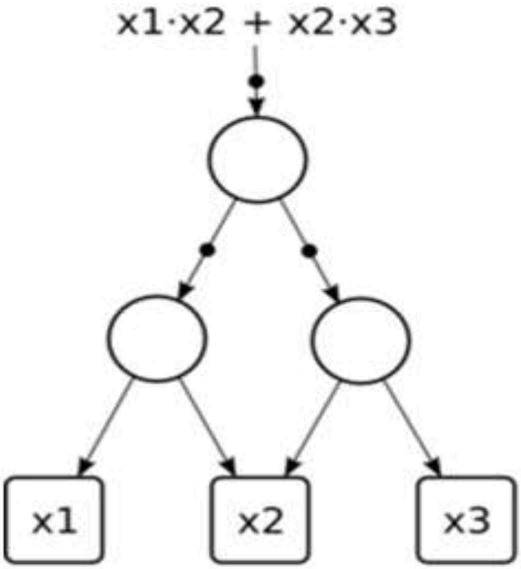
DAC-Aware AIG Re-writing (ABC Tool)

AIG Example

Ex: $y = f(x_1,x_2,x_3) =((x_1 \cdot x_2)' \cdot (x_2 \cdot x_3)')'$

$= (x_1 \cdot x_2) + (x_2 \cdot x_3)$



And-Inverter Graph

And-Inverter Graph (AIG)

- Simple structure
- And-Gates as nodes (shown as circles) with two inputs as edges (shown as arrows)
- Inverter edges marked with a dot
- Used in ABC

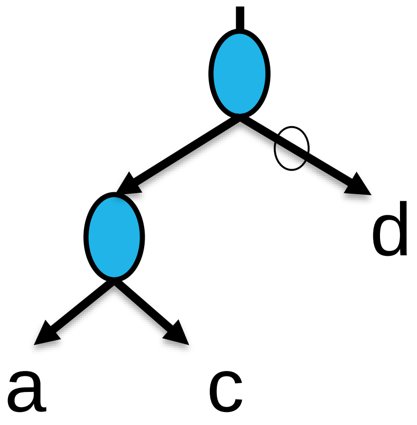
AIG Structure

- A directed graph where

  node = *and* gate

  edge = wire

  circle on edge = inverter



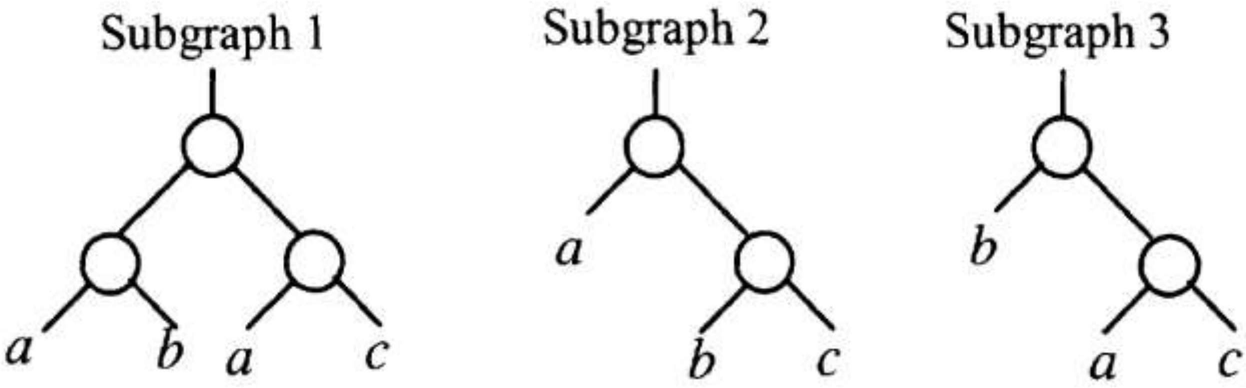
$F = (a \cdot c) \cdot d'$

To Derive a Baseline AIG

- Find SOPs and then factored forms of nodes in a logic network
- Convert AND and OR gates of factored form into 2-input ANDs and inverter using DeMorgan’s rule
- Apply **structural hashing** during AIG construction to ensure that no two AND gates have identical pairs of incoming edges

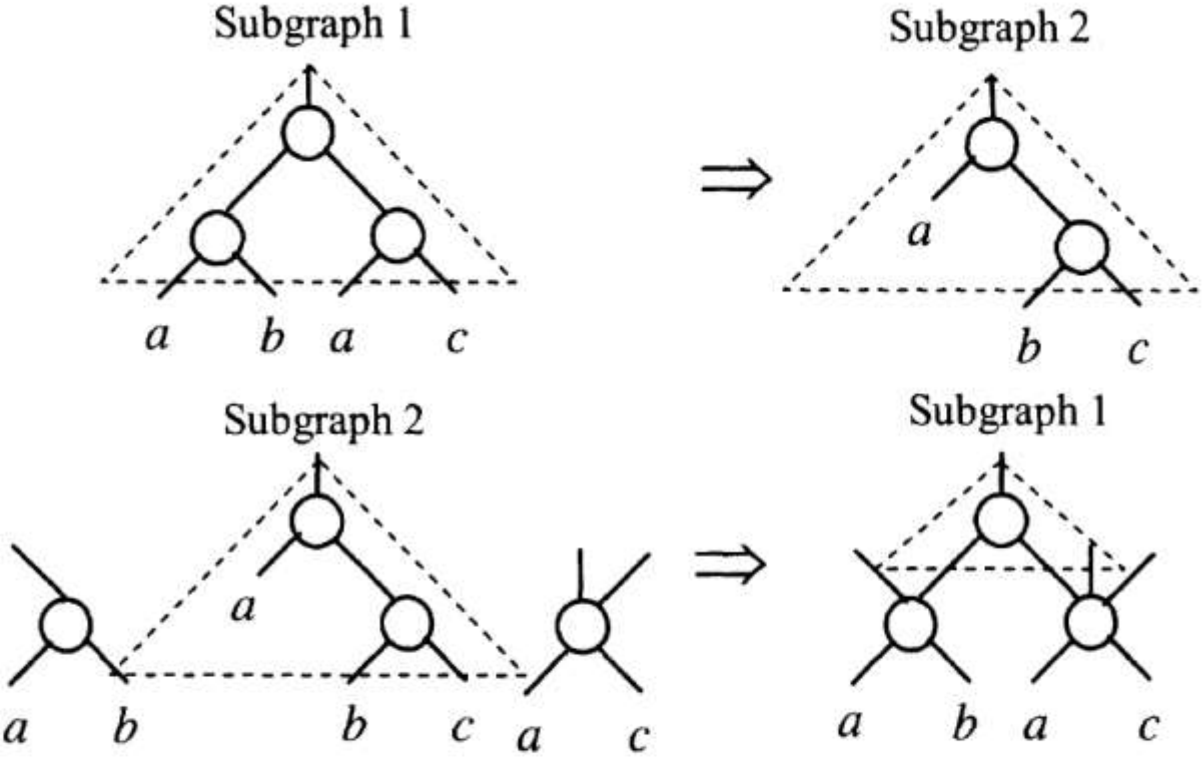
Different AIG Structures for the Same Function

$$F = a \cdot b \cdot c$$



A Simple Example of AIG Rewriting

A rule based transformation



HOW TO RE-WRITE?

A Cut

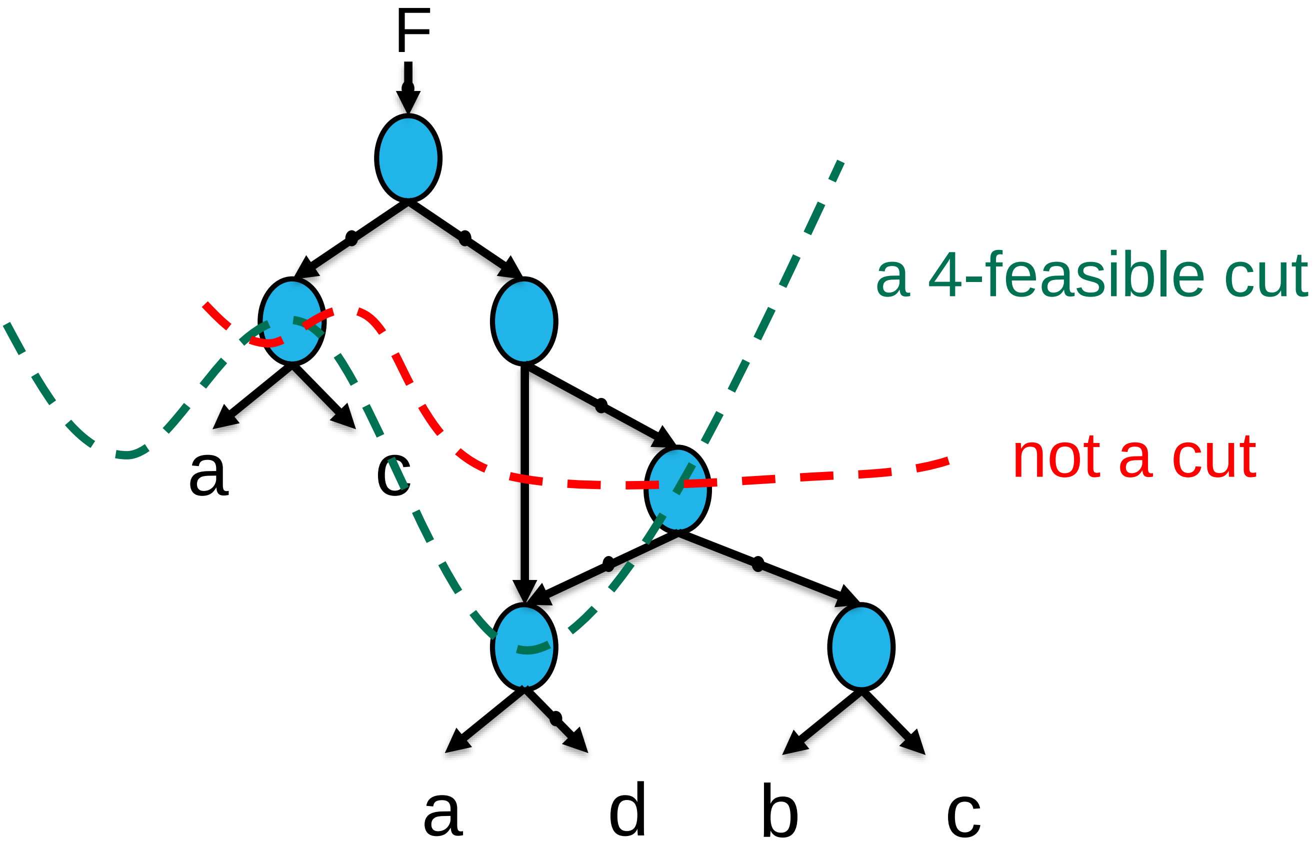
- A cut of C of node N is a set of nodes of the network, called leaves, such that each path from PIs to N passes at least one leaf

- A trivial cut of the nodes is the node itself

- A cut is K-feasible if the number of leaves does not exceed K

An Example of 4-feasible Cut

Find subfunctions rooted at F



Re-Writing Algorithm

- Step 1: Pre-compute all AIG implementations of 4-input functions and store them in a table
  - 2^{16} 4-input functions
  - 222 equivalence classes (NPN)
  - 40 found experimentally to lead to improvement
  - 4-input function stored using 16-bit string (signature)
  - AIG subgraphs stored in shared DAC with about 2000 nodes

. Step 2:

- For a node, find its 4-feasible cuts
  - For each cut, find its NPN equivalence
    - compute the cost of a subgraph
    - choose the subgraph that leads to the largest improvement

- Nodes are processed in topological order
- Logic sharing is checked between the new subgraph and nodes already in the network using reference counters
- The old subgraph is dereferenced and the new subgraph is added

Delay-aware Re-writing

- A subgraph representation will not be accepted if the final logic level is increased
  - Using slack of the node
  - No negative slack after replacement

AIG Refactoring

- Produce deeper permutations of the logic structures
  - Work for larger cuts, K, for 10 <= K <= 20
  - The function is converted to SOP, factored, AIGs built using baseline AIG rewriting

AIG Balancing

- For delay optimization
  - A(BC) = (AB)C = (AC)B is applied to maximally reduce the number of levels of AIG
  - One linear time sweep over the network in a topological order

Zero-cost Replacement Enabled

- Create new re-writing opportunity
  - If the option is enabled, the node is replaced by a new subgraph if the cost = 0
  - Enabled later in the script

An Example of Script in ABC

- A re-writing script, *resyn2*, in *abc.rc*
- b (balance) ;
  rw (rewrite);
  rf (refactor);
  b;
  rw;
  rwz (re-write with 0 cost);
  b ;
  rfz (refactor with 0 cost);
  rwz;
  b
- Perform 10 times over the network