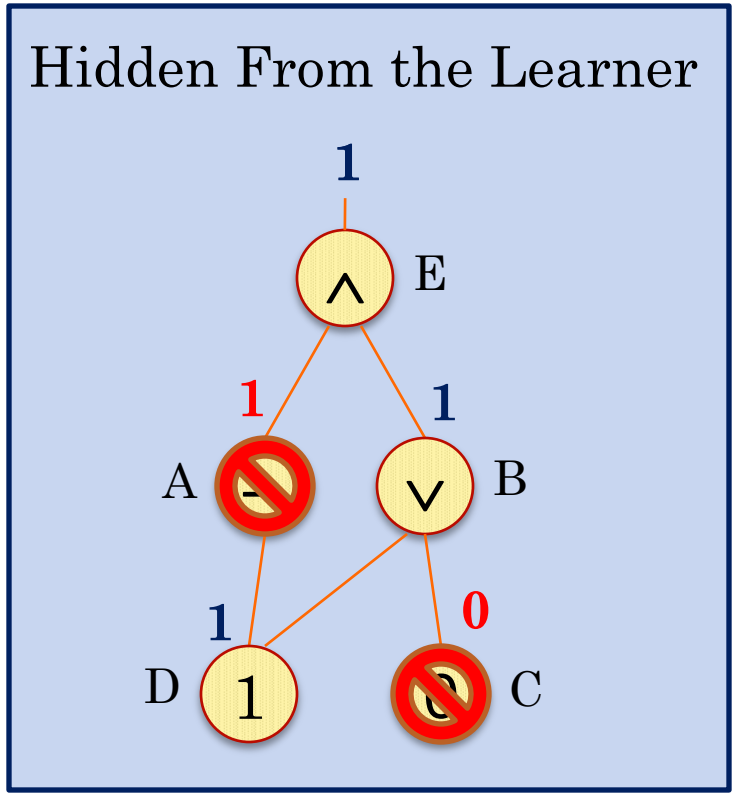
Learning Large-Alphabet and Analog Circuits with Value Injection Queries

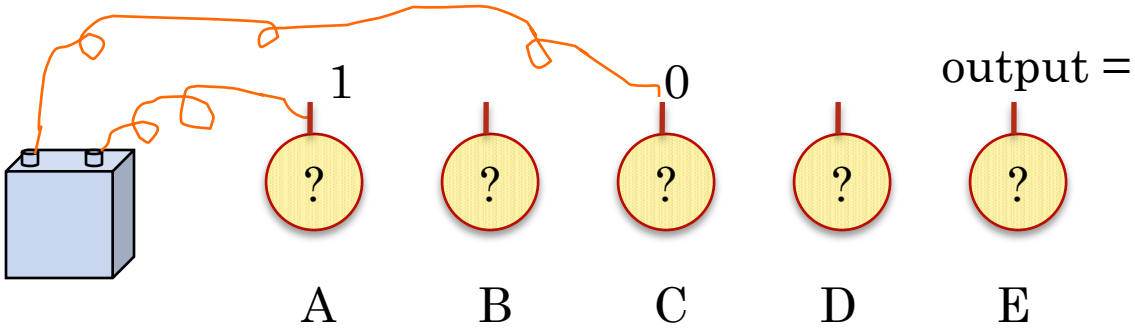
Dana Angluin[1]
James Aspnes[1]
Jiang Chen[2]
Lev Reyzin[1]

[1] Department of Computer Science, Yale University
[2] Center for Computational Learning Systems, Columbia University

The Value Injection Query Model

- Introduced by [AACW '06]
- Experiments on a hidden Circuit.
  - a gate output may be fixed
  - a gate may be left free
- Query
  - given an experiment, we can observe its output
- Example:

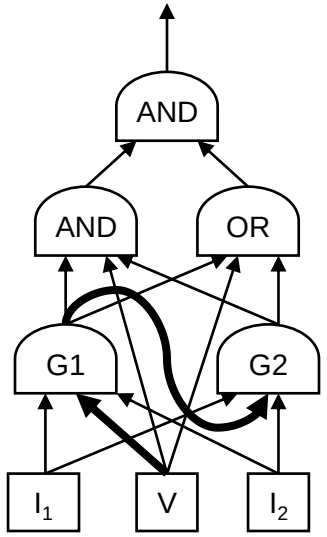


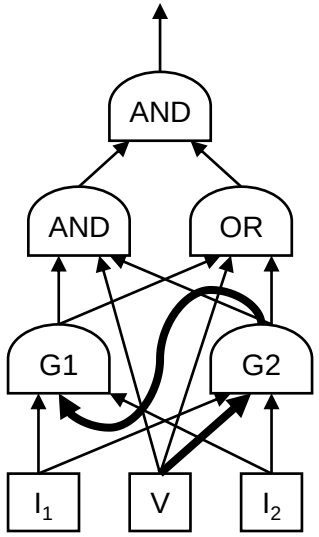


The Learning Problem

- **Behavioral equivalence:** Two circuits C and C' are behaviorally equivalent if for any experiment s, C(s)=C'(s).

- **The Problem:** Given qeuery access to a hidden circuit C^*, find a circuit C behaviorally equivalent to C^* by making value-injection queries.





[ACCW '06]

Motivation for The Model

- To model gene regulatory networks as boolean networks
- to represent gene expressions and disruptions

Previous gene regulatory network model	Fully controllable.	All gates are observable.
Existing circuit learning models	Only inputs can be manipulated.	Only the output is observable.
[AACW '06] model	Fully controllable.	Only the output is observable.

IN BETWEEN

[AACW '06] Results for Boolean Circuits

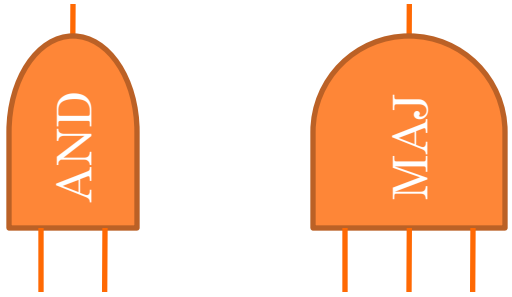
Depth	**Fan-in**	**Gates**	**Learnability**
Unbounded	Unbounded	AND/OR	$2^{\Omega(N)}$ queries
Unbounded	2	AND/OR	NP-hard
Constant	Unbounded	AND/OR/Θ_2	NP-hard
Log	Constant	Arbitrary	Poly-time **(NC1)**
Constant	Unbounded	AND/OR/NOT	Poly-time **(AC0)**

Looking at Large Alphabet Circuits

- Gene regulatory networks have more states than just expressed and disrupted.
- A larger alphabet than {0,1} is needed to more fully represent many other types of networks.
- Looking at what happens for large alphabet size is a natural, interesting theoretical question.

Large-Alphabet Circuits

Gates in Boolean Circuits



Input 1	Input 2	Output
1	1	O_1
1	0	O_2
0	1	O_3
0	0	O_4

Gates in Large-Alphabet circuits

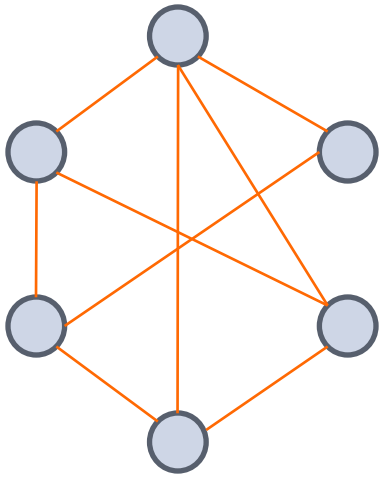
Input 1	Input 2	Output
A	A	O_1
A	B	O_2
A	C	O_3
B	A	O_4
B	B	O_5
B	C	O_6
C	A	O_7
C	B	O_8
C	C	O_9

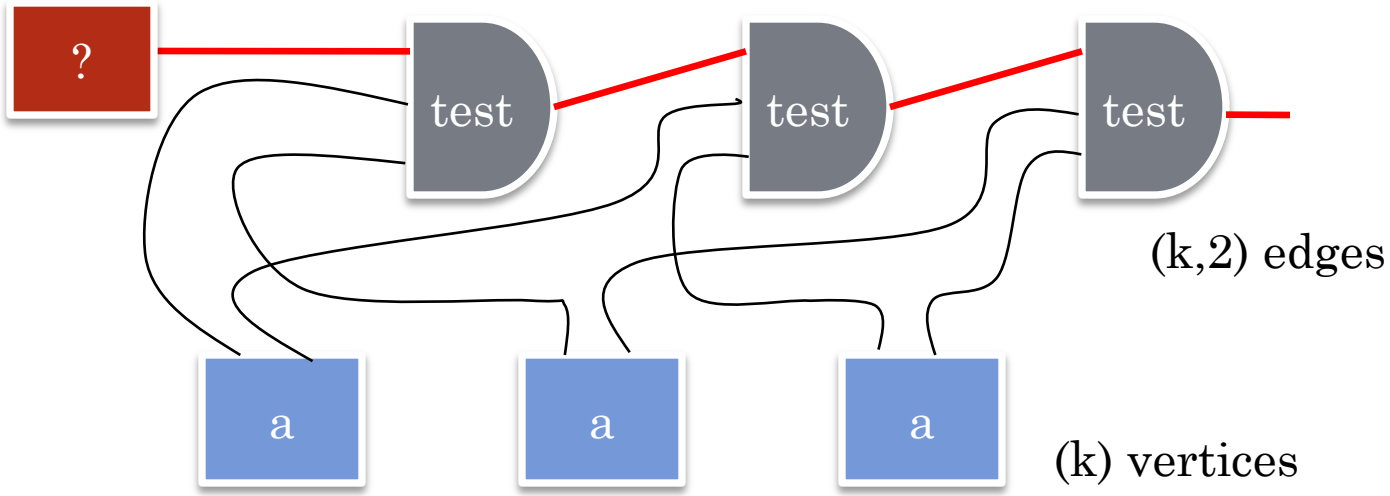
What Happens For Large-Alphabet Circuits? (Our Results)

- There is evidence that learning log depth, constant fan-in large-alphabet circuits may be computationally intractable
- **Transitively reduced** and **bounded shortcut width** circuits can be learned in time polynomial in the number of wires and the alphabet size.
- We can approximately learn bounded shortcut-width analog circuits that satisfy a Lipshitz condition.
- We also consider learning with counterexamples

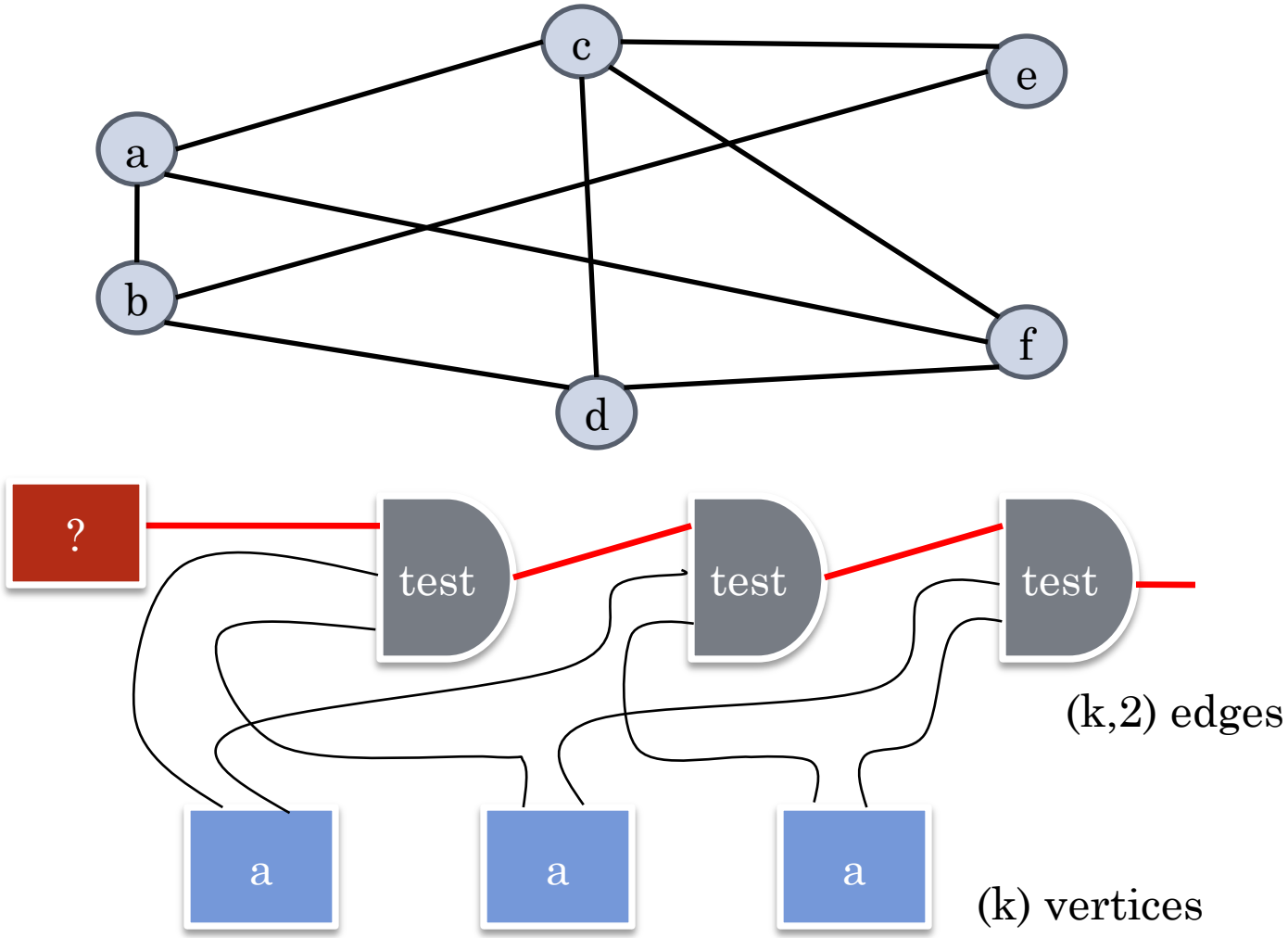
Hardness of Learning Large Alphabet Circuits

- Consider the problem on input (G,k) of telling whether the graph G on n vertices has a clique of size k
- We give a reduction that turns a large-alphabet circuit learning algorithm into a clique tester

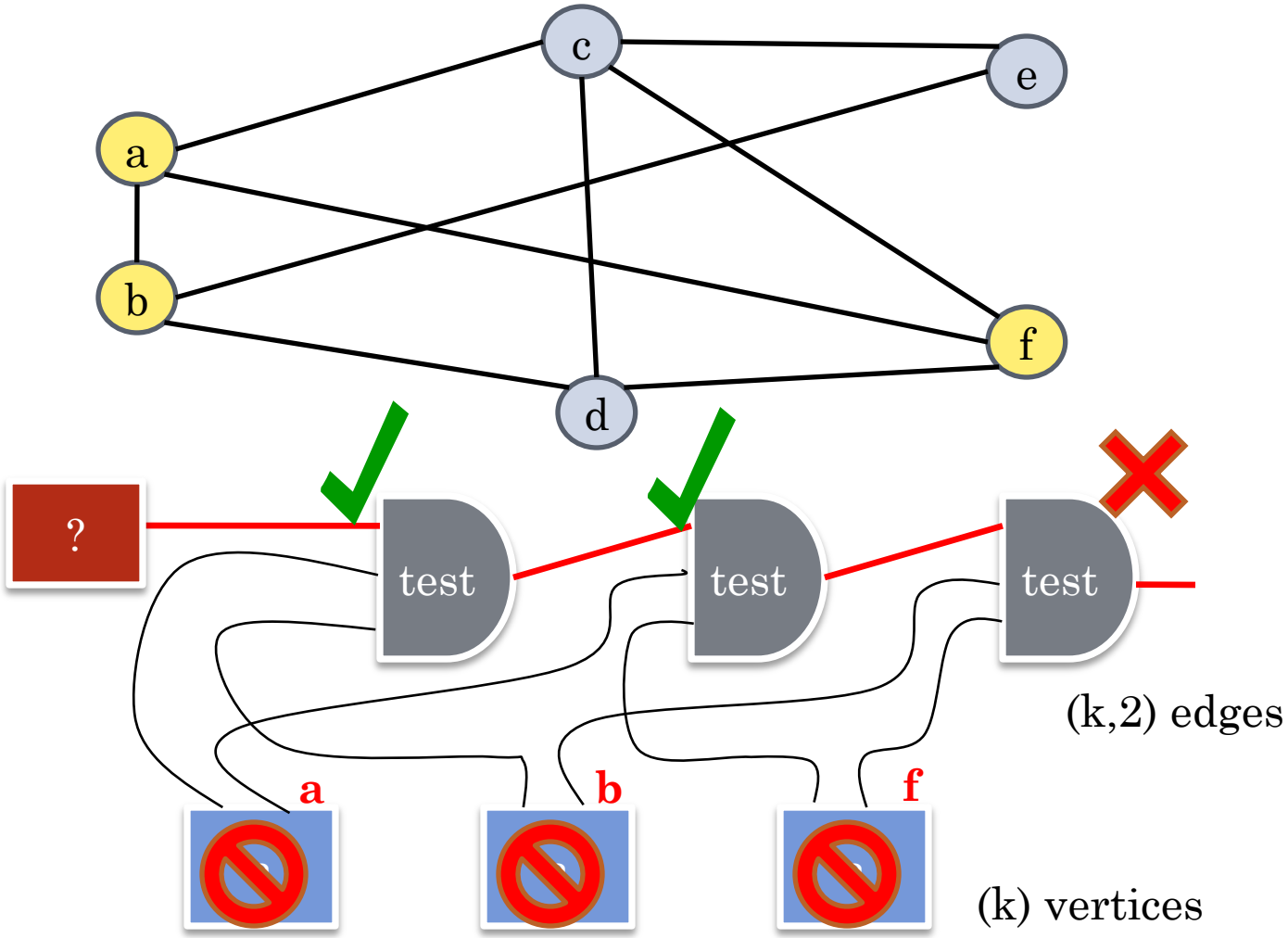




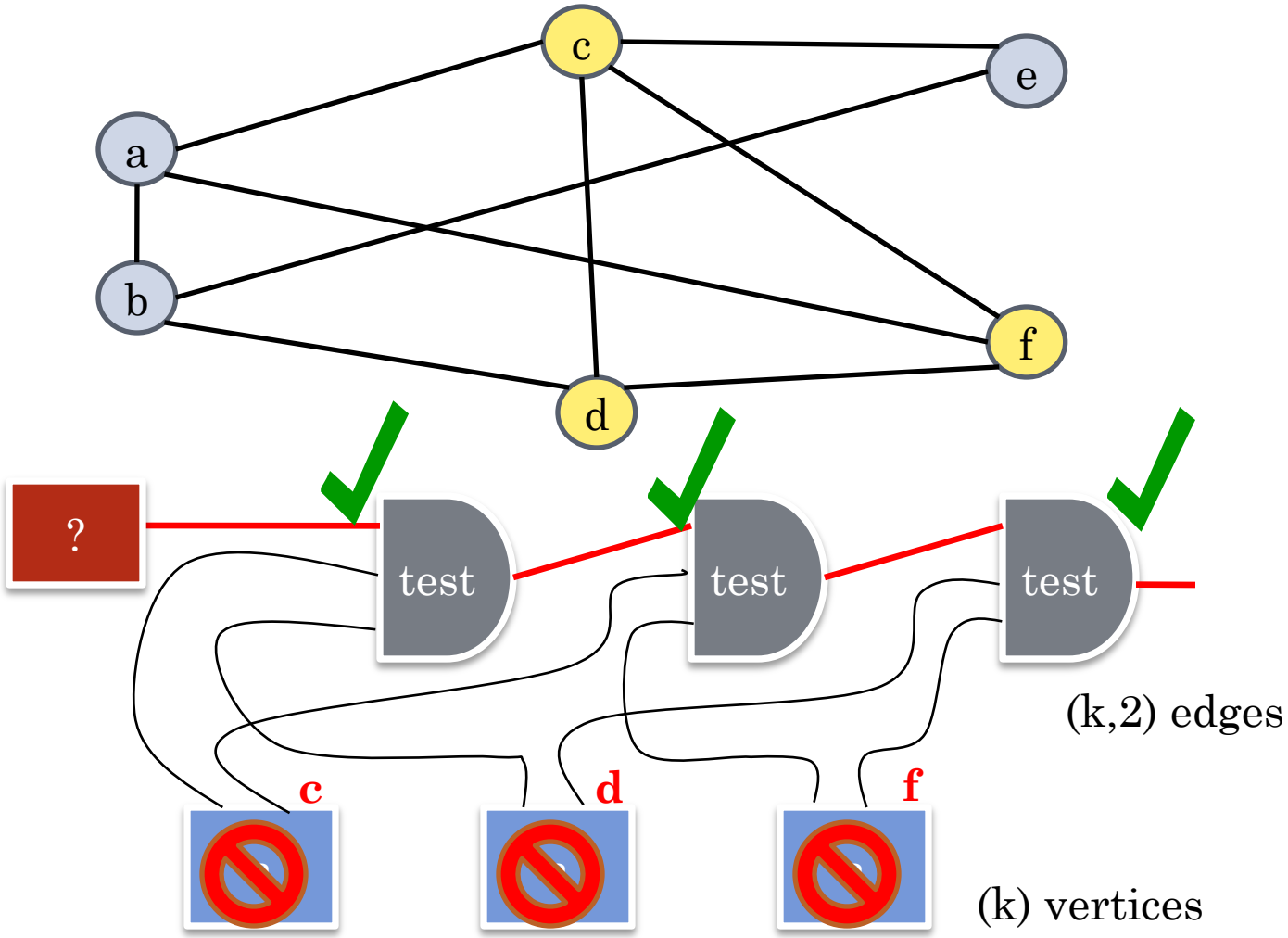
Reducing the Clique Problem to Circuit Learning



Reducing the Clique Problem to Circuit Learning



Reducing the Clique Problem to Circuit Learning



Hardness of Learning Circuits of Unrestricted Topology

- The clique problem is complete for the **parameterized complexity class** W[1]
  - There is no known algorithm for the clique problem that runs in time $f(k)n^c$ (and we believe one doesn't exist)

- **Theorem** **An algorithm for learning circuits polynomial in the number of wires and alphabet size would imply fixed parameter tractability for all problems in W[1]**

To Compare with the Boolean Case

Boolean Circuits [AACW '06]:

Depth	Fan-in	Gates	Learnability
Log	Constant	Arbitrary	Poly-time

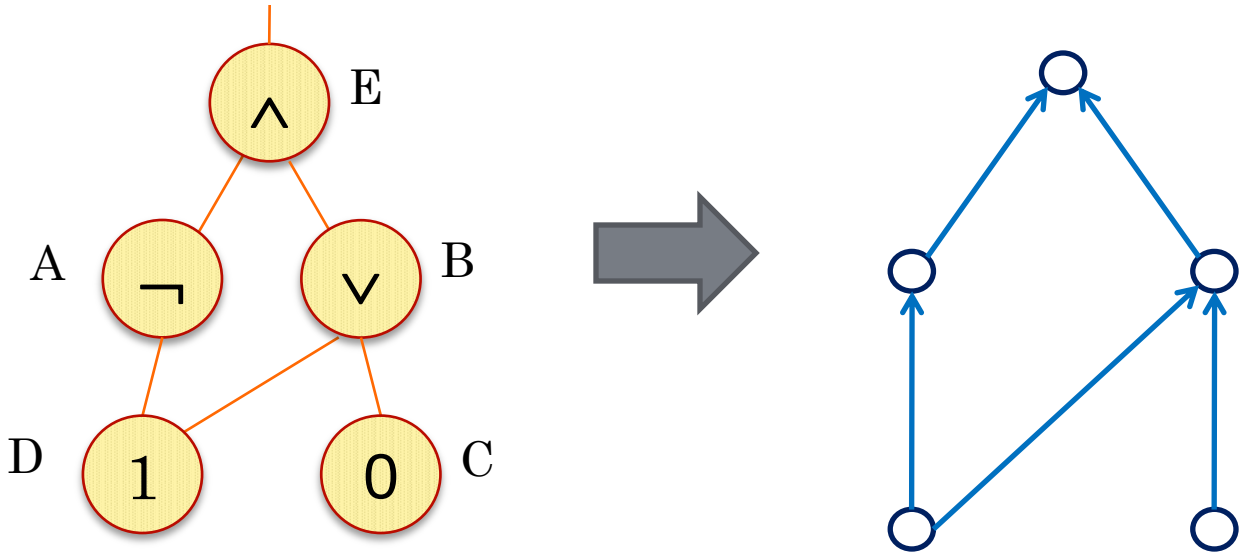
Large Alphabet Circuits:

Depth	Fan-in	Gates	Learnability
Log	Constant	Arbitrary	W[1] Hard

This motivates looking at classes of large-alphabet circuits with restricted topology

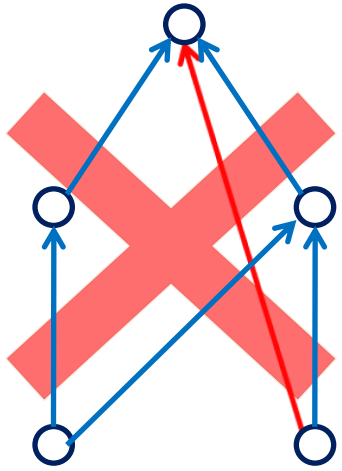
A Circuit's Underlying Graph

We only consider circuits whose simple, connected, directed graphs are acyclic.



Transitively Reduced Circuits

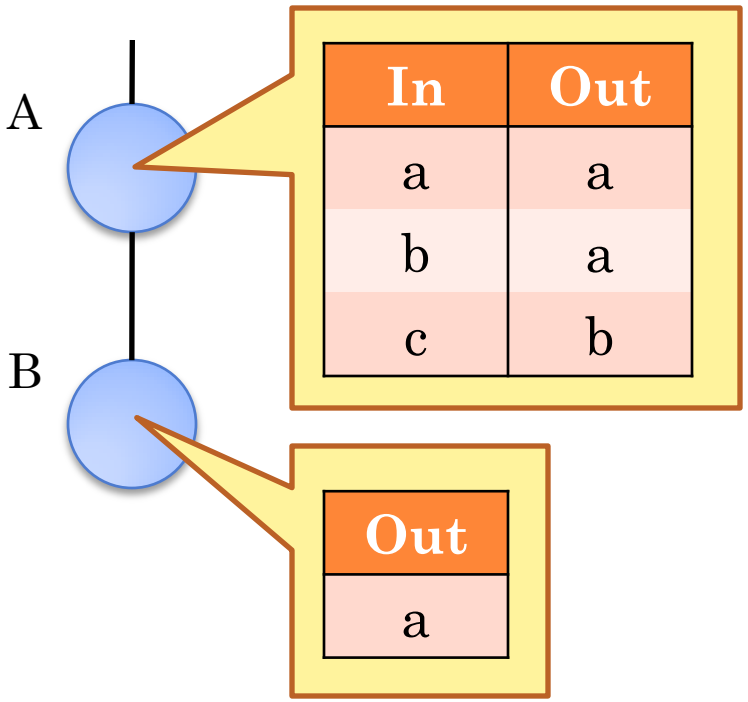
A circuit is transitively reduced if its underlying directed graph has no shortcuts. If (u,v) is an edge and there is a path of length ≥ 2 from u to v, then (u,v) is a **shortcut edge**



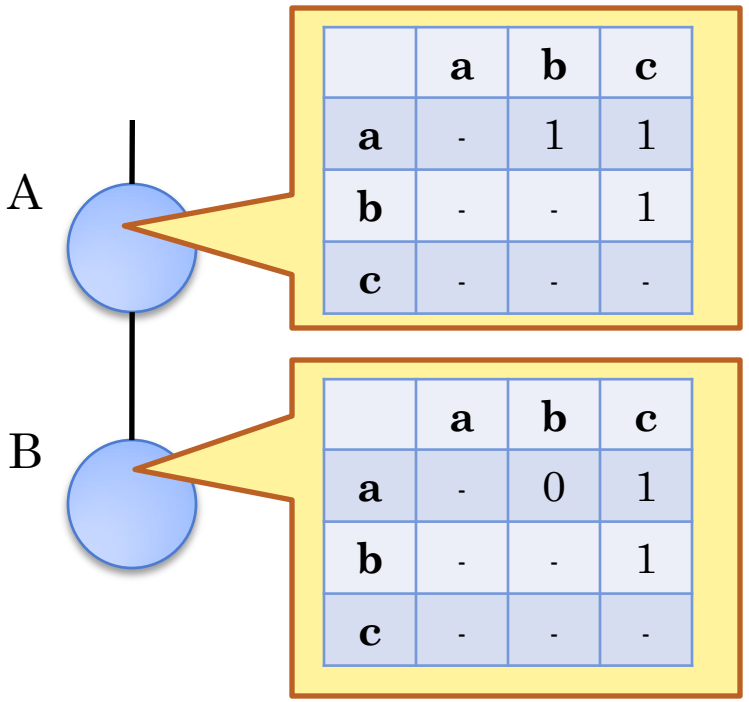
Distinguishing Tables

- For each wire w, we keep a distinguishing table. A 1 entry in $T_w(\sigma,\tau)$ means alphabet values σ and τ are **distinguishable**. For each 1 entry we keep a corresponding **distinguishing path** and a "processed bit."

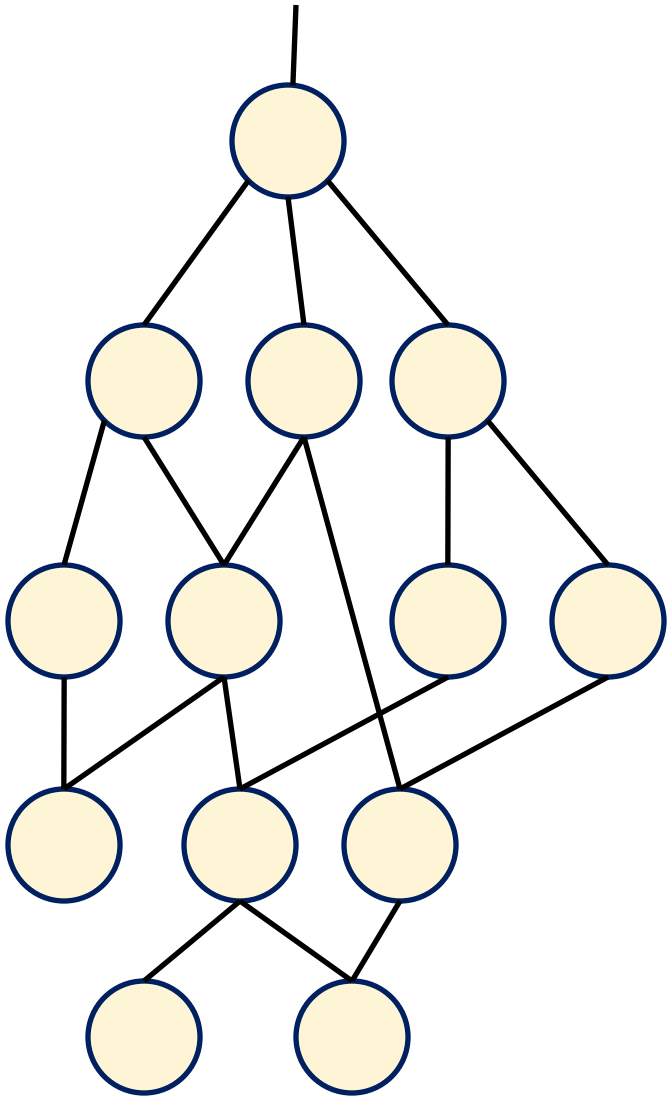
Gate functions



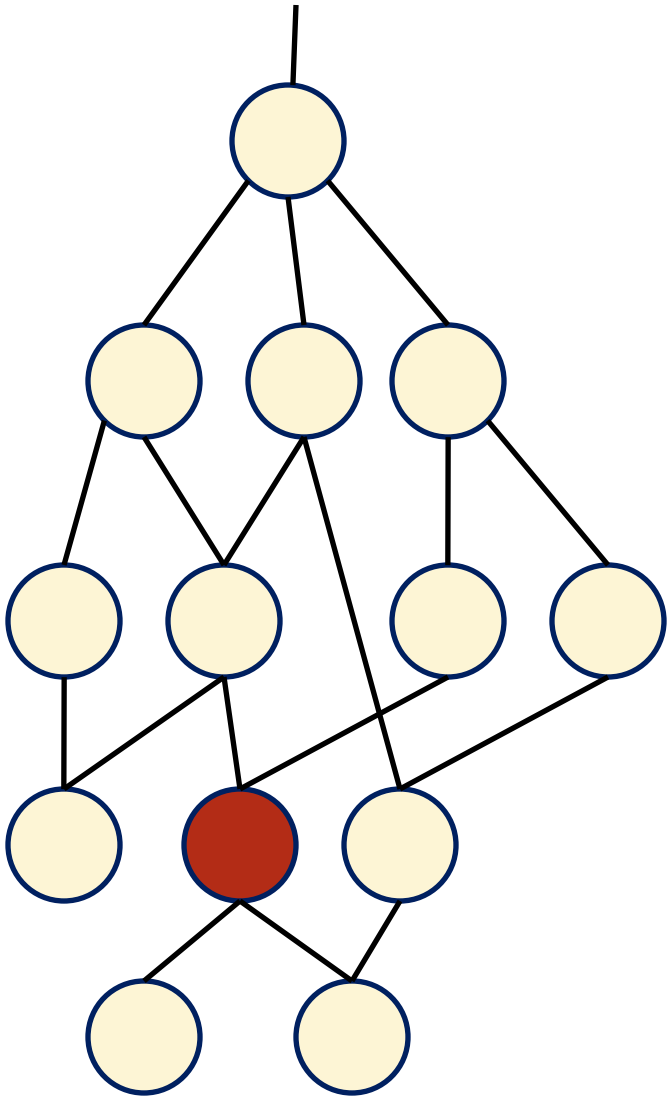
Distinguishing Tables



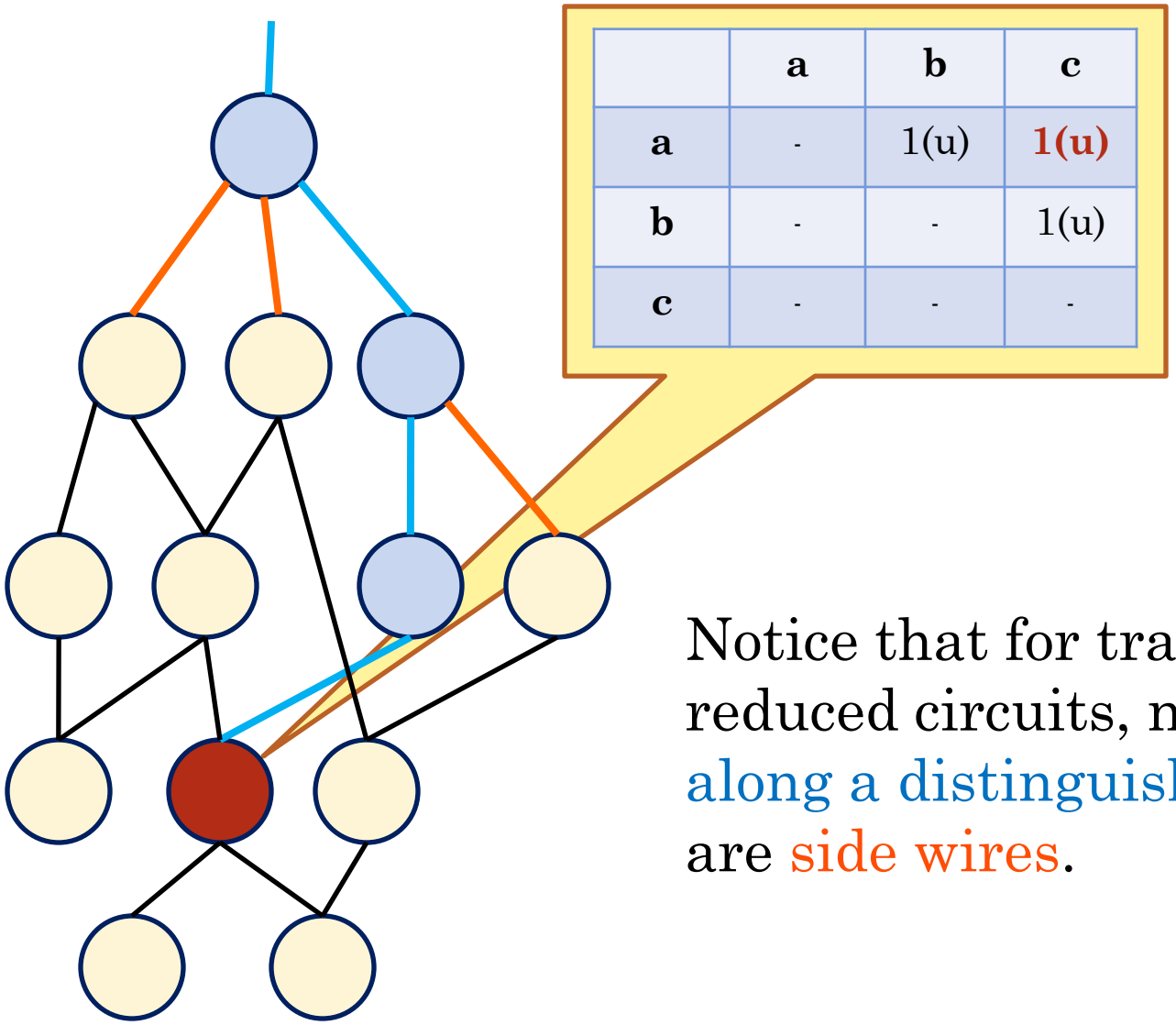
Distinguishing Paths



Distinguishing Paths



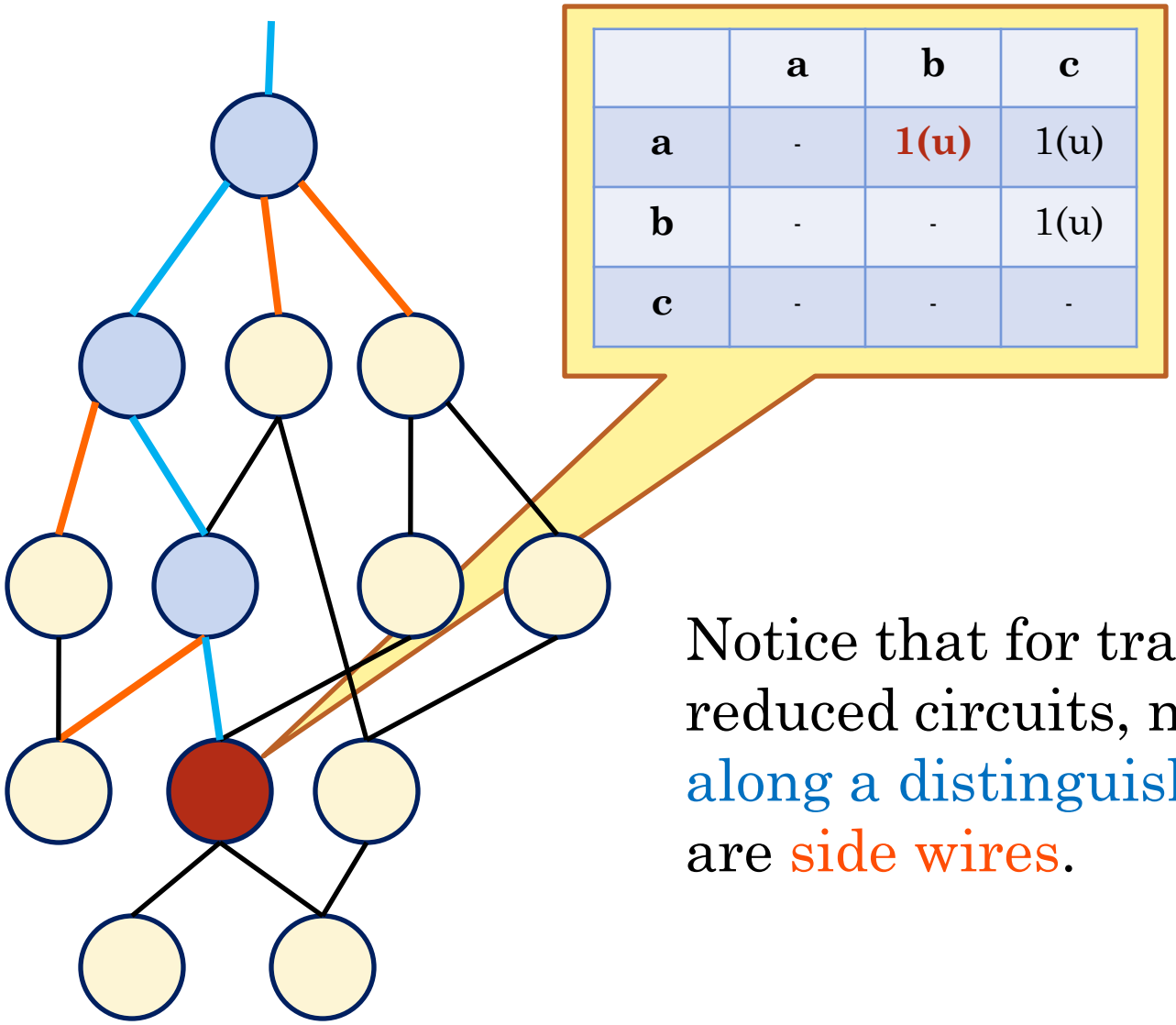
Distinguishing Paths



	a	b	c
a	-	1(u)	**1(u)**
b	-	-	1(u)
c	-	-	-

Notice that for transitively reduced circuits, no wires along a distinguishing path are side wires.

Distinguishing Paths



	a	b	c
a	-	**1(u)**	1(u)
b	-	-	1(u)
c	-	-	-

Notice that for transitively reduced circuits, no wires along a distinguishing path are side wires.

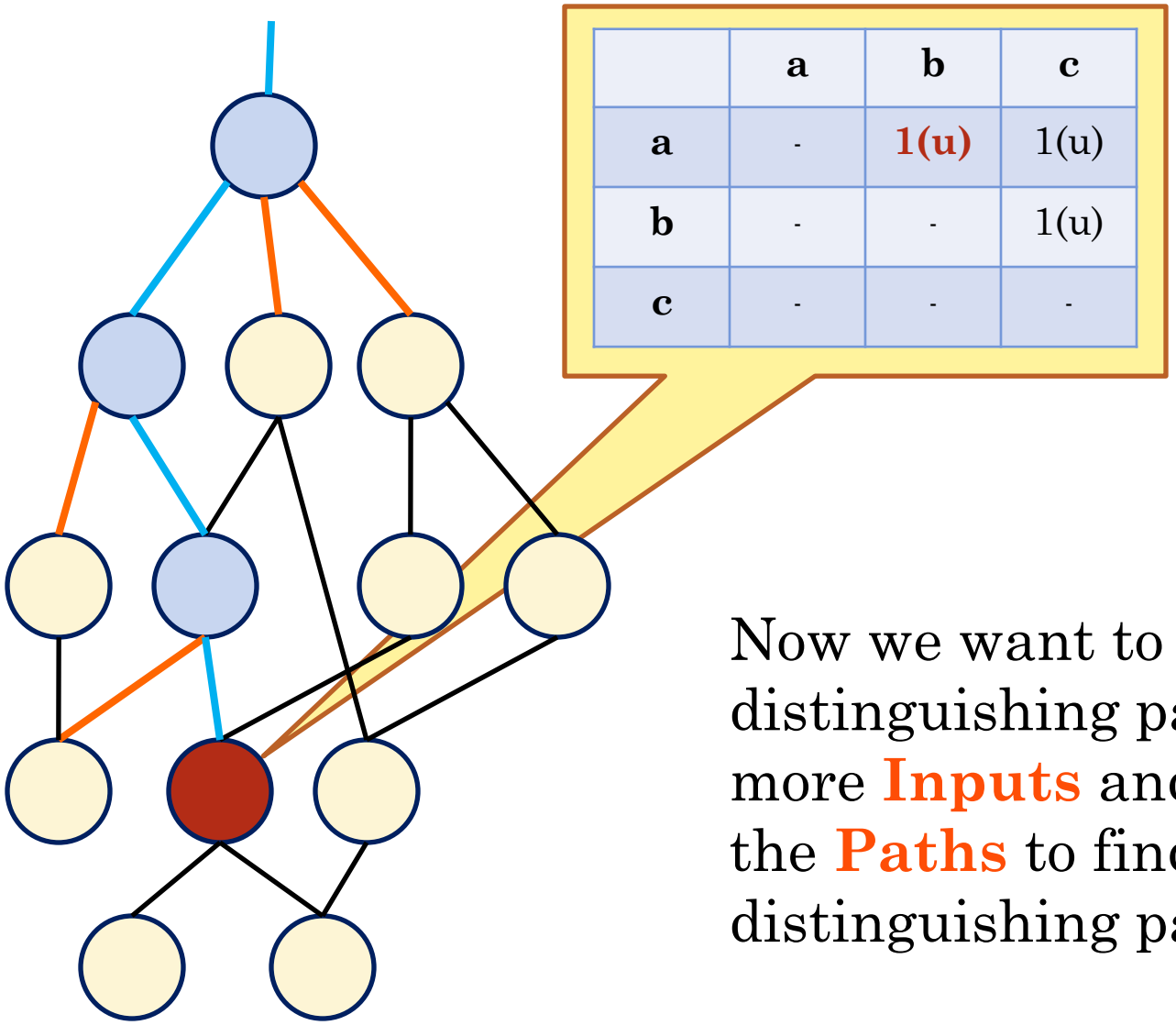
The Distinguishing Paths Algorithm (Outline)

- For the output wire w_n, we initialize T_{w_n} with all values initialized to 1, marked unprocessed. The rest of the tables are initialized to all 0's.

- While there are unprocessed 1 entries, pick one and run **Find Inputs** and **Extend Paths**.

- Finally, **Reconstruct the Circuit**.

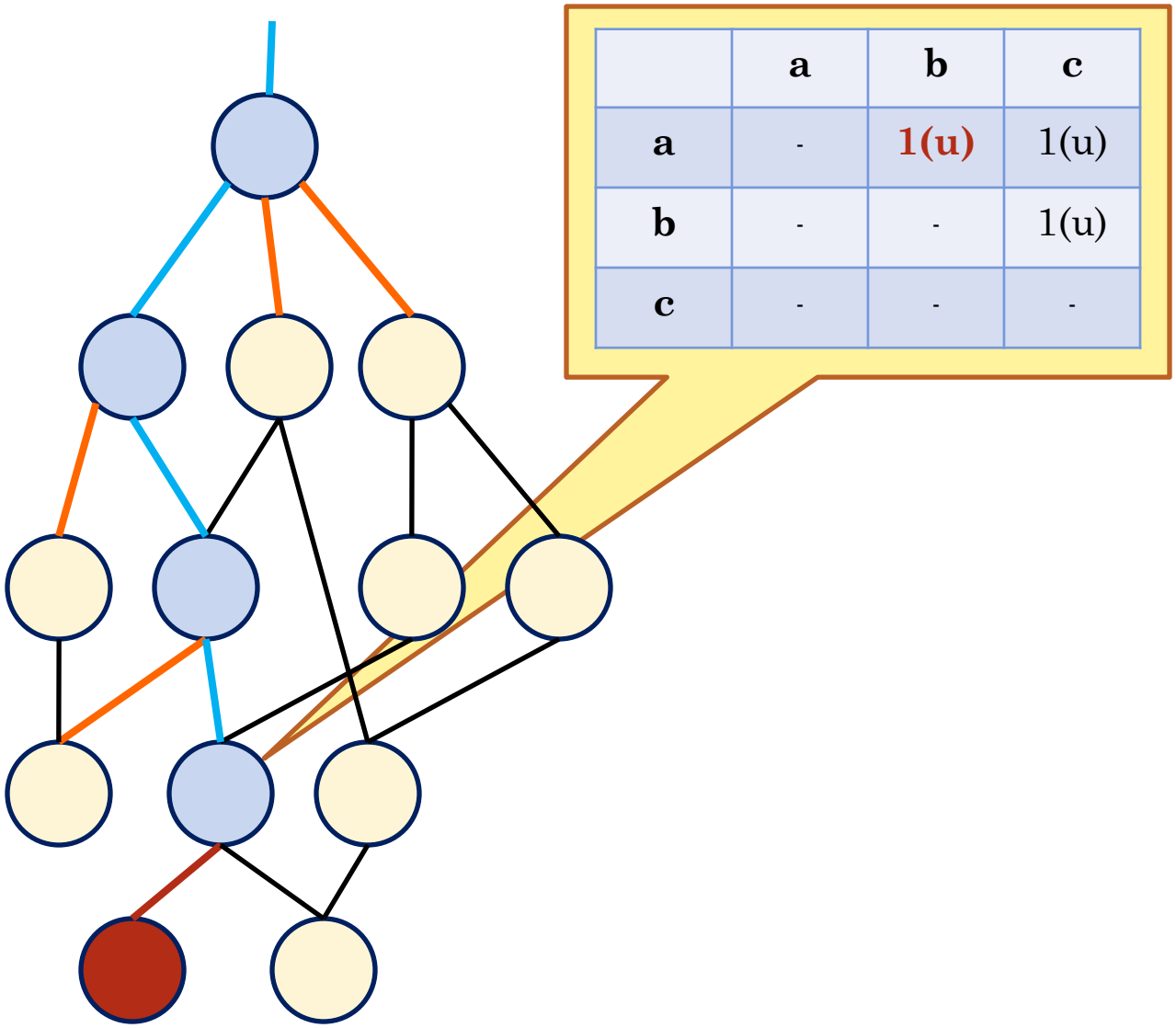
Find Inputs and Extend Paths



	a	b	c
a	-	**1(u)**	1(u)
b	-	-	1(u)
c	-	-	-

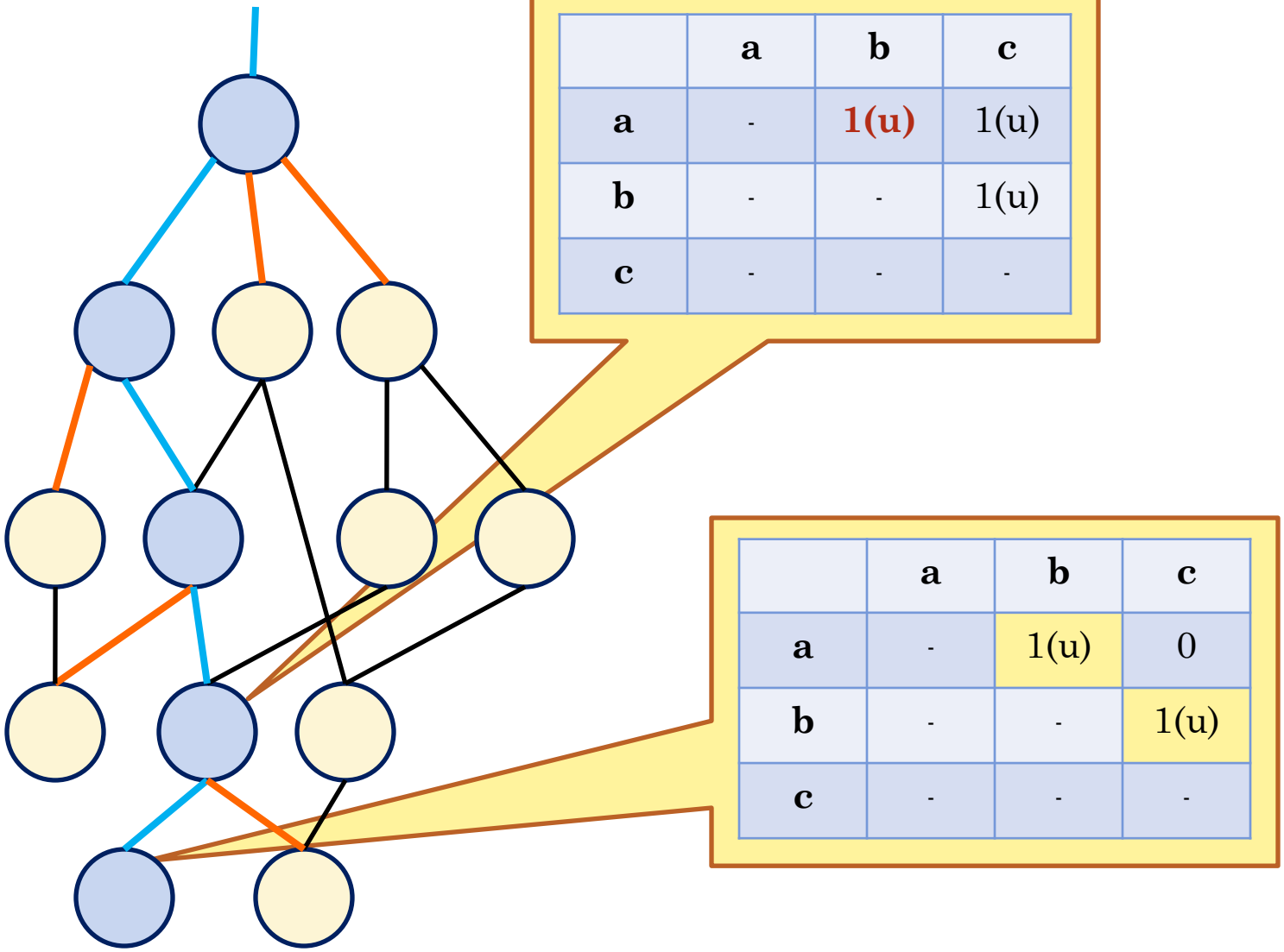
Now we want to use this distinguishing path to **Find** more **Inputs** and **Extend** the **Paths** to find new distinguishing paths.

Find Inputs and Extend Paths



	a	b	c
a	-	**1(u)**	1(u)
b	-	-	1(u)
c	-	-	-

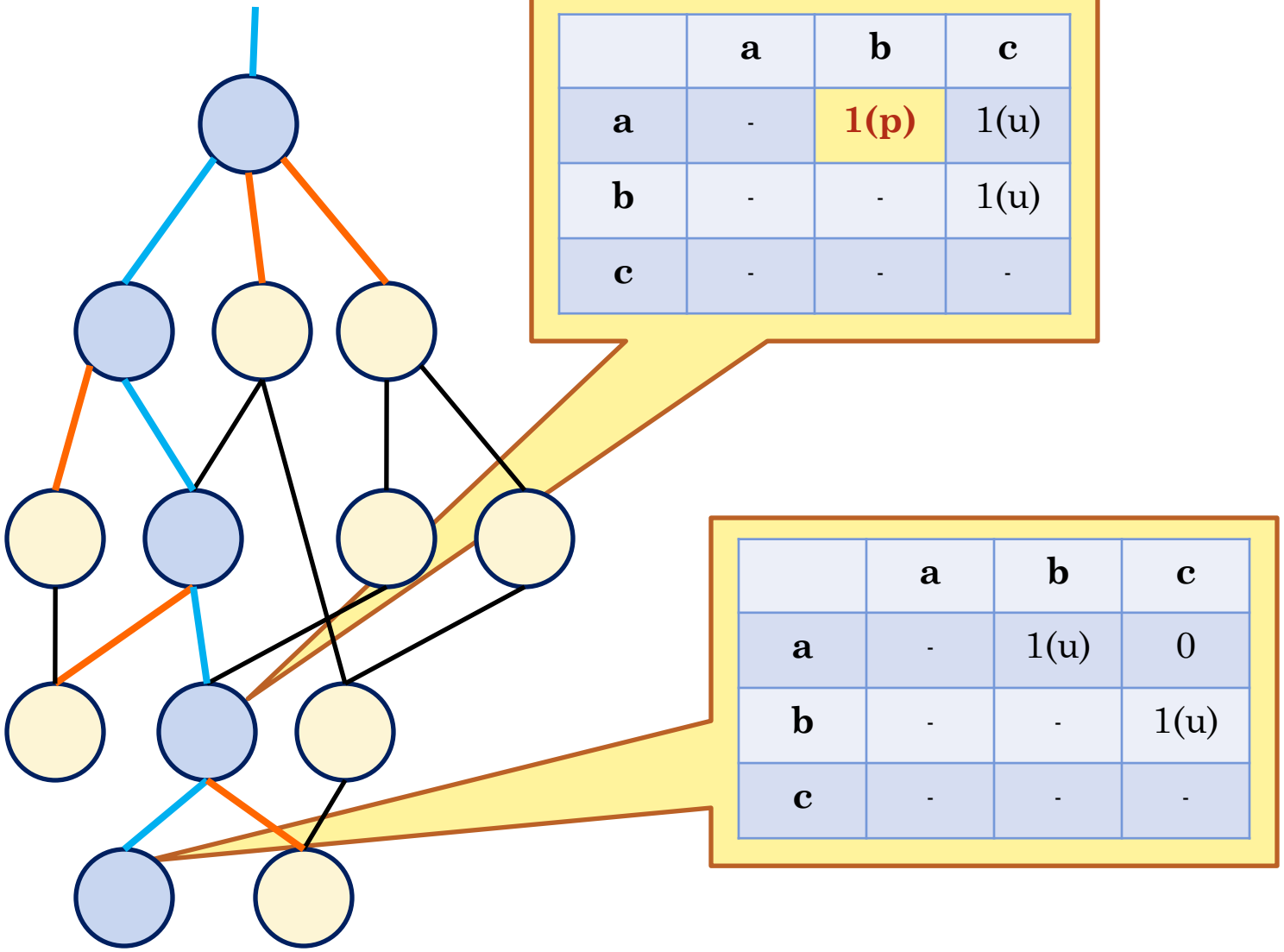
Find Inputs and Extend Paths



	a	b	c
a	-	**1(u)**	1(u)
b	-	-	1(u)
c	-	-	-

	a	b	c
a	-	1(u)	0
b	-	-	1(u)
c	-	-	-

Find Inputs and Extend Paths

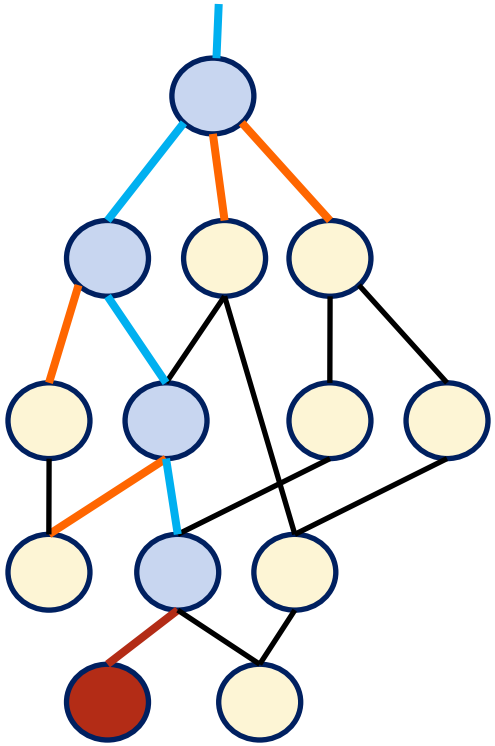


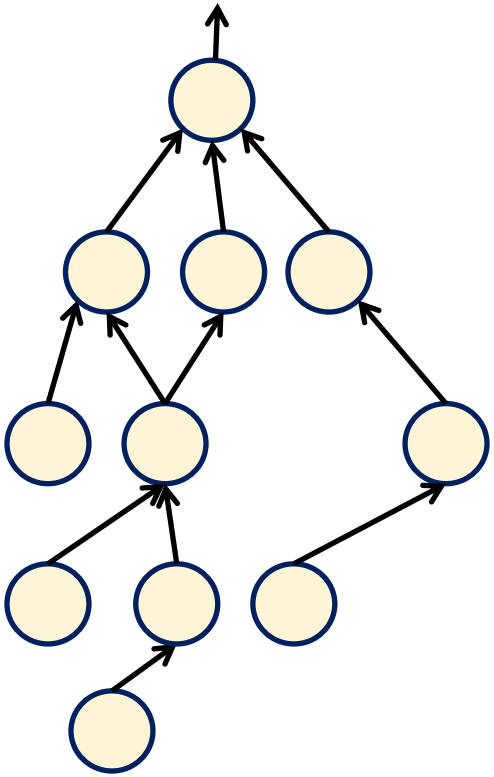
	a	b	c
a	-	**1(p)**	1(u)
b	-	-	1(u)
c	-	-	-

	a	b	c
a	-	1(u)	0
b	-	-	1(u)
c	-	-	-

Reconstructing Transitively Reduced Circuits

- We keep a separate directed graph G to reconstruct the graph of the circuit.





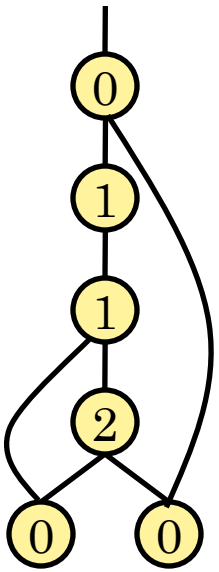
Reconstructing Transitively Reduced Circuits

- We keep a separate directed graph G to reconstruct the graph of the circuit.

- **Theorem** **The complete distinguishing tables and G are enough to construct a circuit behaviorally equivalent to the target circuit in polynomial time and $O(n^{2k+1}s^{2k+2})$ queries.**

Bounded Shortcut Width

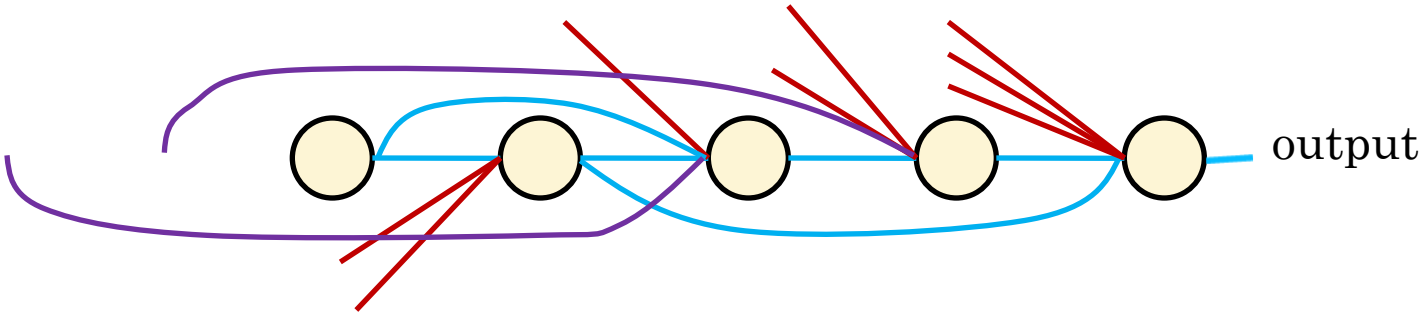
- Bounded shortcut width is a generalization of transitive reduction.
- The shortcut width of a wire w_i is the number of wires w_j such that w_j is both an ancestor of w_i and an input of a descendent of w_i.
  - Transitively reduced circuits have shortcut width 0.



The bounded shortcut width of a circuit is the maximum shortcut width of any output-connected wire in the circuit.

Distinguishing Paths with Shortcuts

- We generalize the definition of a distinguishing path to a **distinguishing path with shortcuts**.
  - These are made of **path wires**, **side wires**, and **cut wires**.



- We also generalize the notion of distinguishing tables to include cut wires.

Learning Circuits of Bounded Shortcut Width

- When all 1 entries in the generalized distinguishing tables are processed, the tables and graph G we can create a set of sufficient experiments for **CircuitBuilder** of [AACW '06].

- **Theorem** **The Shortcuts Algorithm learns the class of circuits having n wires, alphabet size s, fan-in bound k, and shortcut width bounded by b, using $ns^{O(k+b)}$ value injection queries and time polynomial in the number of queries.**

Learning Analog Circuits

- An analog circuit is a circuit for which $\Sigma = [0,1]$.
- ε-equivalence: If $d(C(e),C'(e)) \leq \varepsilon$ for every experiment e, then C and C' are ε-equivalent.
- We can discretize analog circuits that satisfy a Lipshitz condition and use our large-alphabet learning algorithms on them.
- **Theorem** **There exists a polynomial time algorithm that learns up to ε-equivalence any analog circuit of n wires, depth log(n), constant fan-in, Lipshitz gate functions, and shortcut width bounded by a constant.**

Learning with Counterexamples

- We also consider the framework where we have both value injection queries and counterexamples.
- In a counterexample query, the algorithm proposes a hypothesis C and receives as an answer either that C is exactly equivalent to the target circuit or an experiment e such that $C(e) \neq C^*(e)$

- **Theorem** **Circuits whose gates are polynomial time learnable with counterexamples are learnable in polynomial time with experiments and counterexamples.**

Summary and Discussion

- We give algorithms for learning large alphabet and analog circuits and matching lower bounds.
- The learnability of large alphabet circuits seems to depend on their shortcut width – this is quite different from the small alphabet case.
- It would be interesting to try to extend this framework to Bayesian networks (or probabilistic circuits) and other classes.