cadence™

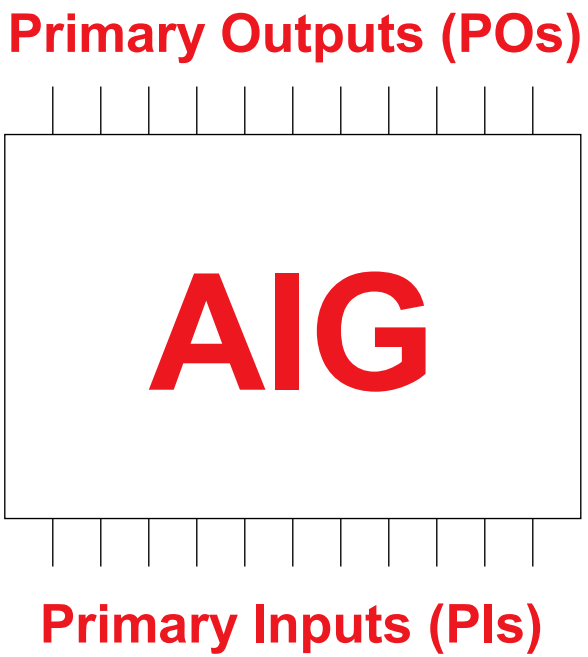
Applying Logic Synthesis for Speeding Up SAT

Niklas Een
Alan Mishchenko
Niklas Sörensson

Problem Formulation

- Given a (combinational) circuit, how do we best clausify it for the CNF-based SAT-solver?

Assume given on *And-Inverter Graph* (AIG) form:



Contract: Clausify logic such that the (functional) relation between PIs and POs is established.

...gives the freedom to:
- Omit internal signals
(don't give them a SAT variable)
- Create new internal signals
(by circuit simplifying rewrites)

cadence

Clausification in the small ("easy")

- How to produce a small set of clauses for 1-output, *k*-input subgraphs ("super-gates") of the AIG for small *k*:s:
  - **k ≤ 4:** Pre-compute and tabulate exact results
  - **4 < k ≤ 16:** Use Minato's ISOP-algorithm

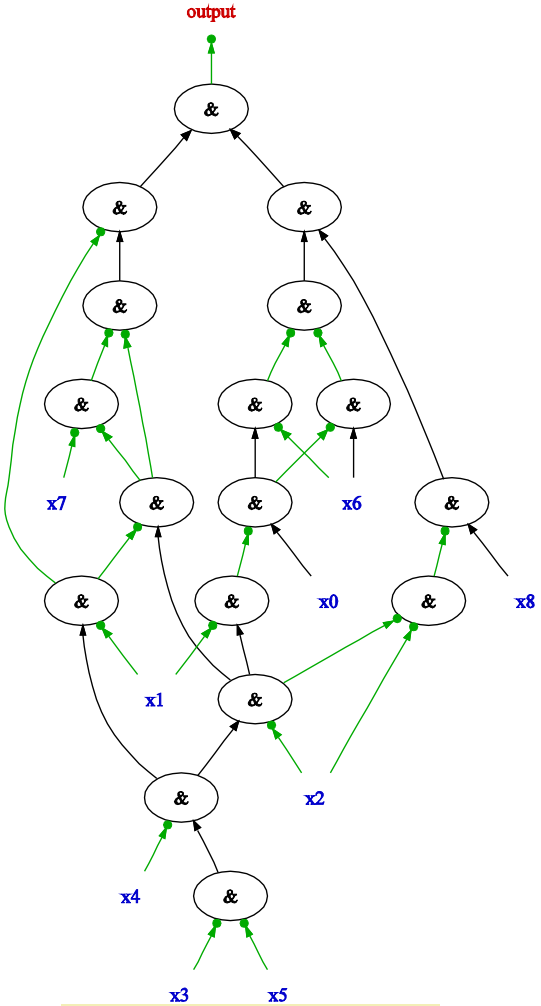
Clausification in the large (hard!)

- How to partition AIG into super-gates?
- How to handle reconvergence? *(the root of all evil)*

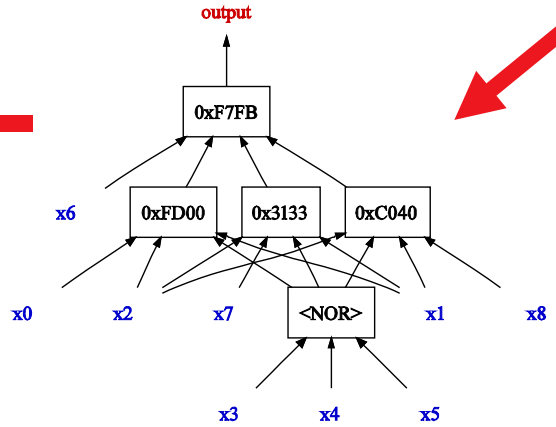
Proposal: Use FPGA-style technology mapping!

Overview – “The three staged rocket”

- The AIG representing the SAT problem is minimized (DAG-aware).
- The minimized AIG is translated into *k*-input LUTs. “cost(LUT) = #clauses”
- Final CNF is optimized by variable elimination and subsumption.



AIG minimization



Techmap for CNF

```
11 7 6 5        -13 4
-11 -7          14 11 9
-11 -6          14 -4 9
-11 -5          14 3 9
12 11 -2        -14 -11 4 -3
12 -4 -2        -14 -9
12 -3 -2        15 -12 -8 -13 14
-12 -11 4 3     15 12 8 -13 14
-12 2           -15 12 -8
13 11 -10 -4    -15 -12 8
13 -3 -10 -4    -15 13
-13 -11 3       -15 -14
-13 10
```

CNF minimization

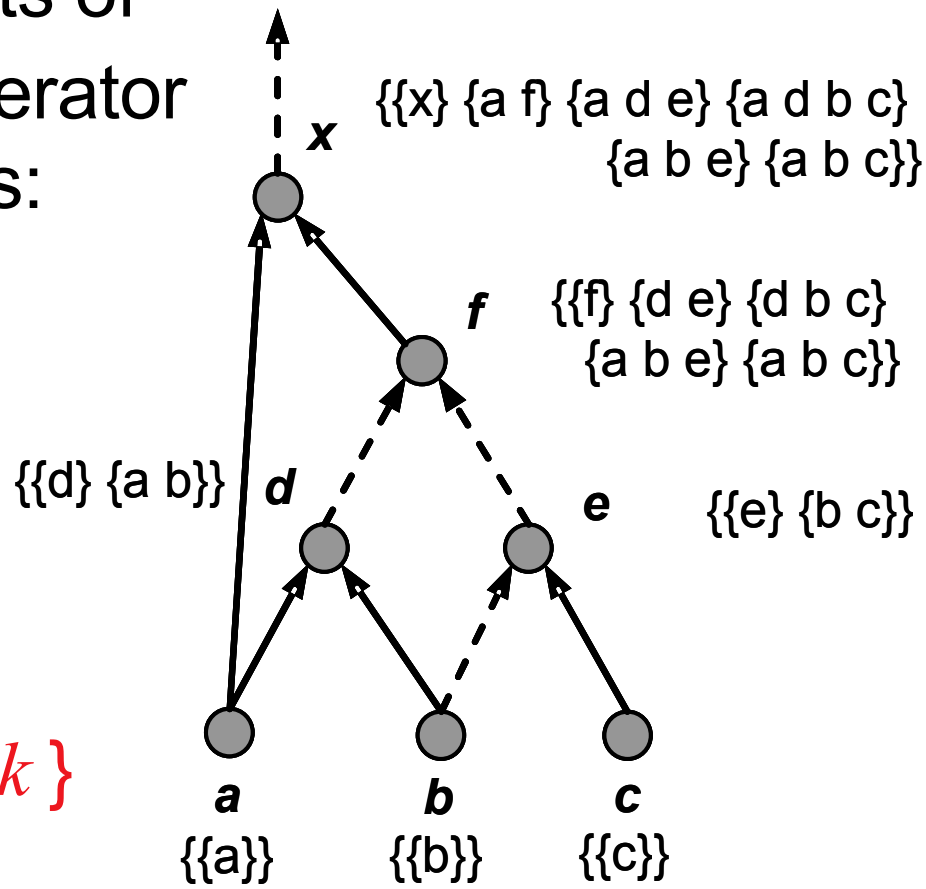
cadence

Cut Enumeration

Let Δ_1 and be Δ_2 two sets of cuts, and the merge operator $\otimes_k$ be defined as follows:

$$\Delta_1 \otimes_k \Delta_2 :=$$

$$\{ C_1 \cup C_2 \mid C_1 \in \Delta_1, C_2 \in \Delta_2, |C_1 \cup C_2| \leq k \}$$

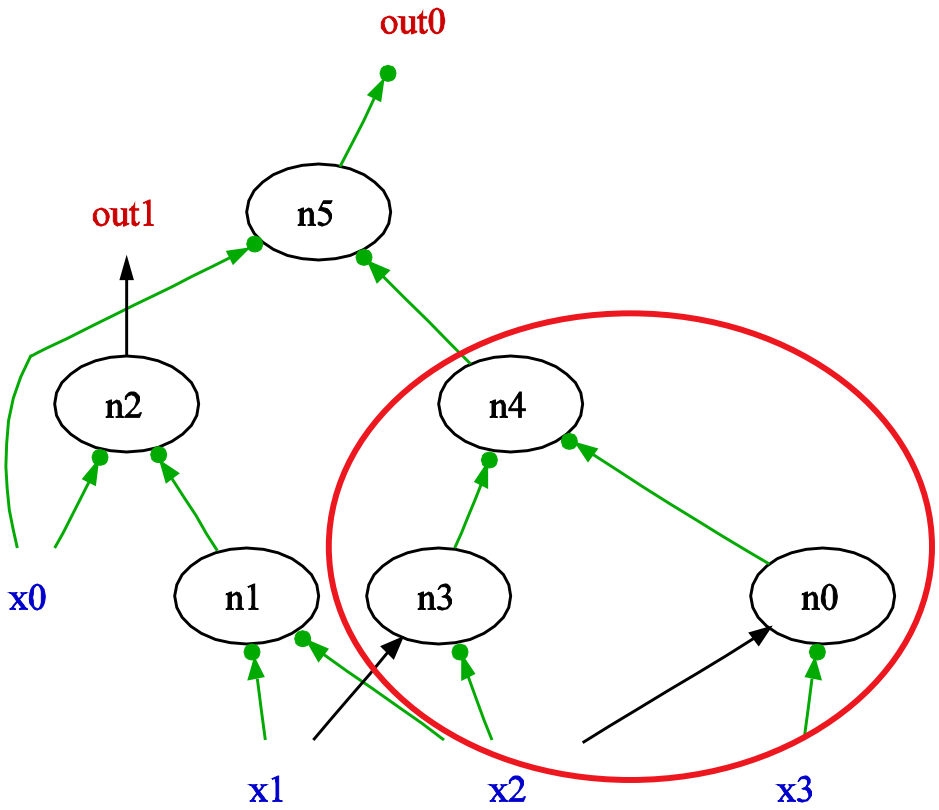


cadence

DAG-Aware Minimization

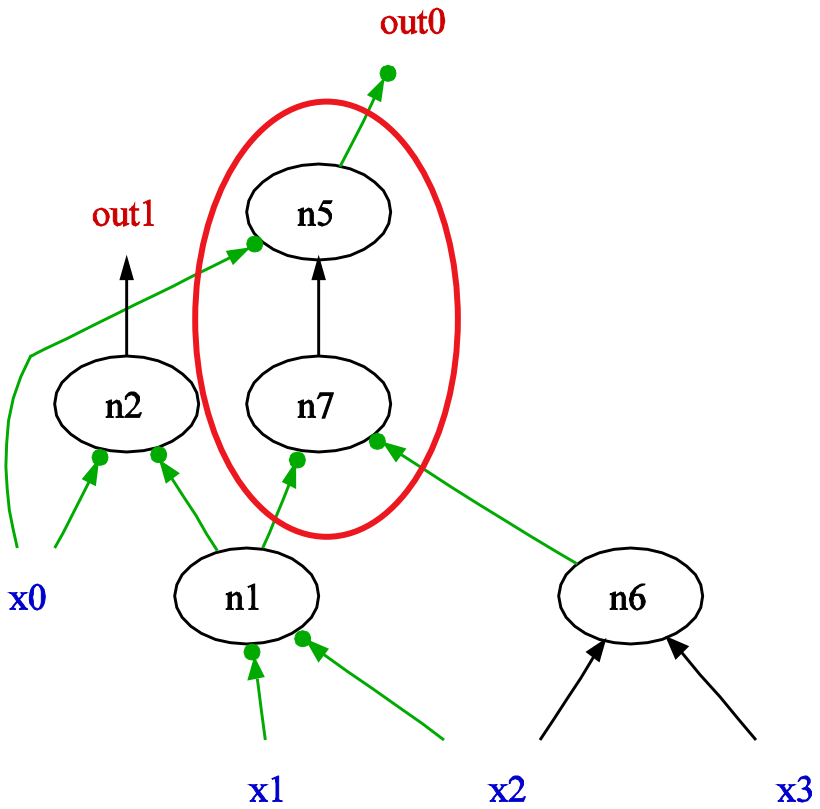
- Minimizes an AIG taking sharing into account.
- Compute “good” AIG representations for each 4-input function.
- Enumerate all cuts: see if cut cone can be replaced by node saving representation.
- If time-budget admits: perturb and repeat.

DAG-Aware Minimization: Example



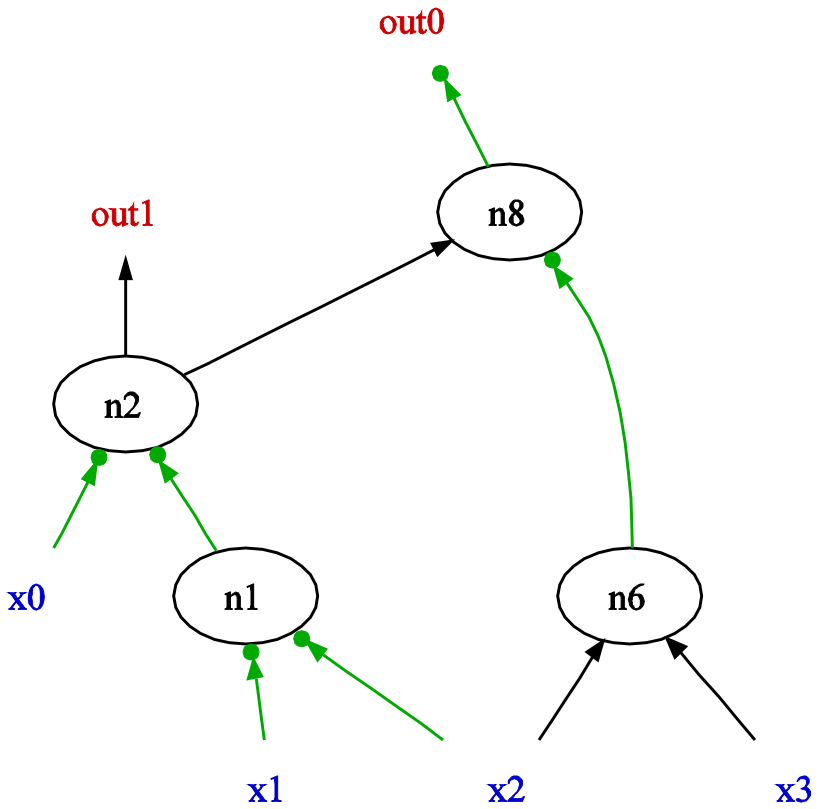
cadence

DAG-Aware Minimization: Example



cadence

DAG-Aware Minimization: Example



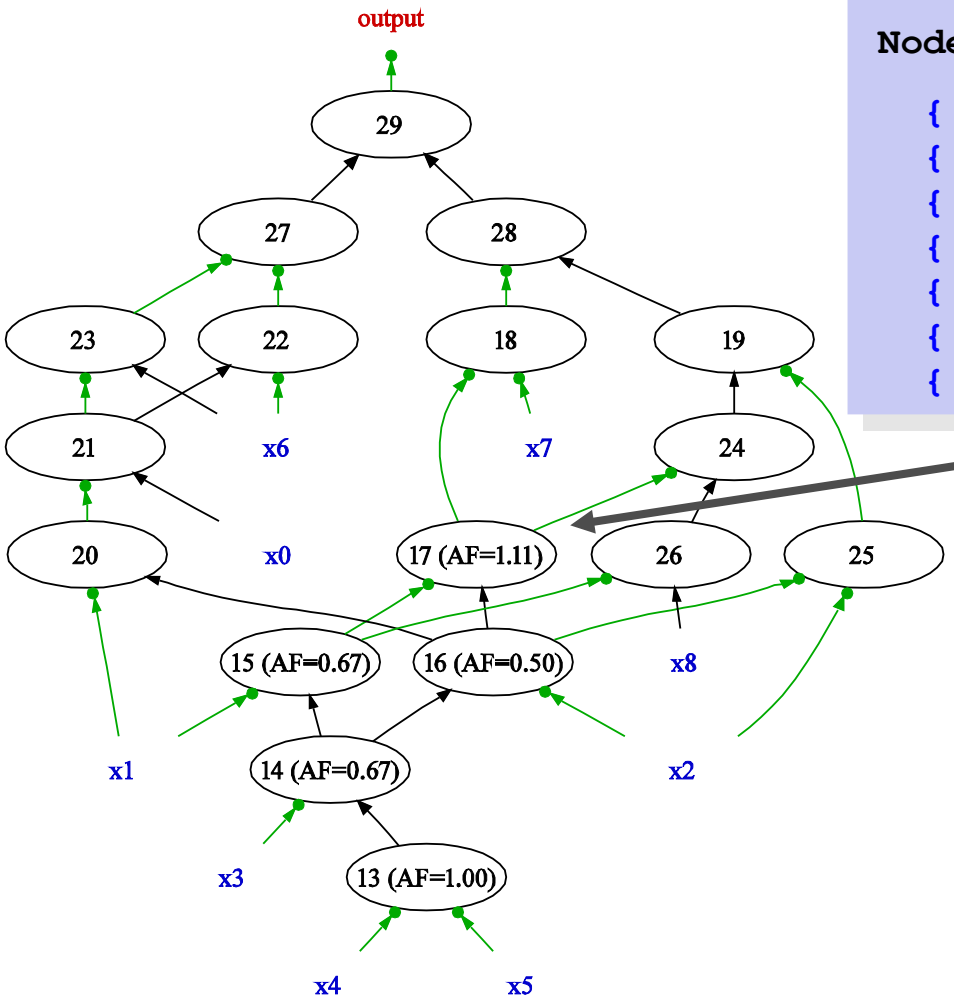
Technology Mapping

- Enumerate all *k*-input cuts (k=4 in example).
- Select a cut for each node (= potential LUT).
- Outputs from logic will recursively induce a subgraph corresponding to LUT representation.
- *Area Flow:* Estimate the area increase that would result from including a node:

$$\frac{\text{cost of node + cost of children}}{\text{estimated number of fanouts}}$$

- "cost of node = 1" in example on following slides (for simplicity), but "#clauses" in real algorithm.

Techmapping – enumerate cuts



Node 17: [req. time=2, fanout est.=1.5]

{ 16 15 }	t=2	AF=1.44	CS=2
{ 15 14 x2 }	t=2	AF=1.56	CS=3
{ 14 x2 x1 }	t=2	AF=1.11	CS=3
{ 16 14 x1 }	t=2	AF=1.44	CS=3
{ 15 13 x3 x2 }	t=2	AF=1.78	CS=4
{ 13 x3 x2 x1 }	t=2	AF=1.33	CS=4
{ 16 13 x3 x1 }	t=2	AF=1.67	CS=4

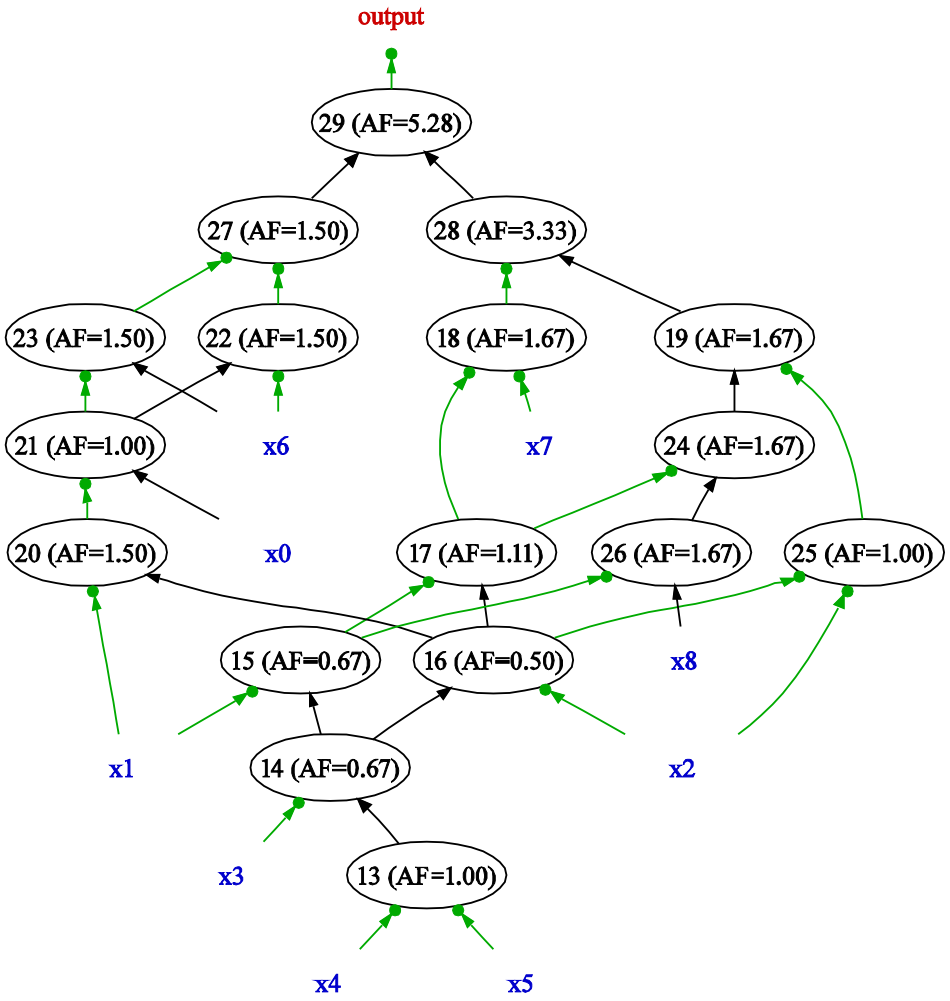
t = Arrival Time

AF = Area Flow
(estimated area required for introducing node)

CS = Cut Size

cadence

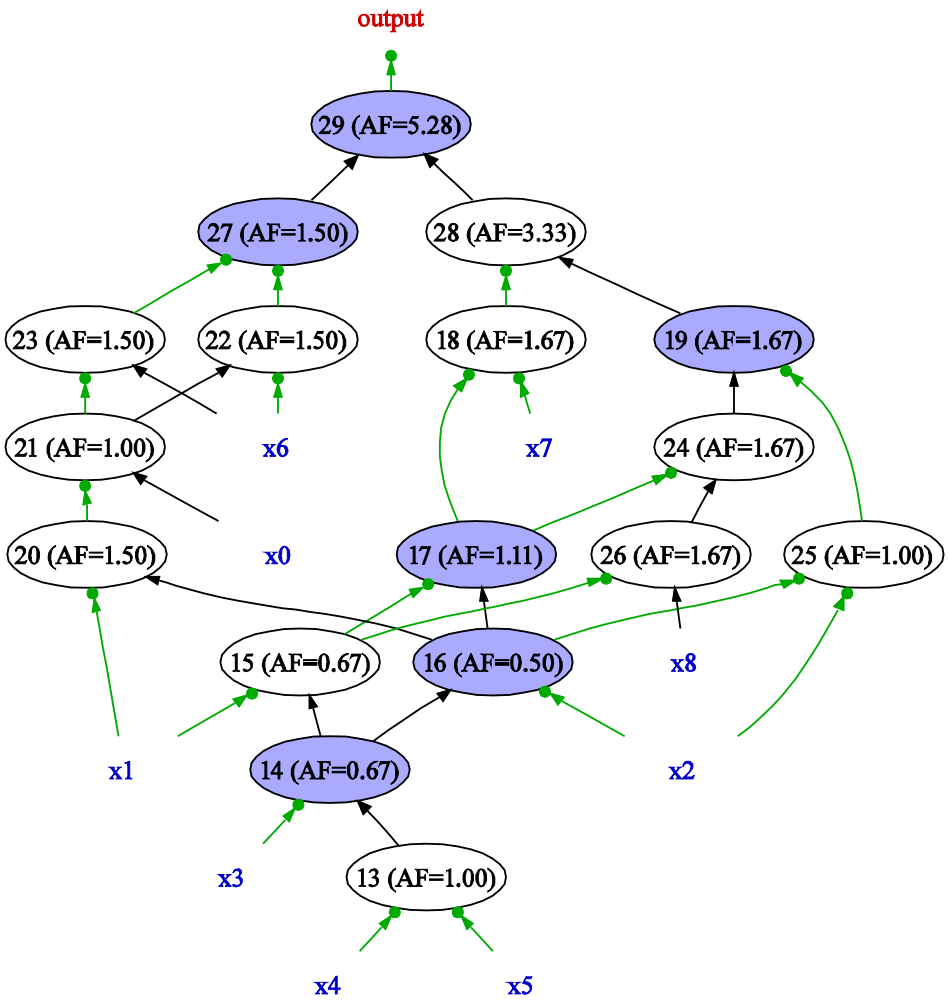
Techmapping – best cut for each node




```
13: { x5 x4          }
14: { x5 x4 x3       }
15: { x5 x4 x3 x1    }
16: { x5 x4 x3 x2    }
17: { 14 x2 x1       }
18: { 14 x7 x2 x1    }
19: { 14 x8 x2 x1    }
20: { 16 x1          }
21: { 16 x1 x0       }
22: { 16 x6 x1 x0    }
23: { 16 x6 x1 x0    }
24: { 14 x8 x2 x1    }
25: { x5 x4 x3 x2    }
26: { 15 x8          }
27: { 16 x6 x1 x0    }
28: { 18 15 x8 x2    }
29: { 27 19 17 x7    }
```


cadence™

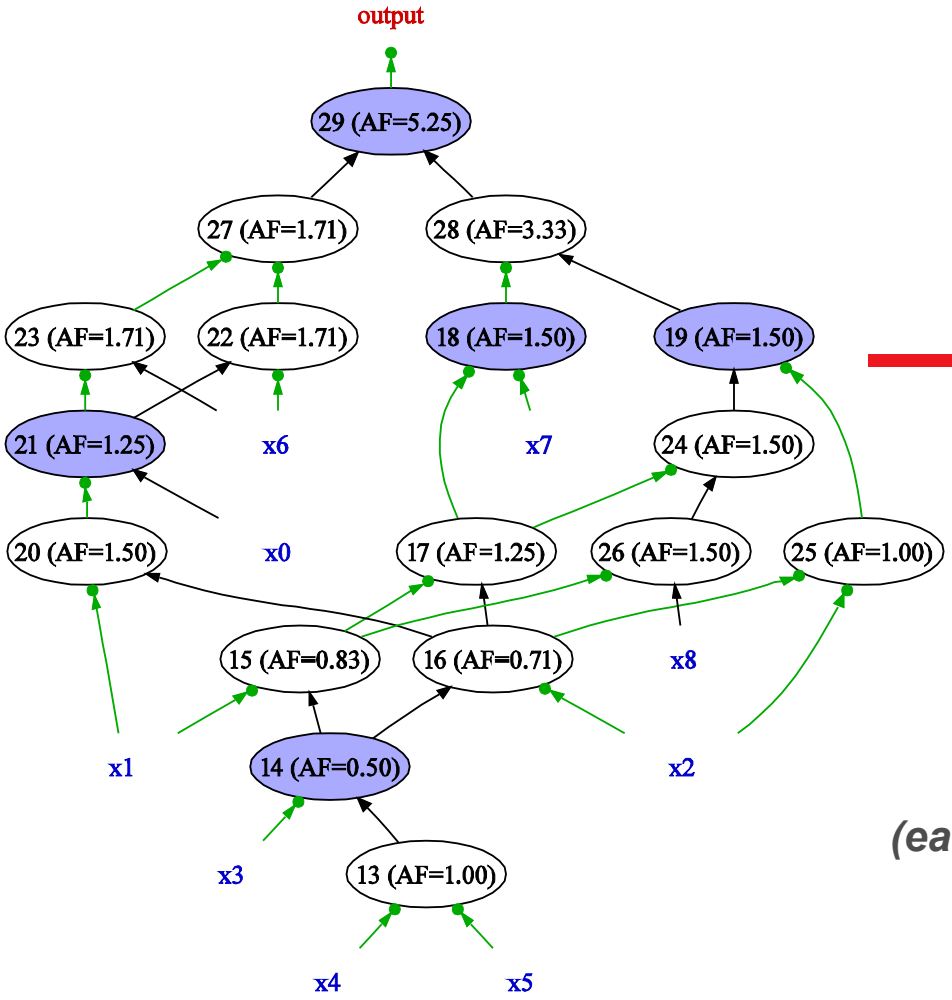
Techmapping – induced subgraph

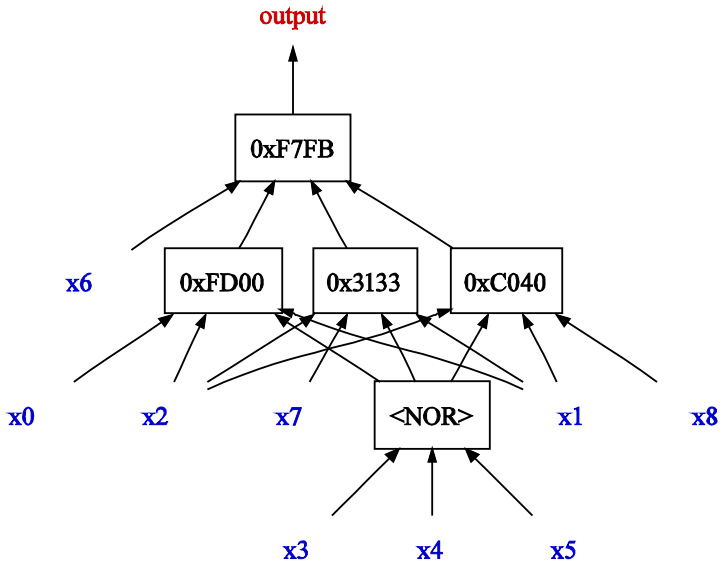



```
13: { x5 x4          }
14: { x5 x4 x3       }
15: { x5 x4 x3 x1    }
16: { x5 x4 x3 x2    }
17: { 14 x2 x1       }
18: { 14 x7 x2 x1    }
19: { 14 x8 x2 x1    }
20: { 16 x1          }
21: { 16 x1 x0       }
22: { 16 x6 x1 x0    }
23: { 16 x6 x1 x0    }
24: { 14 x8 x2 x1    }
25: { x5 x4 x3 x2    }
26: { 15 x8          }
27: { 16 x6 x1 x0    }
28: { 18 15 x8 x2    }
29: { 27 19 17 x7    }
```


cadence™

Techmapping – iterate procedure





"Super-gate" representation

(each box will expand to a set of clauses)

cadence

Benchmark Results

Problem	SAT Runtime (sec) – Cadence BMC (orig)	S	D	DS	T	TS	DT	DTS
CIS-70u	21.9	12.3	3.6	3.1	2.5	4.1	**1.2**	1.3
CIS-71s	15.2	8.8	7.7	3.9	**2.1**	3.1	4.0	2.7
olm-154u	116.4	48.3	41.1	37.7	11.6	34.4	15.6	**9.3**
olm-155s	101.8	22.9	12.9	16.2	18.2	50.6	13.4	**6.9**
r1-18u	1516.0	139.4	361.9	119.4	196.3	78.8	78.8	**39.0**
r1-19s	1788.2	276.7	535.0	154.8	317.8	137.1	131.9	**42.5**
r2-19u	403.8	214.4	239.8	169.7	140.9	**73.7**	114.8	78.1
r2-20s	3066.1	893.4	1002.9	353.2	376.2	313.5	687.5	**96.5**
r6-19u	316.1	225.6	133.9	104.7	107.9	107.6	**53.2**	55.0
r6-20s	2305.4	456.4	863.1	385.8	507.0	236.9	307.2	**101.2**
Total speedup:		4.2x	3.0x	7.2x	5.7x	9.3x	6.9x	22.3x
Arithmetic average speedup:		3.9x	3.6x	6.5x	6.3x	7.6x	9.2x	19.7x
Harmonic average speedup:		2.7x	2.9x	4.8x	5.3x	4.9x	6.6x	11.5x

S = cnf **S**implification D = **D**ag shrink T = **T**echmap for cnf

Conclusions

- Two techniques from logic synthesis was used:
  - DAG-aware minimization
  - Technology mapping (adapted for CNF generation)
- Both techniques contributed to speed-ups
- Orthogonal to CNF-based preprocessing
- For BMC problems, the speedup was ~10x.