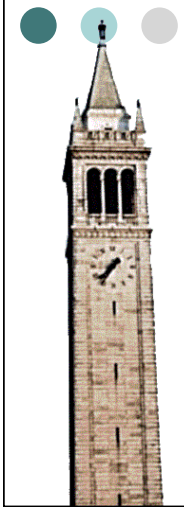


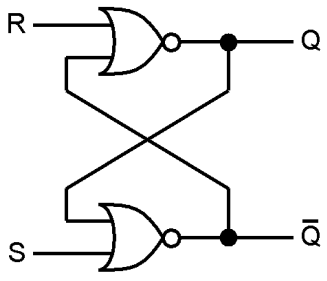
Fundamental Algorithms for System Modeling, Analysis, and Optimization

Edward A. Lee, Jaijeet Roychowdhury, Sanjit A. Seshia
UC Berkeley
EECS 144/244
Fall 2011

Lecture N: Synchronous Systems

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).

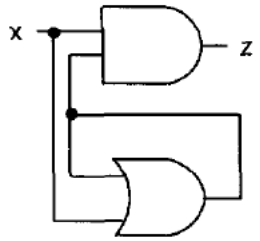


Fig. 1. Cyclic Combination Circuits: A Simple Example.

[Malik, Trans. on CAD, 1994]

Practical Cyclic Combinational Circuits

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

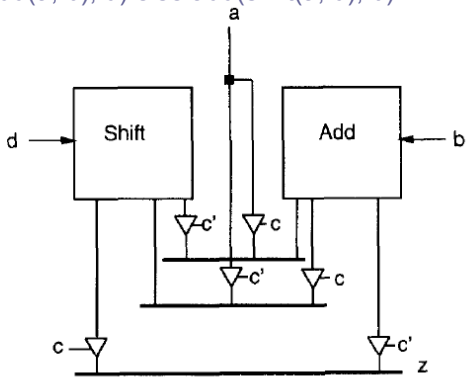


Fig. 3. Cyclic combinational circuits: practical example.

[Malik, Trans. on CAD, 1994]

Esterel:
A Synchronous/Reactive Programming Language

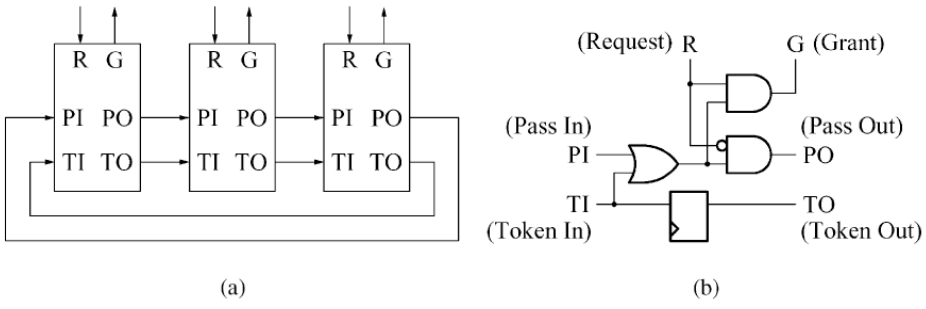
```
present I then
  present S then emit T end
else
  present T then emit S end
end
```

"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, DATE, 1996]

Practical Cyclic Combinational Circuits

Token ring protocol manager.



Considering the path from latch outputs to latch inputs, the circuit has cycles but is combinational.
[Edwards & Lee, *Science of Computer Programming*, 2003]

Cyclic Combinational Circuits with Sequential Parts

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

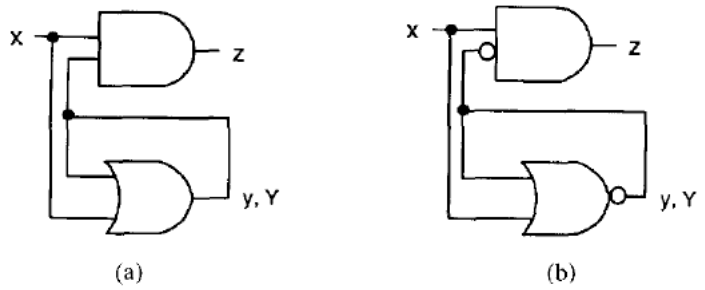


Fig. 4. Cyclic combinational circuits with sequential parts.

If x=1, circuit (b) may oscillate if there are gate delays.
[Malik, Trans. on CAD, 1994]

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

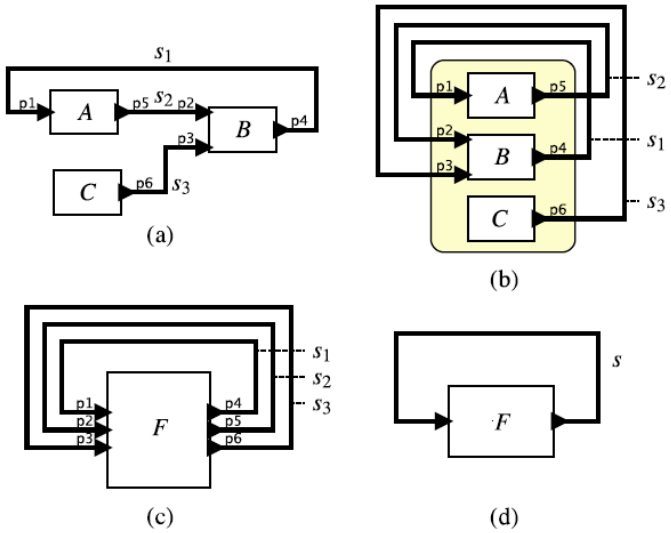
Solution: Constructive Fixed-Point Semantics

Two parts to this solution:

1. Fixed-point semantics
2. Constructive semantics

Fixed-Point Semantics

We will develop the solution first with 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 feedback paths.
- Evaluate components (gates) in arbitrary order repeatedly until no further progress is made.
- If the result has all signals "known," then declare the constructive solution.

Applying the procedure

Initialize with input values and "unknown" on other nodes.

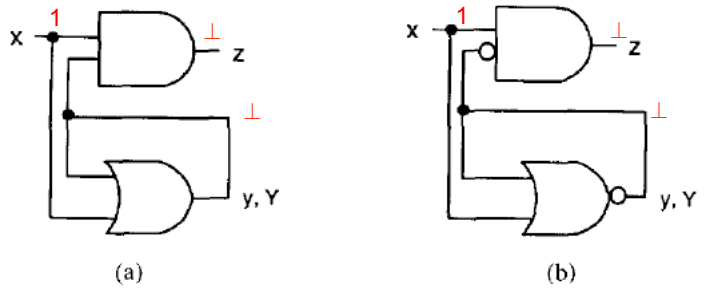


Fig. 4. Cyclic combinational circuits with sequential parts.

[Malik, Trans. on CAD, 1994]

Applying the procedure

Evaluate lower gates.

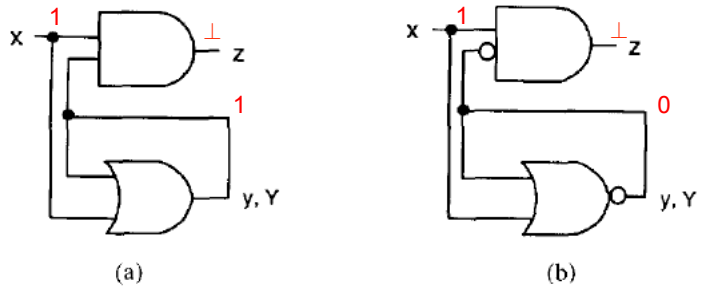


Fig. 4. Cyclic combinational circuits with sequential parts.

[Malik, Trans. on CAD, 1994]

Applying the procedure

Evaluate gates in arbitrary order until no progress is made.

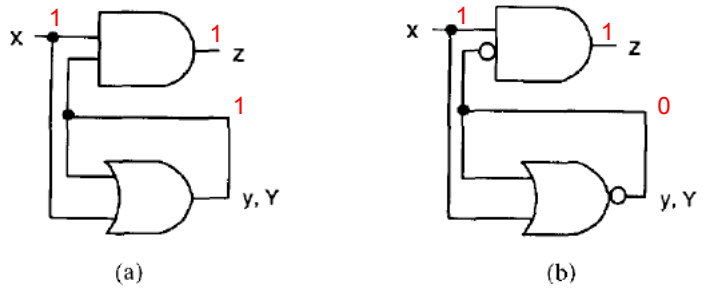
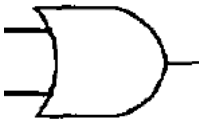


Fig. 4. Cyclic combinational circuits with sequential parts.

At this point, all nodes are known.
[Malik, Trans. on CAD, 1994]

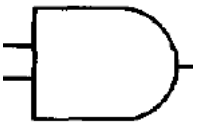
Implicitly using Parallel (Non-Strict) Or



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	T
T	⊥	T	T

Applying the procedure with input 0

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

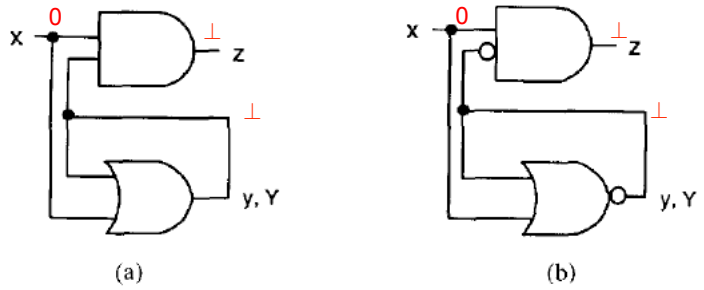


Fig. 4. Cyclic combinational circuits with sequential parts.

[Malik, Trans. on CAD, 1994]

Applying the procedure with input 0

Evaluate gates in arbitrary order until no progress is made.

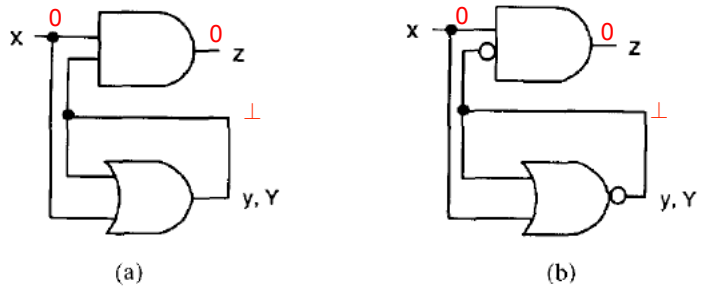


Fig. 4. Cyclic combinational circuits with sequential parts.

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 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.

What is Order?

Intuition:

1. 0 < 1
2. 1 < ∞
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?

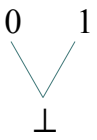
Partial Orders

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 $(A, \leq)$.

Flat Order



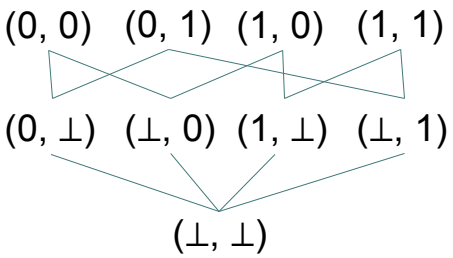
Hasse diagram

This means:

$\perp \leq 0$ $\perp \leq 1$

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

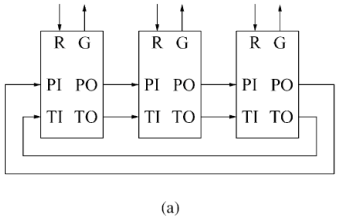
This means:

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

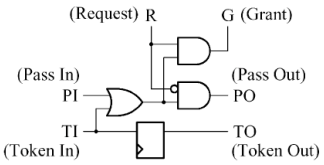
This generalizes to arbitrary m-tuples.

Height is $m+1$

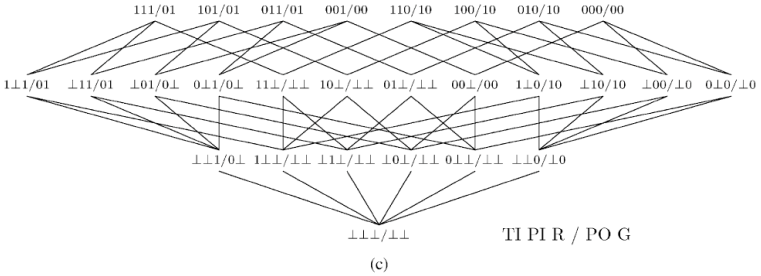
Flat order for practical application



(a)



(b)



(c)

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

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*.

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*.

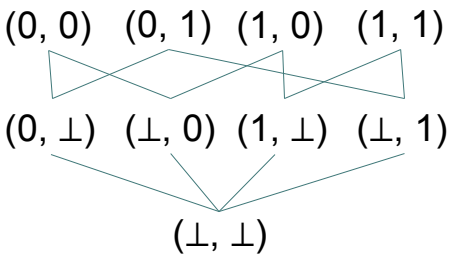
Monotonic (Order Preserving) Functions

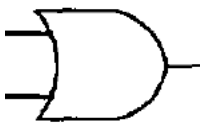
Let $(A, \leq)$ and $(B, \leq)$ be posets.

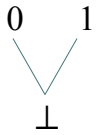
A function $f: A \rightarrow B$ is called *monotonic* if

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

Parallel Or is Monotonic on the Flat Order



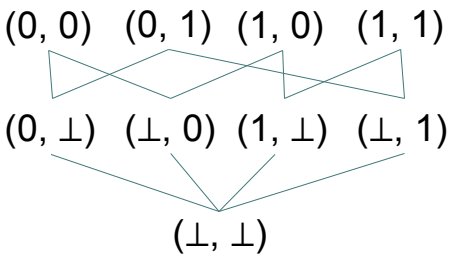


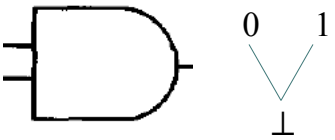


input 1

input 2	⊥	F	T
⊥	⊥	⊥	T
F	⊥	F	T
T	T	T	T

Parallel And is Monotonic on the Flat Order

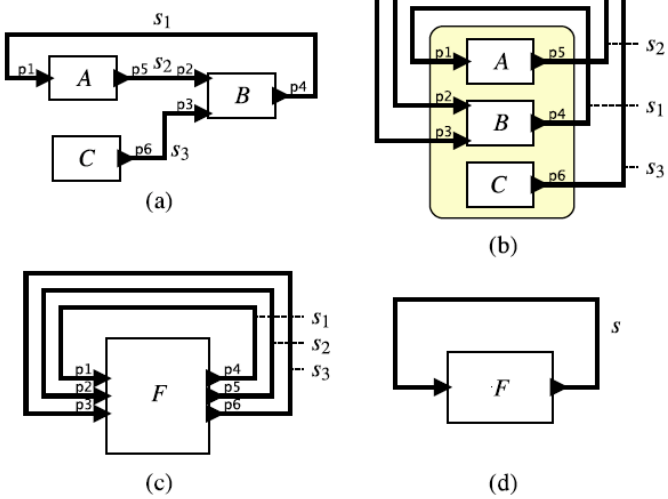




input 1

input 2	⊥	F	T
⊥	⊥	F	⊥
F	F	F	F
T	⊥	F	T

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

$\vee C = f^m(\bot)$ 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:

$\bot \leq f(\bot)$

$f(\bot) \leq f^2(\bot)$ by monotonicity

…

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

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:

$\bot \leq a$

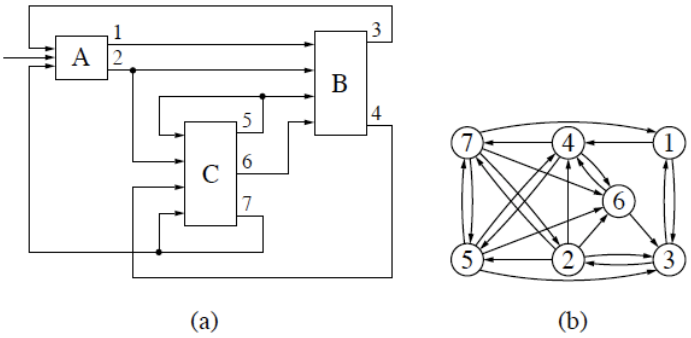
$f(\bot) \leq f(a) = a$

$\ldots$

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

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

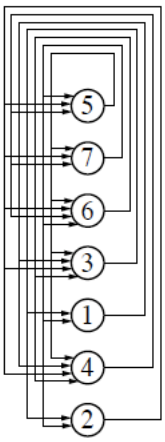
Brute Force Application of the Theorem



Given a circuit (a), construct a dependency graph of the nodes of the circuit. (Self loops are removed).

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

Brute Force Application of the Theorem



Form a monotonic function as a parallel composition of all the (monotonic) components. Evaluate this function (evaluate all the components) until all signals are known.

Each evaluation either makes at least one more signal known, or has converged to a fixed point. Therefore, the number of evaluations in the worst case is N^2, where N is the number of components.

But 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.

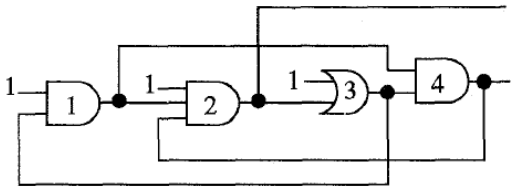


Figure 3: Number of gate evaluations depends on evaluation order.

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

Bourdoncle's Algorithm

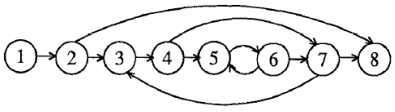


Figure 4: A directed graph.

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, DATE, 1996]

Complexity of Bourdoncle's Algorithm

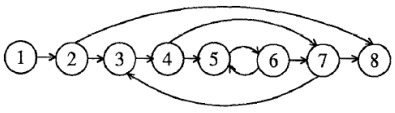


Figure 4: A directed graph.

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, DATE, 1996]

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

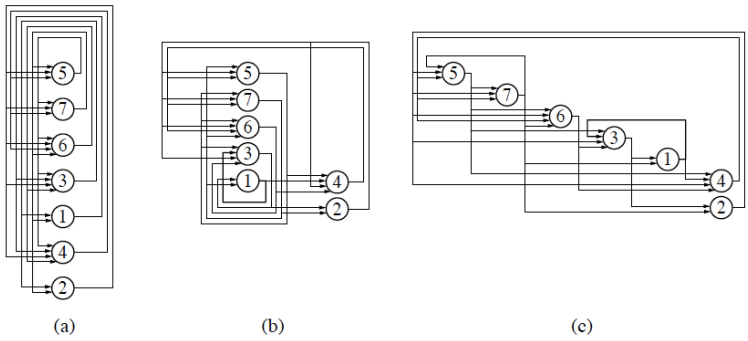


Fig. 4. Decomposing the dependency graph in Fig. 3b using Bekić's theorem. (a) A brute-force evaluation using Theorem 1 involves evaluating a feedback loop with seven wires. Cost: $7^2 = 49$, (b) splitting it into two using Bekić's theorem (the X function contains nodes 2 and 4, the others are part of 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]

Circuits that are constructive for some inputs (and a use where in some states, it is constructive)

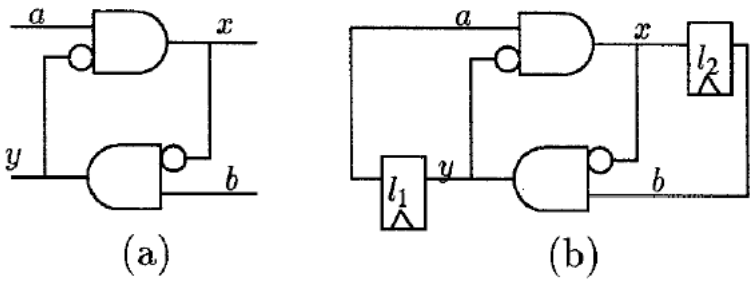
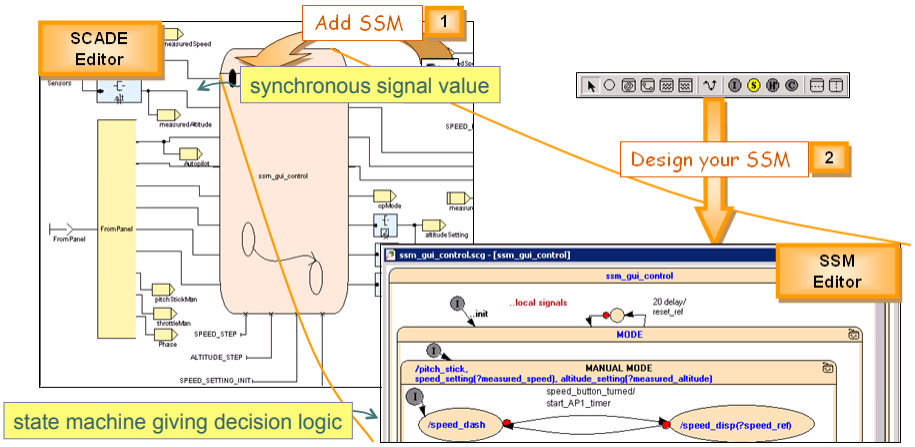


Figure 1: Circuits are well-behaved unless $a = b = 1$.

The question becomes: is it possible to have a = b = 1 (is this state *reachable*)? [Shiple, Berry, and Touati, DATE, 1996]

Synchronous Languages: Stateful systems

Example: SCADE is based on Lustre and Esterel. Synchronous composition of state machines.



from http://www.esterel-technologies.com/

Constructive Composition of State Machines

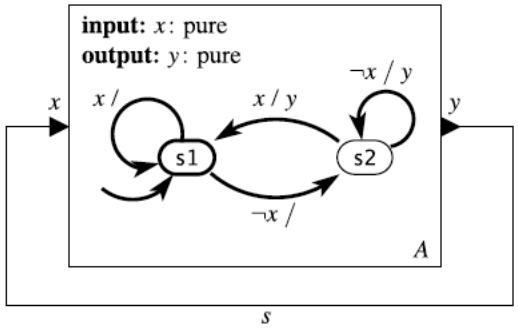


Figure 6.2: A simple well-formed feedback model.

[Lee & Seshia, *Introduction to Embedded Systems, 2010*]

Constructive Composition of State Machines

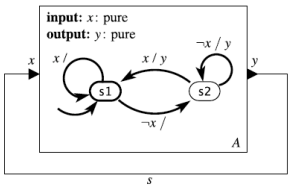


Figure 6.2: A simple well-formed feedback model.

Note that combinational vs. sequential was a red herring! The real issue is well-formed-ness and constructiveness!

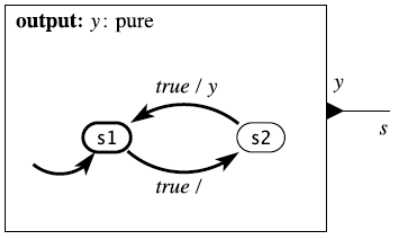


Figure 6.3: The semantics of the model in Figure 6.2.

[Lee & Seshia, *Introduction to Embedded Systems, 2010*]

Constructive Composition of State Machines

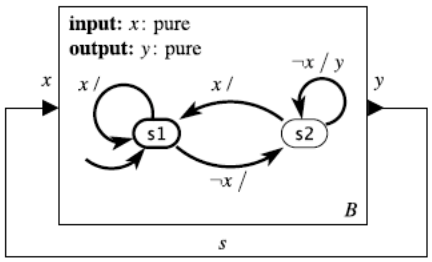


Figure 6.4: An ill-formed feedback model that has no fixed point in state s2.

[Lee & Seshia, *Introduction to Embedded Systems, 2010*]

Constructive Composition of State Machines

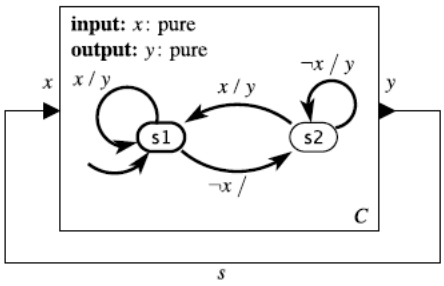


Figure 6.5: An ill-formed feedback model that has more than one fixed point in state s1.

[Lee & Seshia, *Introduction to Embedded Systems, 2010*]

Constructive Composition of State Machines

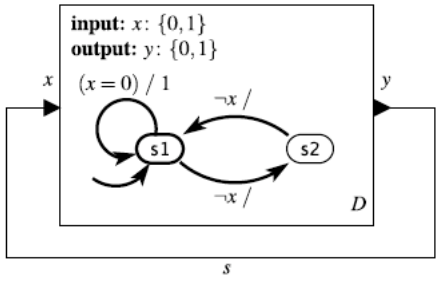


Figure 6.6: A well-formed feedback model that is not constructive.

[Lee & Seshia, *Introduction to Embedded Systems, 2010*]

References

1. Bourdoncle, F. (1993). Efficient chaotic iteration strategies with widenings. International Conference on Formal Methods in Programming and their Applications, Novosibirsk, Russia, Springer-Verlag.
2. Malik, S. (1994). "Analysis of Cyclic Combinational Circuits " IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 13(7): 950-956.
3. Shiple, T. R., G. Berry and H. Touati (1996). Constructive Analysis of Cyclic Circuits. European conference on Design and Test (DATE).
4. Berry, G. (1999). *The Constructive Semantics of Pure Esterel* - Draft Version 3, Book Draft.
5. Edwards, S. A. and E. A. Lee (2003). "The Semantics and Execution of a Synchronous Block-Diagram Language." *Science of Computer Programming* 48(1): 21-42.
6. E. A. Lee & S. A. Seshia, *Introduction to Embedded Systems – A Cyber-Physical Systems Approach, 2010, http://LeeSeshia.org.*