HW1:
SIS & ABC Report

# Introduction of SIS | ABC

Speaker: 唐梧遷

Keyword: *AIG, SIS, SAT Solver*
*Mar.*31, 2025

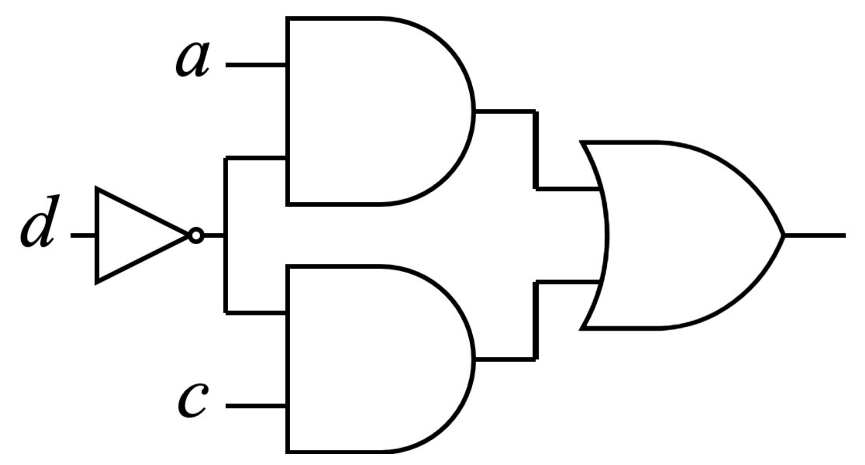
towne.cpp@gmail.com

Topic: Logic Optimization

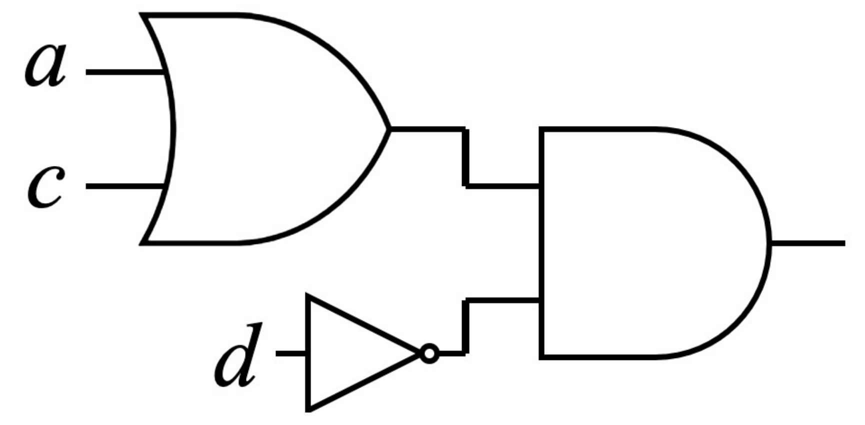
Goal

Find an equivalent representation of a logic circuit that minimizes the **area** of the circuit under some **delay constraints**.

$$f_1 = a \cdot d' + c \cdot d' \xleftrightarrow{\text{Functional equivalent}} f_2 = (a + c) \cdot d'$$



Better Area

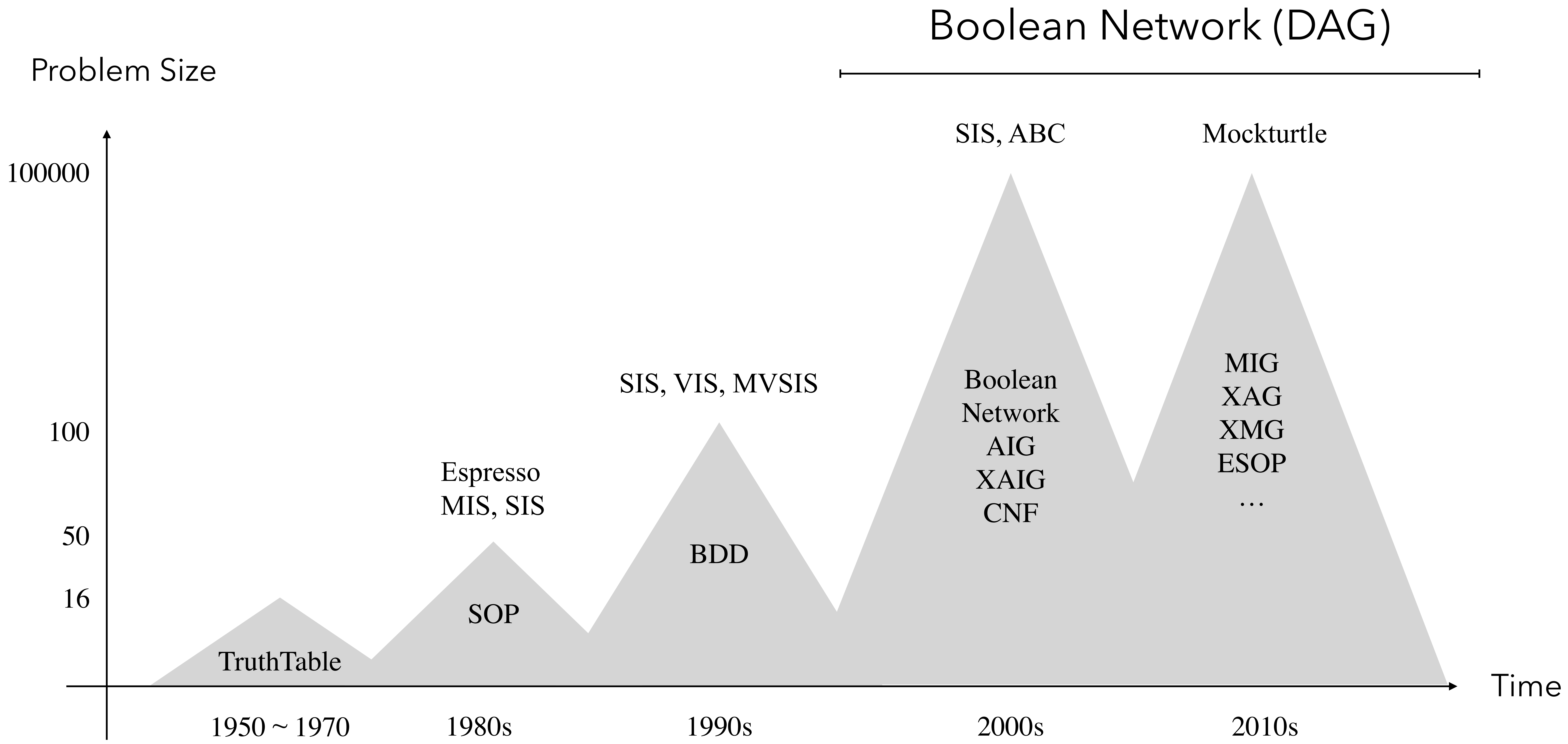


Operator[1]: 2 And + 1 OR

Operator[1]: 1 And + 1 OR

1. Inverters are ignored in logic area calculations for simplicity and negligible impact.

Historical Perspective



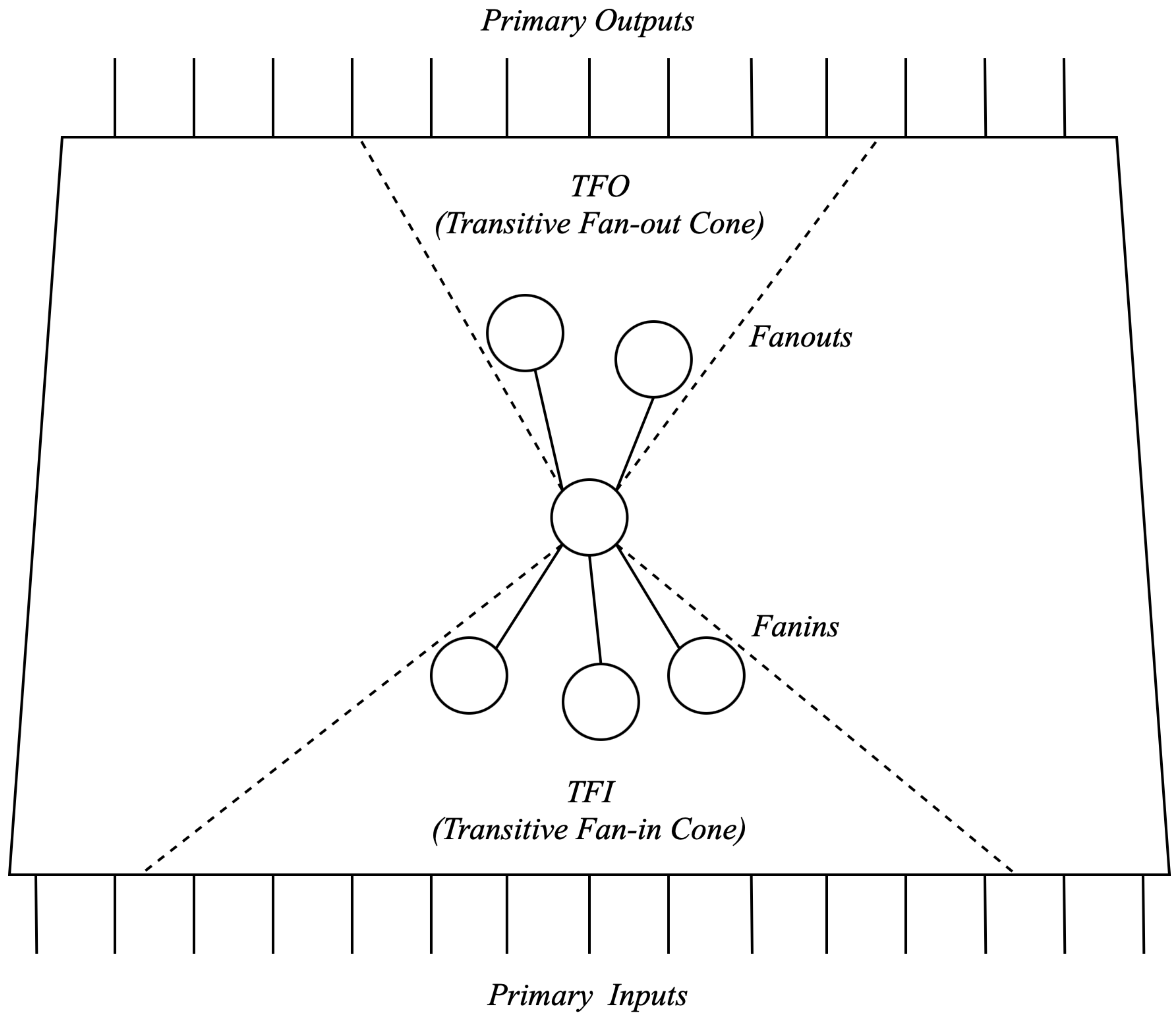
Terminalogy

Logic function (F = ab +a′d):

- Variables (a, b, d)
- Literals (a, a', b, d)
- Minterms ($abd,\ abd', a'bd, a'b'd$)
- Cube: set of Minterms ($ab, a'd$)

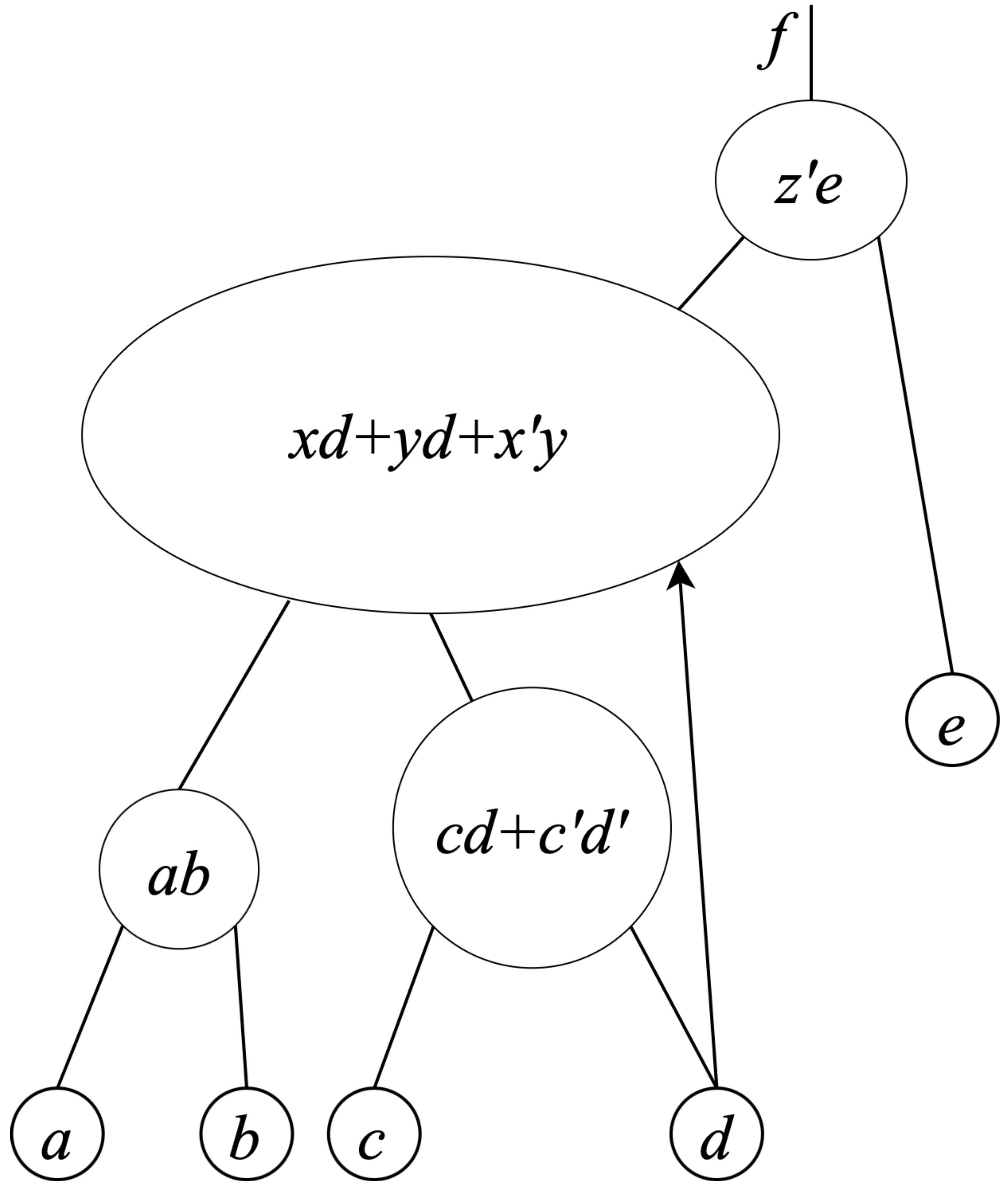
Logic network:

- Primary inputs/outputs (PI/PO)
- Logic nodes
- Fanins/Fanouts
- Transitive fanin/fanout cone (TFI/TFO)
- Cut and Window (define in ABC section)



SIS: A System for Sequential Circuit Synthesis

Representation of SIS



(a) Boolean network in SIS

Network manipulation (algebraic):

- Elimination
- Factoring/Decomposition

Node minimization:

- Espresso
- Don't cares computed using **BDD**
- Resubstitution

Technology mapping

- Tree based

How to optimize circuit

```
sis> read_blif mcnc/mlex/dalu.blif
sis> print_stats
dalu          pi=75   po=16   nodes=1131      latches= 0
lits(sop)=3588
sis> decomp
sis> print_stats
dalu          pi=75   po=16   nodes=1428      latches= 0
lits(sop)=3364
sis> full_simplify
sis> print_stats
dalu          pi=75   po=16   nodes=1428      latches= 0
lits(sop)=2828
sis> collapse
sis> print_stats
dalu          pi=75   po=16   nodes= 16       latches= 0
lits(sop)=24902
sis> decomp -g
sis> print_stats
dalu          pi=75   po=16   nodes=489       latches= 0
lits(sop)=2311
```

script.rugged

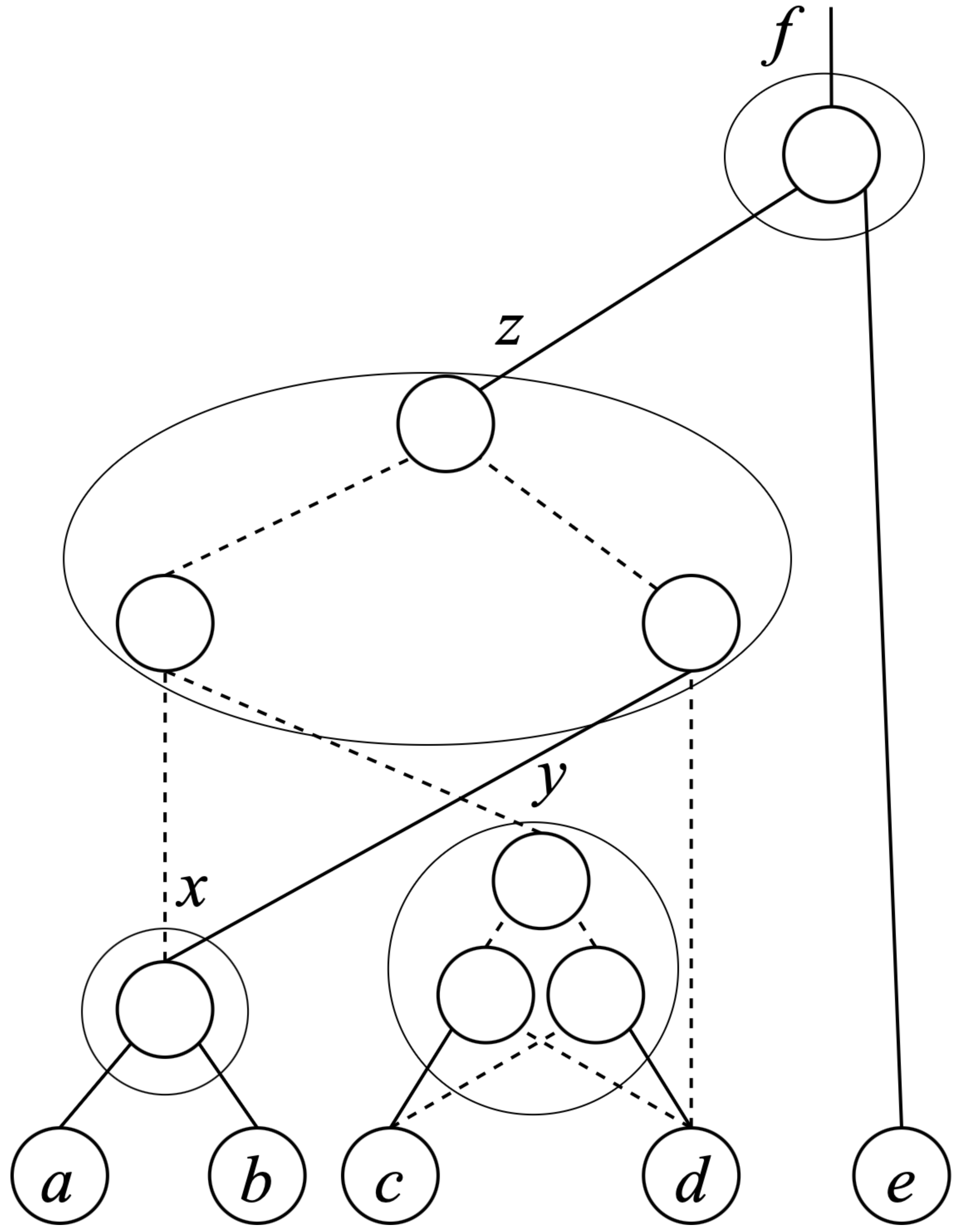
```
sweep; eliminate -1
simplify -m nocomp
eliminate -1
sweep;
eliminate 5
simplify -m nocomp
resub -a
fx
resub -a; sweep
eliminate -1; sweep
full_simplify -m nocomp
```

How to optimize circuit using script.rugged

```
sis> read_blif mcnc/mlex/dalu.blif
sis> print_stats
dalu          pi=75   po=16   nodes=1131      latches= 0
lits(sop)=3588
sis> source script.rugged
sis> print_stats
dalu          pi=75   po=16   nodes=225       latches= 0
lits(sop)=1606
```

ABC: A System for Sequential Synthesis and Verification

Representation of ABC



(a) AIG in ABC, AIG is a **Boolean network** of 2-input AND nodes and inverters (dotted lines)

AIG manipulation (graph/boolean):

- Rewriting/Refactoring
- Balancing

Node minimization:

- Boolean decomposition
- Don't cares computed using **simulation** & **SAT**
- Resubstitution with don't cares

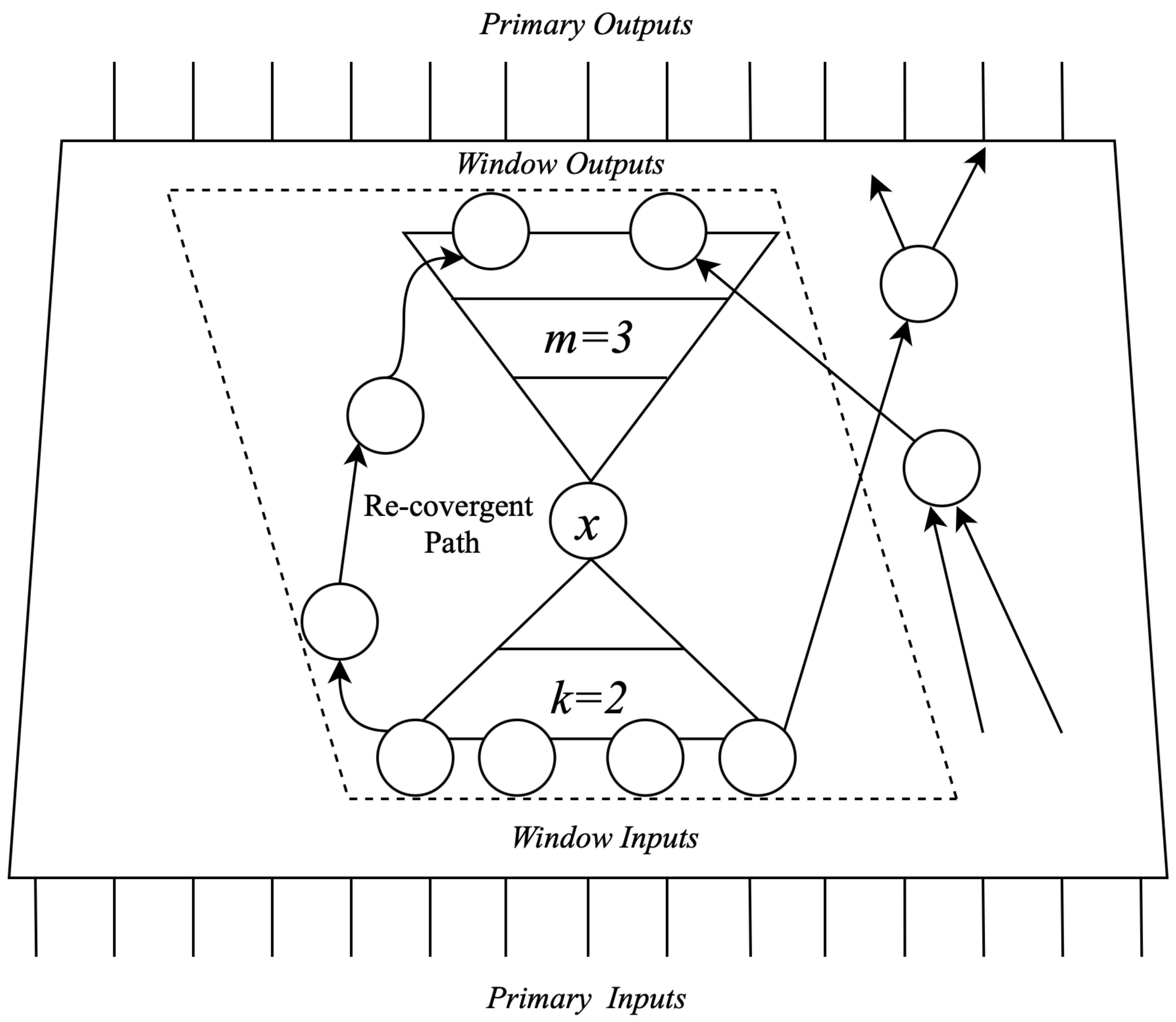
Technology mapping

- Cut based with **choice** nodes

Basic Operation unit: Window

Definition of window for node x:

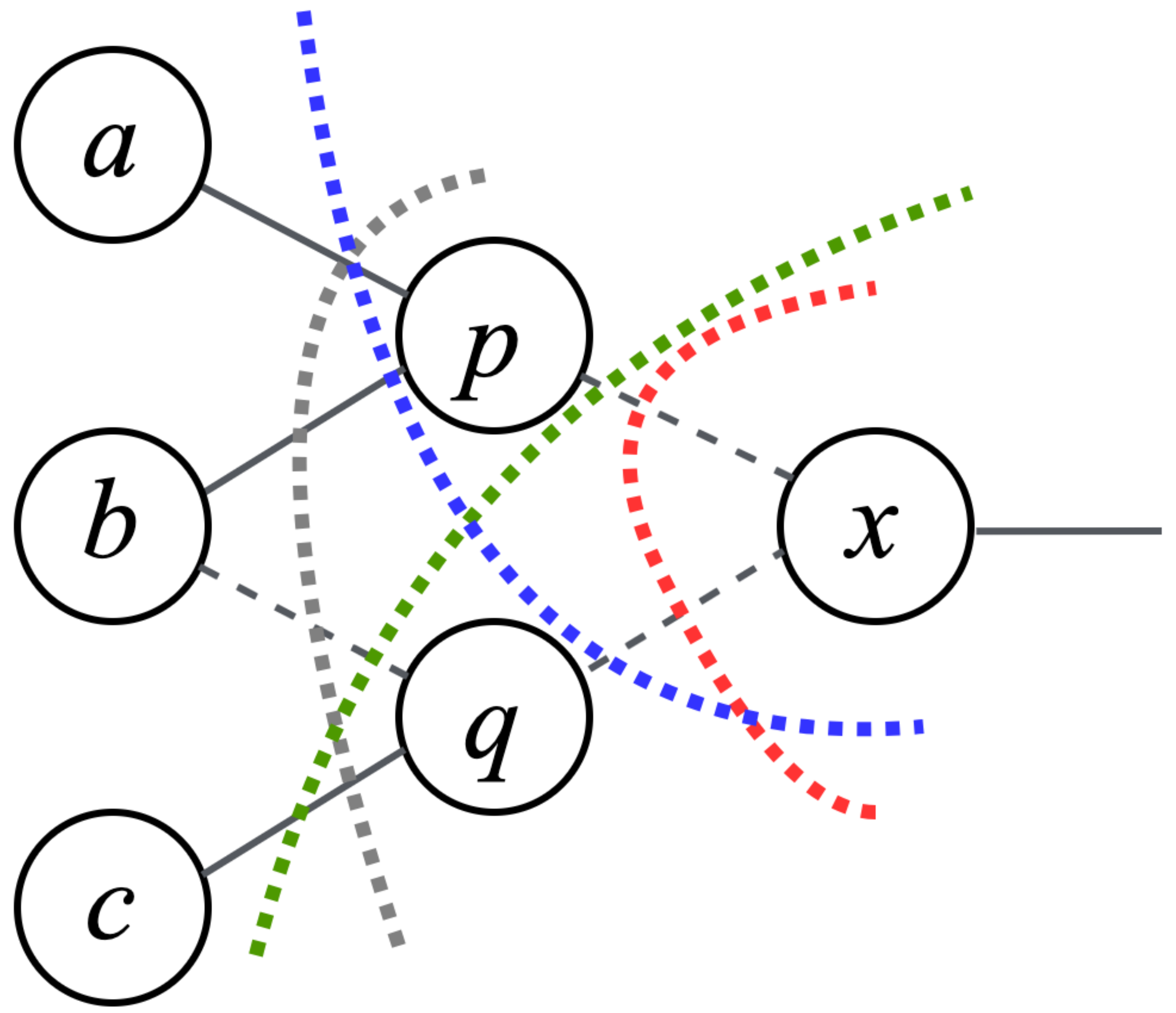
- A window of node x is the node's context, in which can operation is performed.
- It includes:
  - *k* levels of the TFI
  - *m* levels of the TFO
  - All convergent path between Window Inputs & Window Outputs
- Used in command *mfs, &mfs* (LUT Resynthesis)



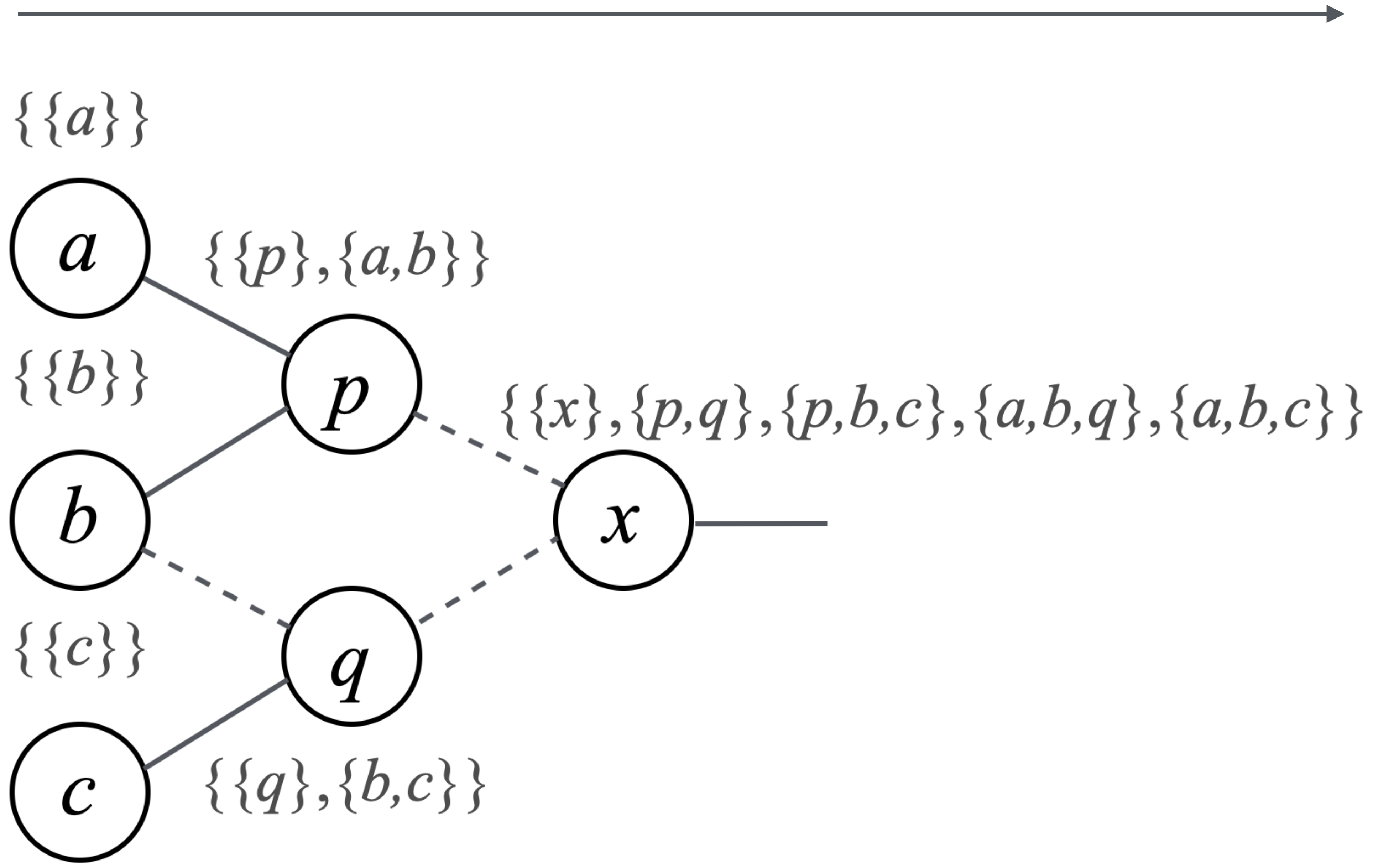
Basic Operation unit: Cut

Definition of cut for node x:

- A cut of node x in an AIG is a set of nodes that blocks all paths from PI of x to x.



Computation is done bottom-up



There are many cuts for the same node, each cut is a different SIS node.

AIG manipulation with cuts is equivalent to working on many boolean networks at the same time.

Algorithm: AIG Rewriting

Core concept of rewrite

Identify optimal circuit region, **replace** it with optimal equivalent logic.

Definition of "circuit region"

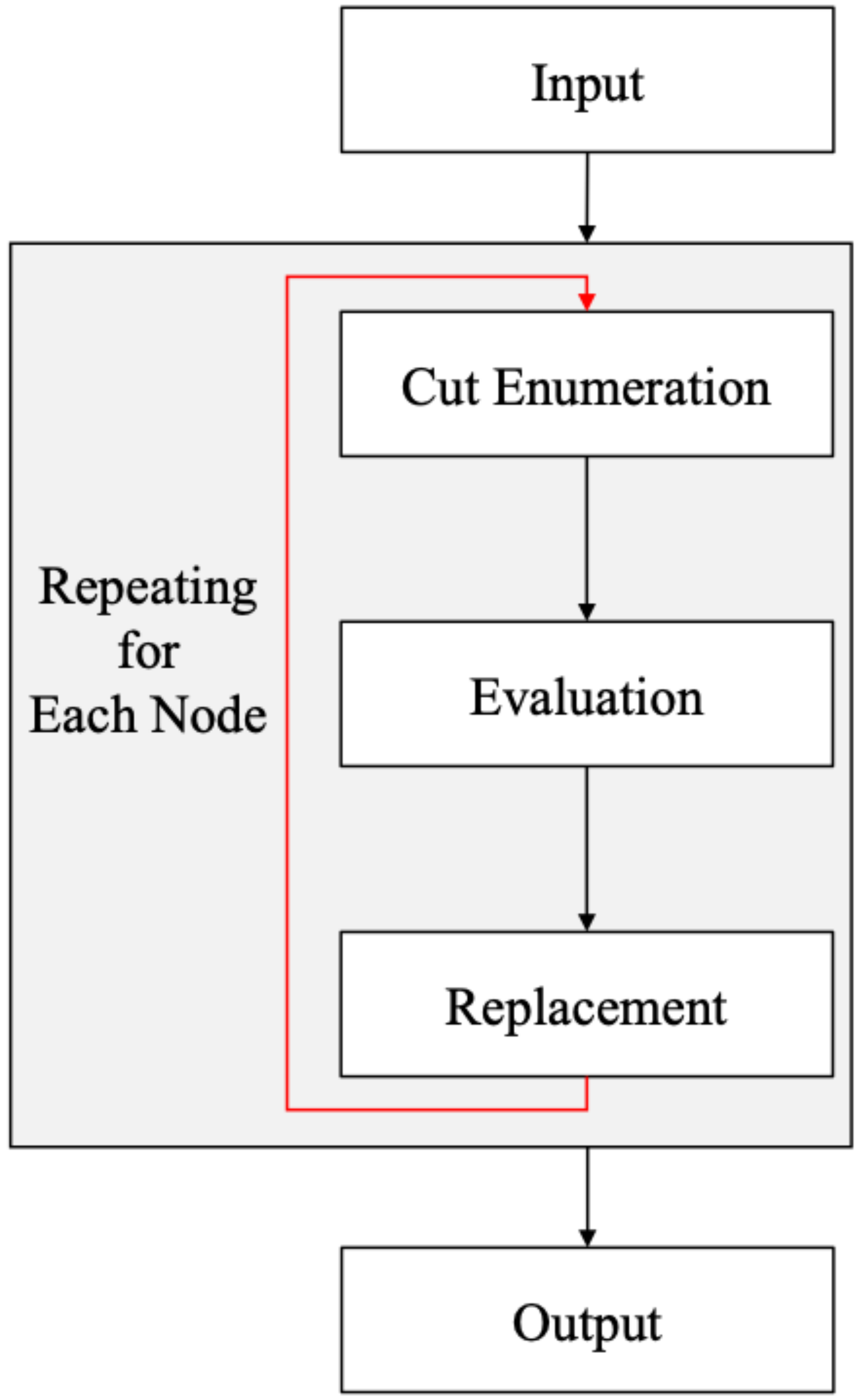
It is the *k*-feasible cut of an AIG node.

- A *k*-feasible cut represents a *k*-input boolean expression to be replaced.
- In ABC, by default, *k*=4.

Algorithm

For each node x in AIG (topological order):

1. **Cut Enumeration:** Generate ≤ 8 (by default) cut of node x.
2. **Evaluation:** For each cut c of node x, identify optimal circuit region and corresponding optimal equivalent logic.
3. **Replacement**

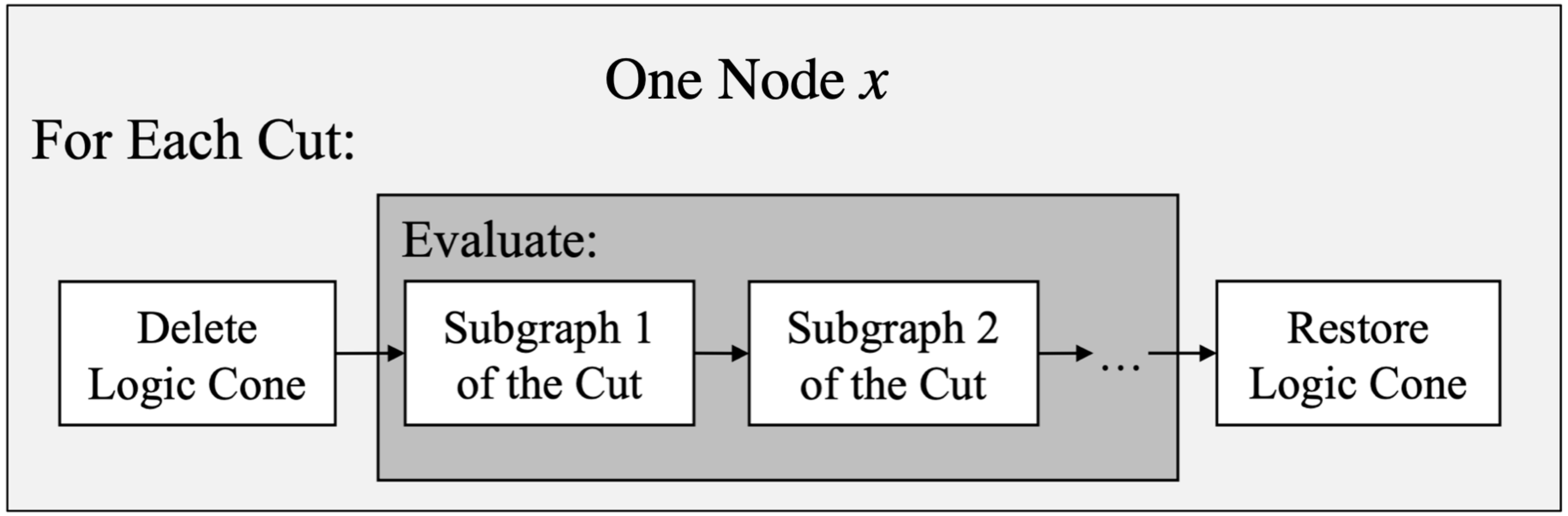


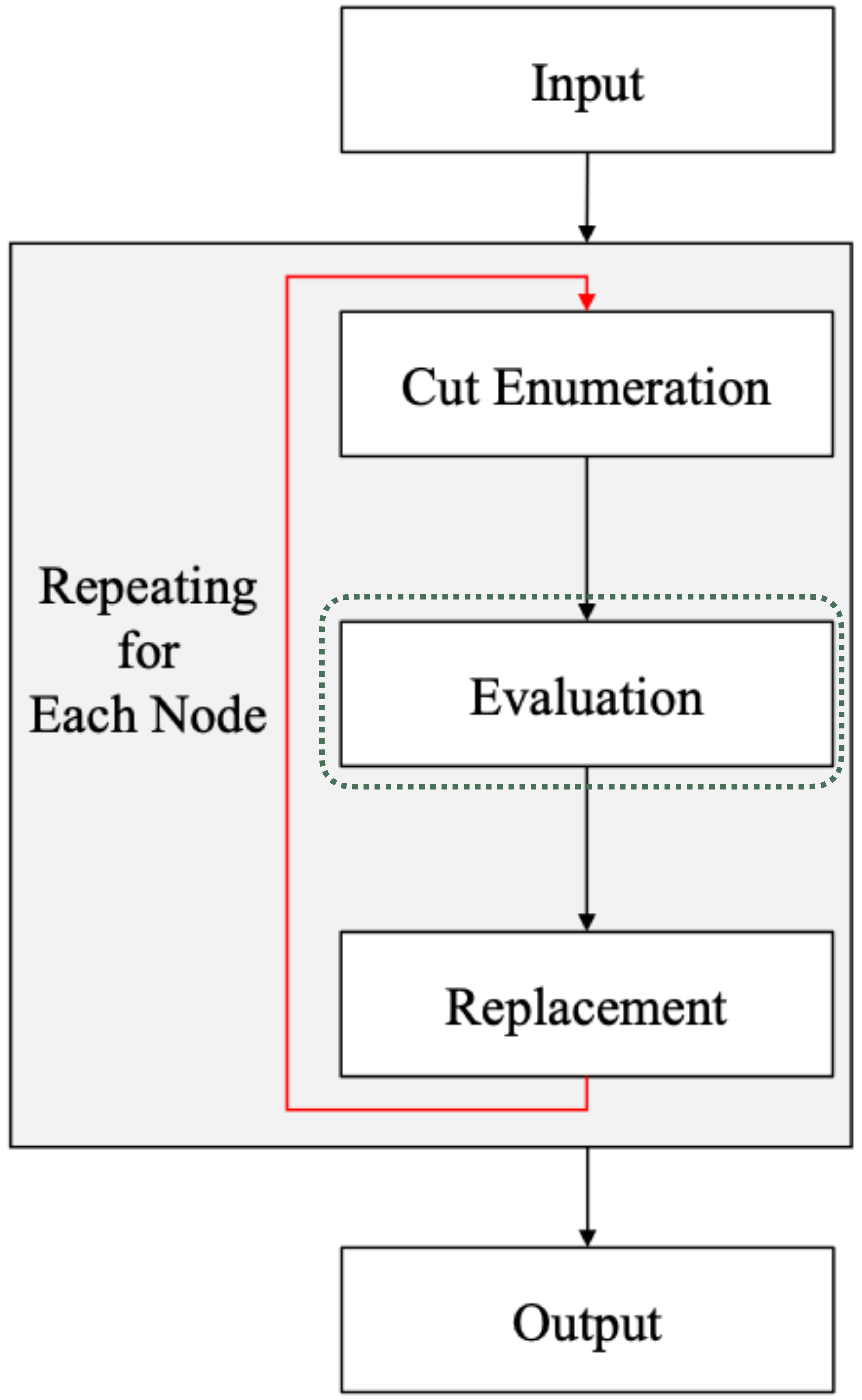
(a) AIG Rewriting

AIG Rewriting: Evaluation

Evaluation: For node x, identify optimal cut c_{opt} and corresponding optimal equivalent logic.

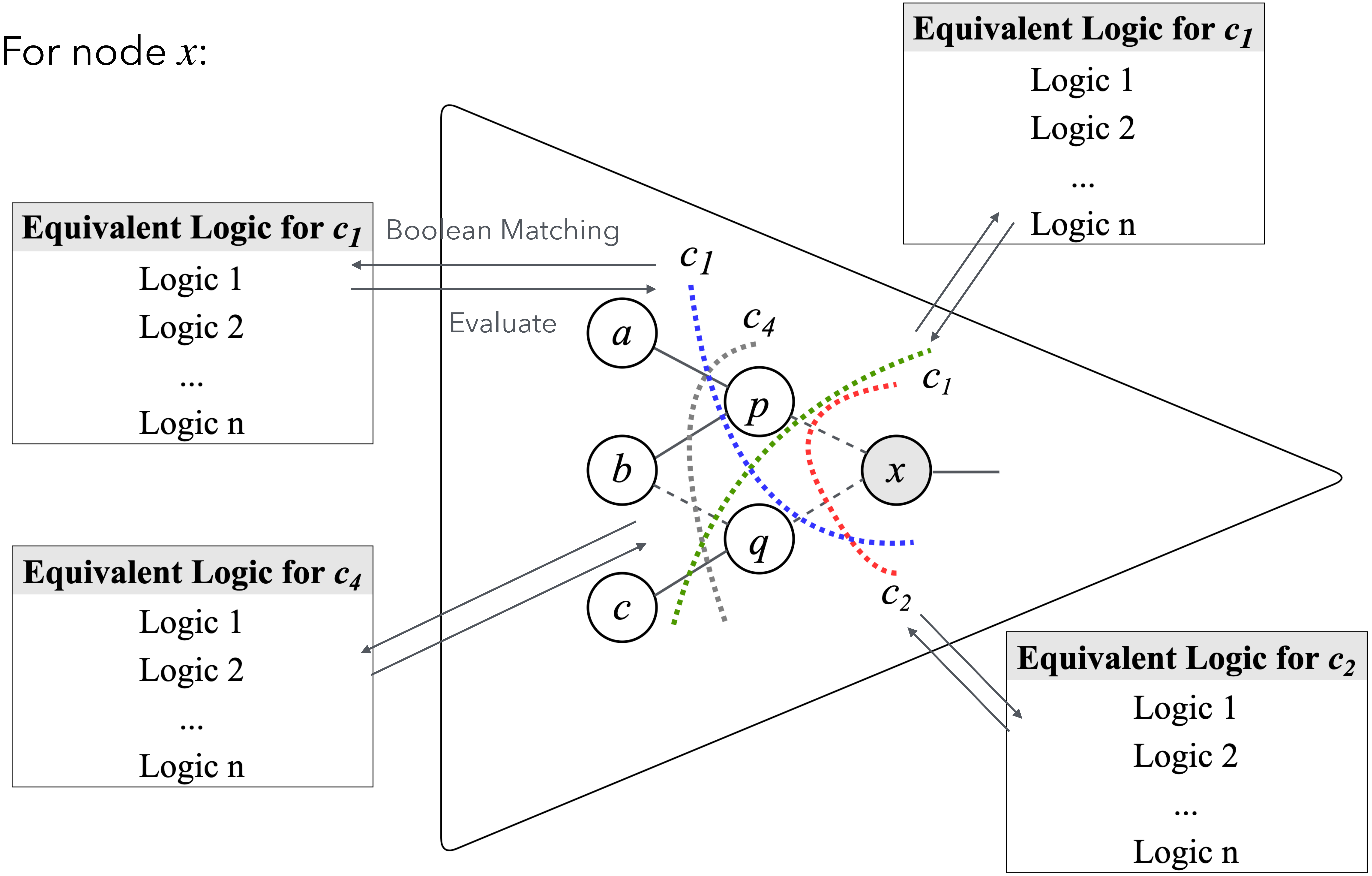
1. Using the pre-computed library to find equivalent logic for each cut c
2. For each cut c, replace the logic with the equivalent that provides the highest gain.
3. Compare all cuts and their best equivalent logic, and select the cut with the highest gain.





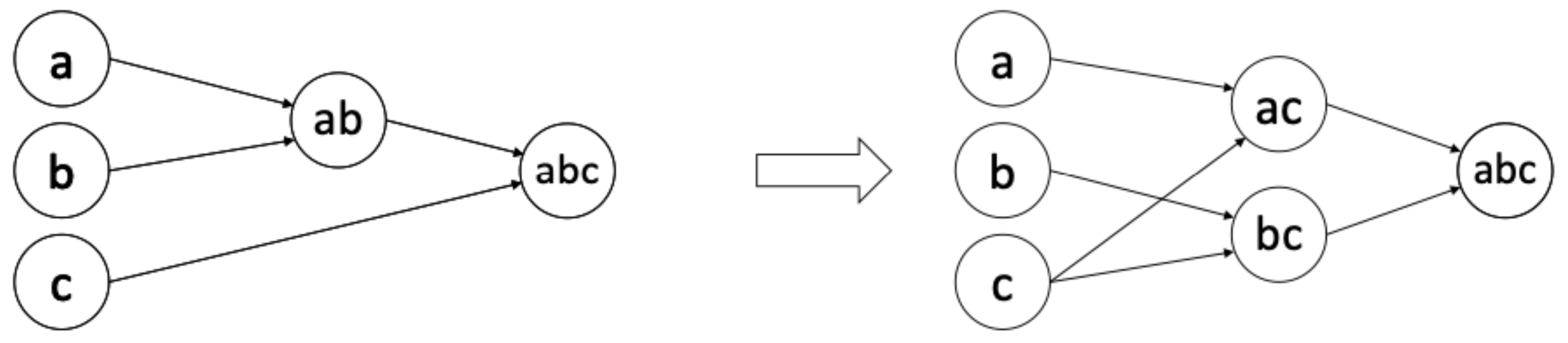
(a) AIG Rewriting

For node x:

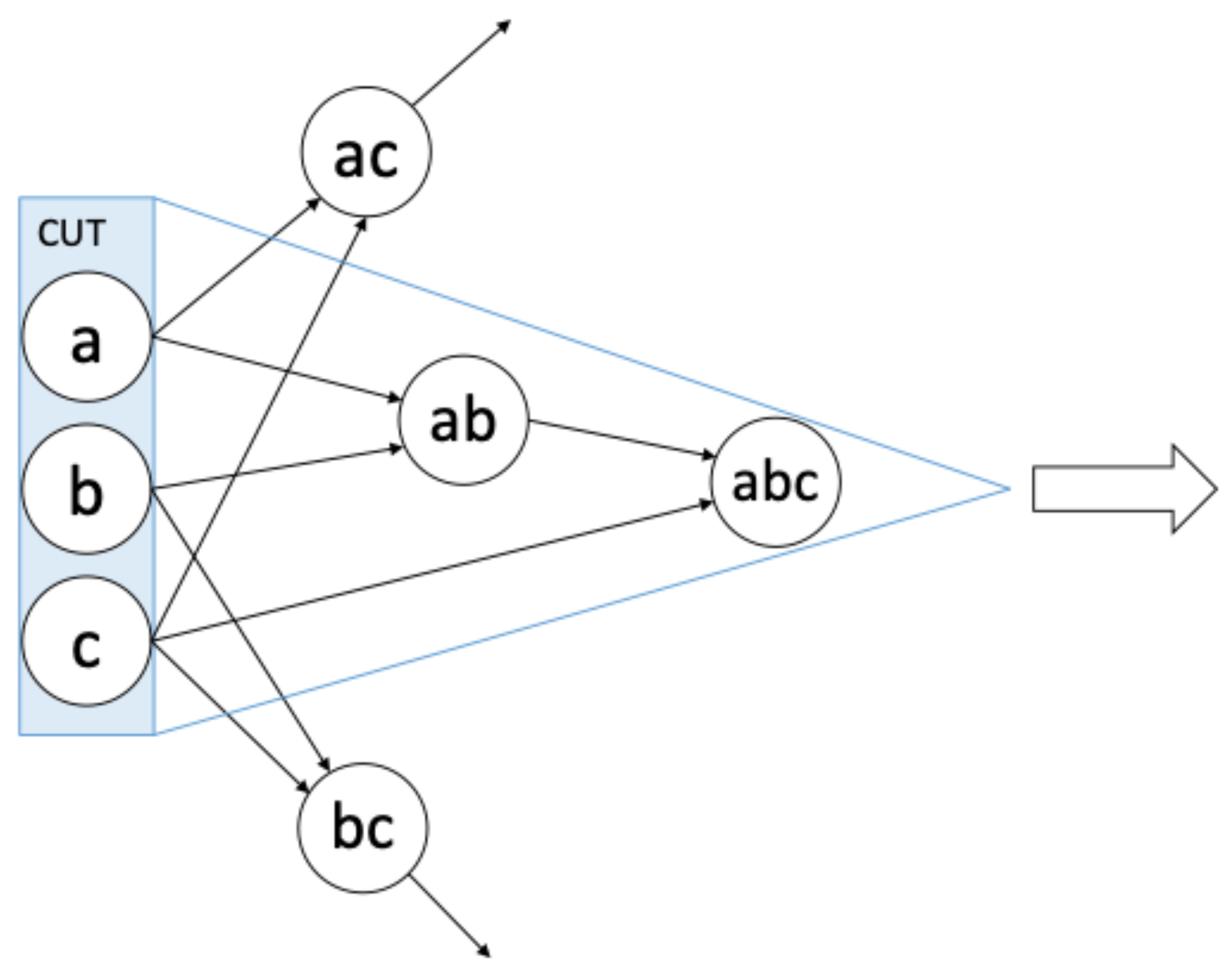


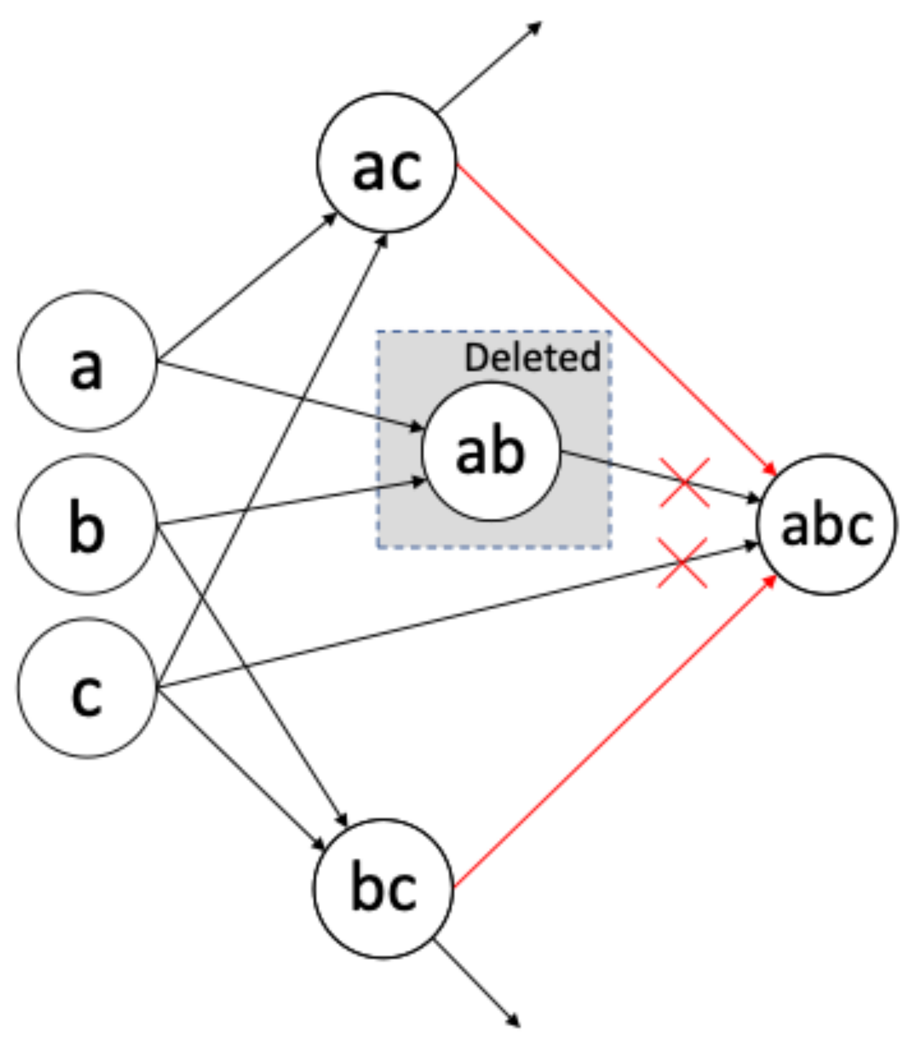
Equivalent Logic for c_1
Logic 1
Logic 2
...
Logic n
Boolean Matching
Evaluate
c_1
c_4
c_1
c_2
a
b
c
p
q
x
Equivalent Logic for c_1
Logic 1
Logic 2
...
Logic n
Equivalent Logic for c_4
Logic 1
Logic 2
...
Logic n
Equivalent Logic for c_2
Logic 1
Logic 2
...
Logic n

AIG Rewriting: Replacement

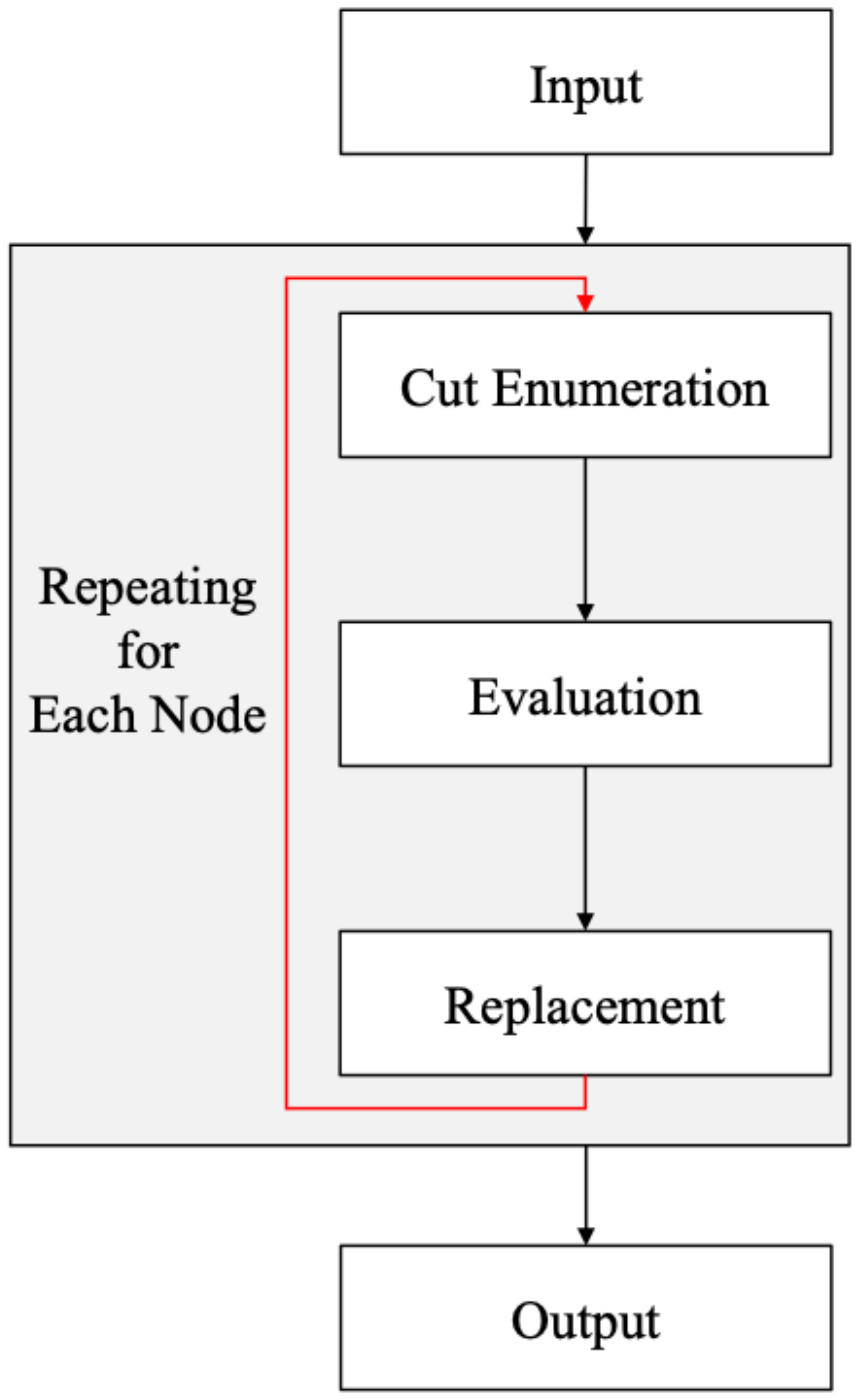


(a) Two Logically Equivalent Graphs





(b) Replacement using Graphs in (*a*)



(a) Single threaded AIG Rewriting

How to optimize circuit

```
abc 01> read mcnc/mlex/dalu.blif
abc 03> strash;print_stats
dalu                          : i/o =   75/   16  lat =    0  and =   1371  lev = 35
abc 05> rewrite
abc 06> print_stats
dalu                          : i/o =   75/   16  lat =    0  and =   1202  lev = 33
```

How to optimize circuit using ABC9

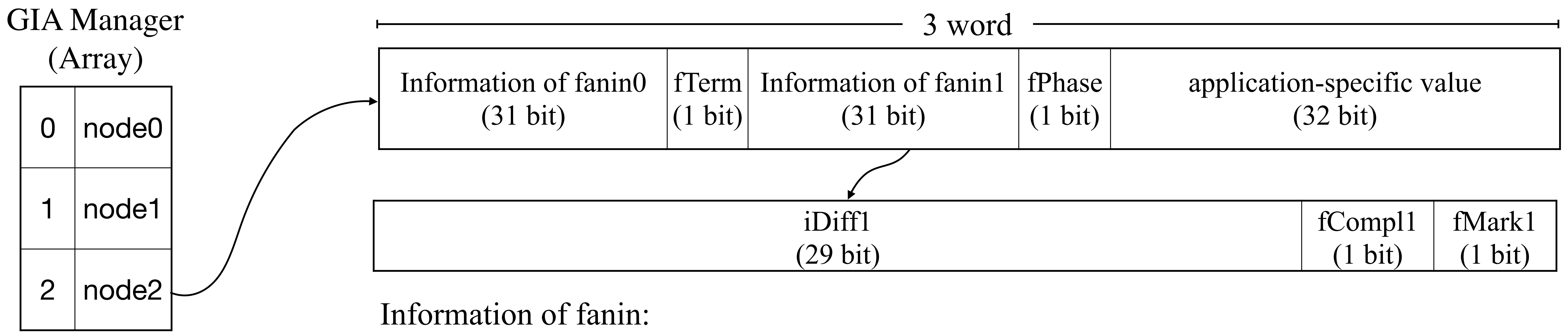
```
abc 01> read mcnc/mlex/dalu.blif
abc 02> strash;print_stats
dalu                          : i/o =   75/   16  lat =    0  and =   1371  lev = 35
abc 03> &get     (# transform AIG to GIA data structure)
abc 03> &ps
dalu     : i/o =     75/     16  and =    1371  lev =   35 (26.88)  mem = 0.02 MB
abc 03> &syn4
abc 03> &ps
dalu     : i/o =     75/     16  and =     997  lev =   35 (23.62)  mem = 0.01 MB
abc 03> &put     (# transform GIA to AIG data structure)
abc 04> print_stats
dalu                          : i/o =   75/   16  lat =    0  and =    997  lev = 35
```

GIA: New AIG manager in ABC9 (better implementation of AIG, 2012)

GIA: Gia_Obj_t

- Cache-Friendly & Memory Efficient: Bit-Packing
- Support native XOR & MUX nodes: *&st -m -L 1* identifies XOR structures, enabling native XAIG representation.
- However, ABC lacks native XAIG based synthesis, so standard algorithms such as rewrite break XAIG equivalence.

Example: node2 = node0 and node1



Information of fanin:

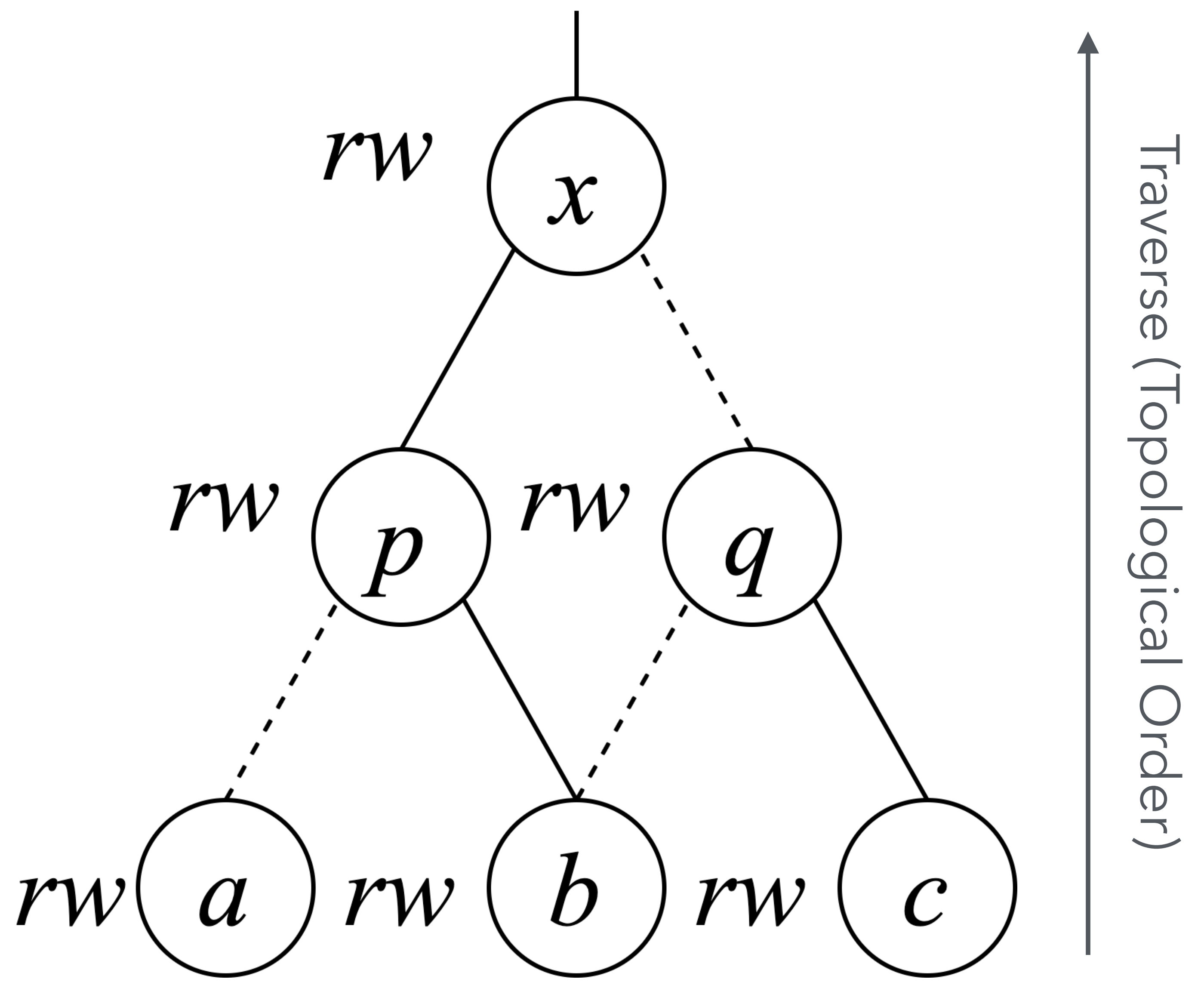
- **iDiff1 (Index Difference):** Index(node2) - Index(node1) = 1
- **fCompl1 (isComplement?):** True
- **fMark1:** User-defined bit, can be used as a flag (e.g., isVisit?)

fTerm: is current node is PI/PO node?

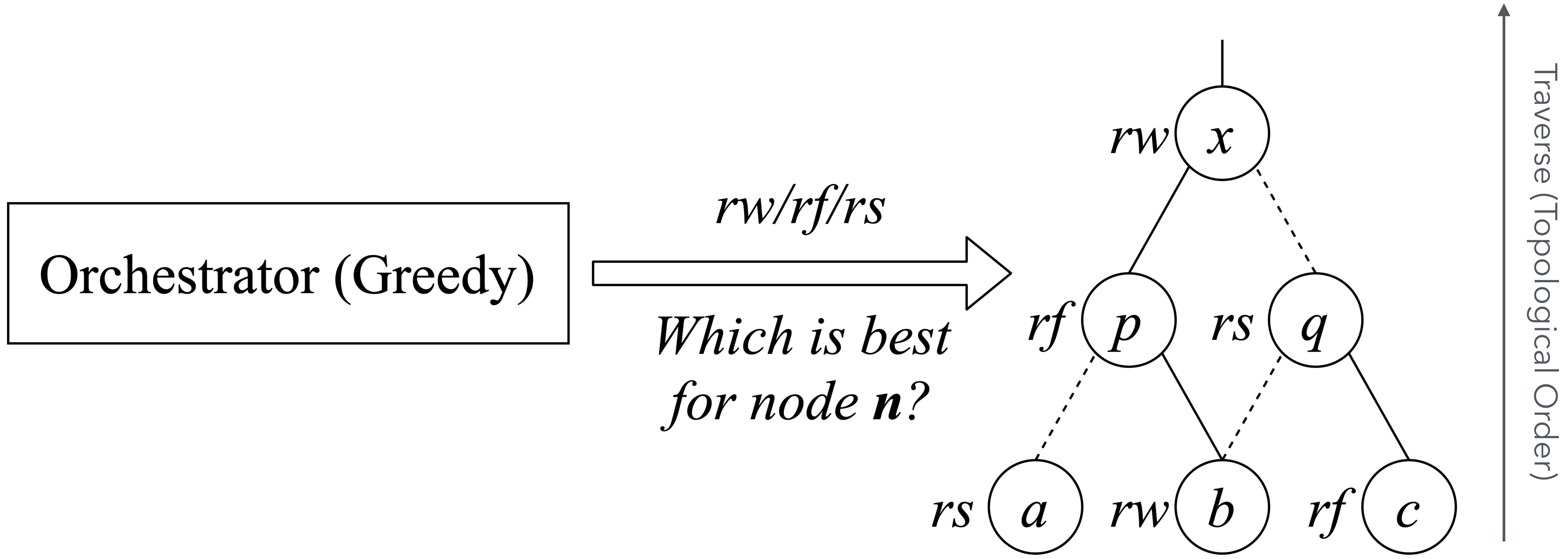
fPhase: Output value of current node under PI = 0000… (can be used in fast simulation)

XOR support: If iDiff0 > iDiff1, node is XOR gate.

orchestrate: Greedy Node-wise Optimization Operator (node level operator, AIG, 2023)



(a) Traditional AIG Rewriting



(b) Orchestration: Greedily optimizes each AIG node by selecting the *rw/rs/rf* with the highest local gain in a single traversal.

How to optimize circuit using orchestration

```
abc 01> read ../mcnc/mlex/dalu.blif
abc 02> strash;print_stats
dalu                    : i/o =   75/   16  lat =    0  and =   1371  lev = 35
abc 03> orchestrate
abc 03> ps
dalu                    : i/o =   75/   16  lat =    0  and =   1100  lev = 32
```

&deepsyn: Automated High-effort Optimization Operator (command level operator, GIA, 2019)

```
abc 01> read ../mcnc/mlex/dalu.blif
abc 02> &get;&ps
dalu     : i/o =     75/     16  and =     1371  lev =   35 (26.88)  mem = 0.02 MB
abc 03> &deepsyn -I 3 -J 200
Completed 200 iterations without improvement in 81.43 seconds.
Completed 200 iterations without improvement in 75.21 seconds.
Completed 200 iterations without improvement in 48.19 seconds.
abc 4858> &ps;&put
dalu     : i/o =     75/     16  and =      527  lev =   14 (12.12)  mem = 0.01 MB
```

Parameter

```
Outer-loop: Start from initial AIG each iteration
   Termination Condition:
      -I  : the number of iterations [default = 1]

Inner-loop: Start from current best and perform greedy randomized optimization
   Termination Condition:
      -J  : the number of steps without improvements [default = 1000000000]
      -T  : the timeout of in seconds (0 = no timeout) [default = 0]
      -A  : the number of nodes to stop (0 = no limit) [default = 0]
```

Why &deepsyn so powerful: Area-increasing Transformation

Repeat:
1. Global transformations
2. Local transformations

Operators selected by &deepsyn

Global transformations (always selected):
- &dch: structural choices
- &if: priority-cut-based *k*-LUT mapping
- &mfs: Area-Oriented *k*-LUT Resynthesis (using windowing & don’t care)

Local transformations (selected probabilistically):
- &fx: fast extraction (Algebraic Division, kernal/cokernal)
- &compress2rs: AIG rewriting for area
- &dc2: AIG rewriting for delay

General AIG Command Framework

Repeat: choice -> LUT mapping -> LUT Resynthesis -> transfer to AIG -> AIG optimization

&dch: structural choices (node level operator, GIA, 2011)

The *&dch* command searches for and stores functionally equivalent nodes (choices) in the AIG, allowing later optimization steps to exploit these structural alternatives.

```
abc 01> read dalu.blif
abc 02> &get;&ps
dalu     : i/o =     75/     16  and =    1371  lev =   35 (26.88)  mem = 0.02 MB
abc 02> &dch
abc 02> &ps
dalu     : i/o =     75/     16  and =    2116  lev =   26 (17.75)  mem = 0.03 MB  ch =  362
cst =       0  cls =    329  lit =     362  unused =    1500  proof =     0
abc 02> &syn4
abc 02> &ps
dalu     : i/o =     75/     16  and =     924  lev =   25 (18.25)  mem = 0.01 MB
```

Reference (use &syn4 directly)

```
dalu     : i/o =     75/     16  and =     997  lev =   35 (23.62)  mem = 0.01 MB
```

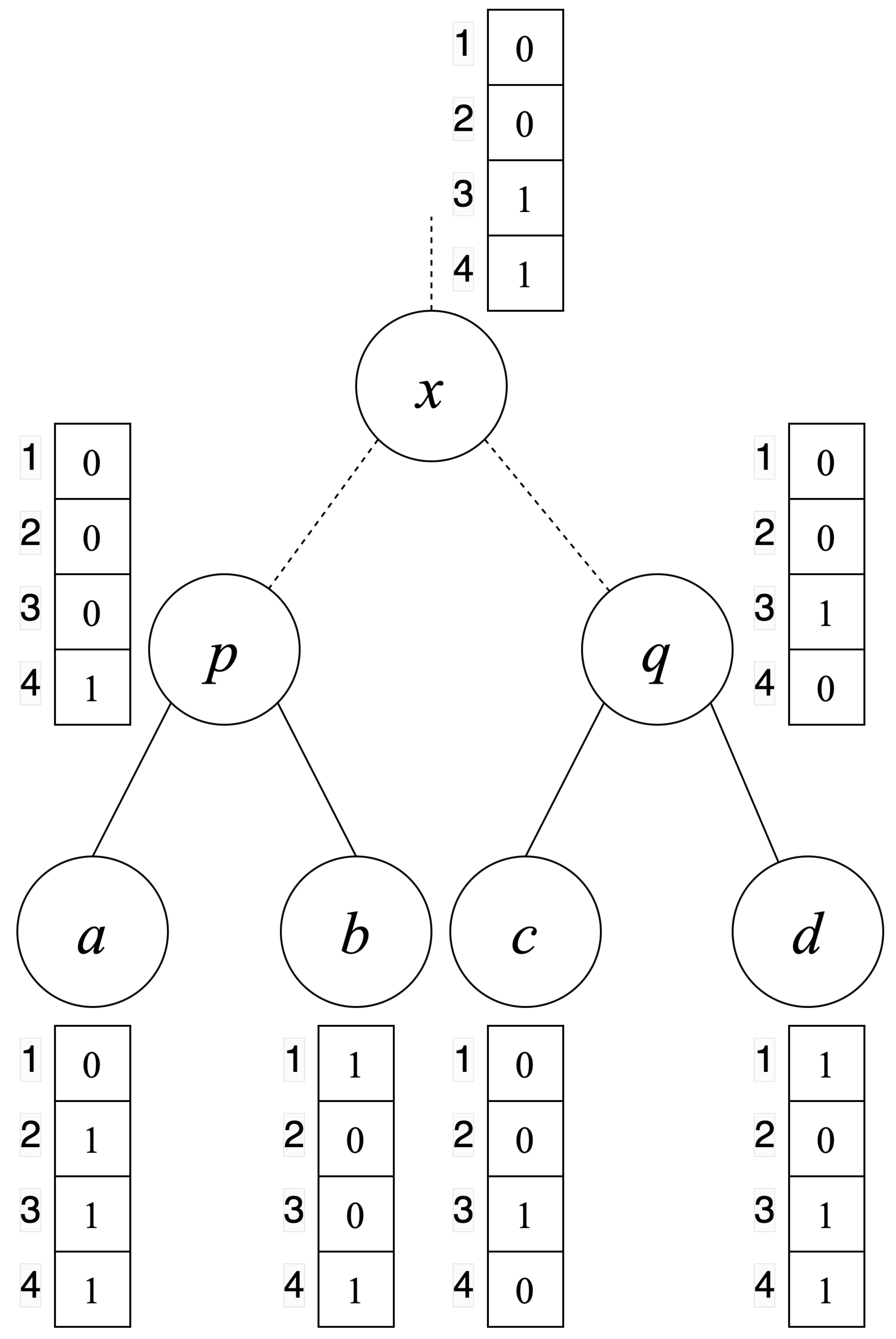
What is structural choices?

Restructured AIG Snapshot
Using Resynthesis to generate more potential equivalent logic
Original AIG
1. Fast Simulation (Coarse grained)
2. SAT Solver (fine grained)
Boolean Matching
Equivalent Class of node *x*
TFI(y)
TFI(x)
TFI(u)
TFI(v)
Choose Logic TFI(x) as Representative
Choice logic for follow-up optimization
Reconstructed AIG after
v
u
h
g
x
y

Bitwise Simulation

Core concept of bitwise simulation

1. Multiple simulation patterns are packed into 32 or 64 bit strings.
2. Perform bitwise simulation at each node in topological order



SAT Solver will introduced in next homework

Thanks