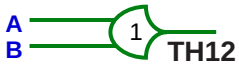
The NCL Function Set and its Use

by Karl Fant

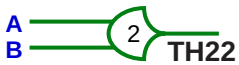
Feb, 2015

NCL Function Set

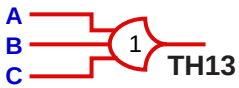
1. A



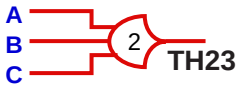
2. A | B



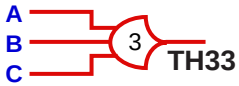
3. AB



4. A | B | C



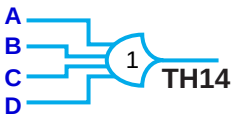
5. AB | BC | AC



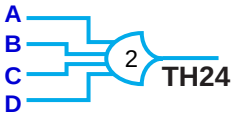
6. ABC

7. A | BC

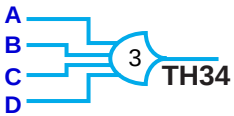
8. AB | AC



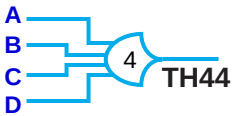
9. A | B | C | D



10. AB | AC | AD | BC | BD | CD



11. ABC | ABD | ACD | BCD



12. ABCD

13. A | BC | BD | CD

14. AB | AC | AD | BCD

15. ABC | ABD | ACD

16. A | BCD

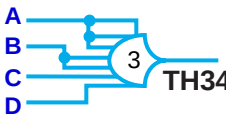
17. AB | AC | AD

18. A | B | CD

19. AB | AC | AD | BC | BD

20. AB | ACD | BCD

21. ABC | ABD

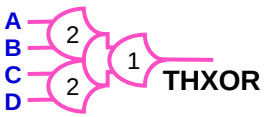


22. A | BC | BD

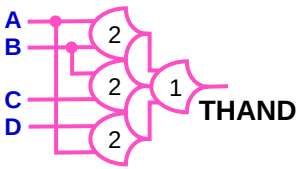
23. AB | ACD

24. AB | AC | AD | BC

25. AB | AC | BCD



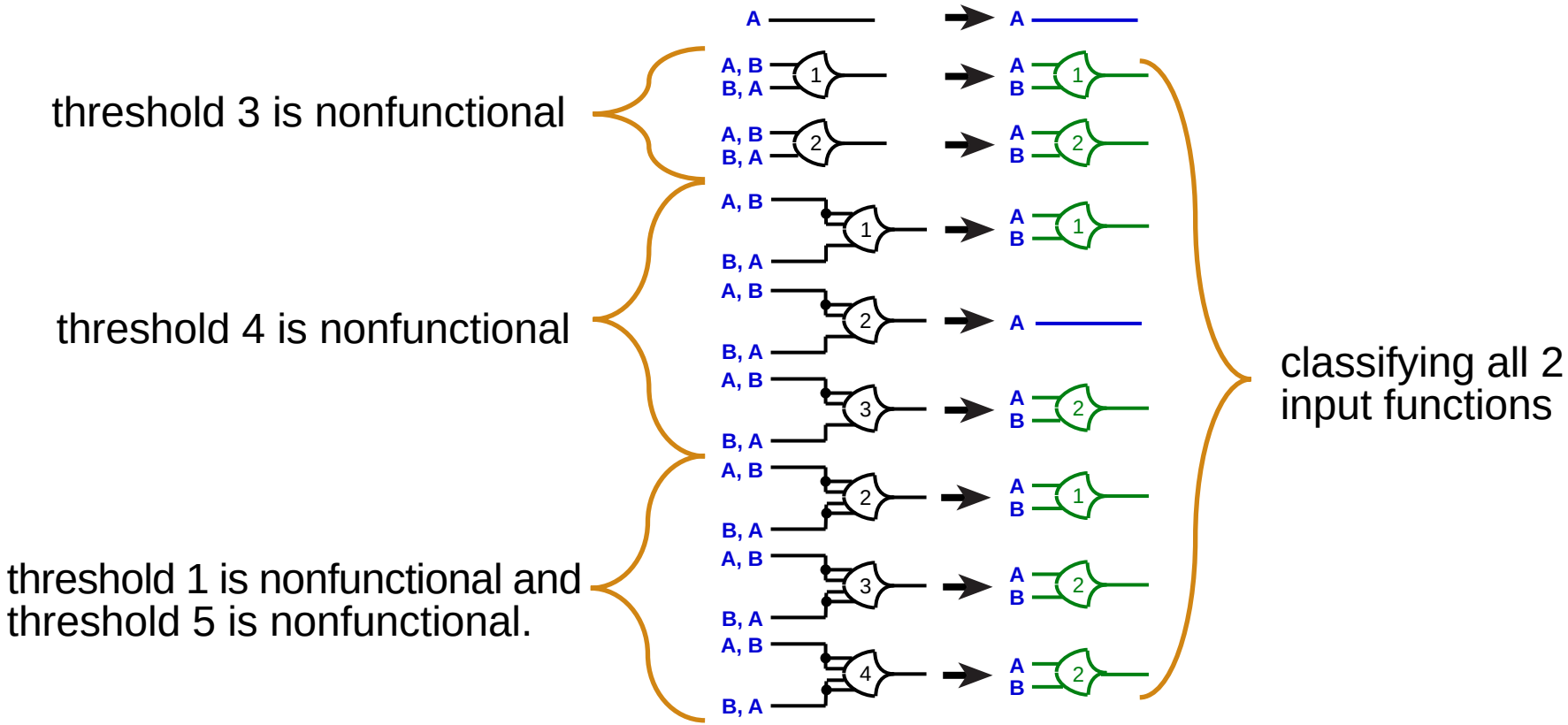
26. AB | CD



27. AB | BC | AD

28. AC | BC | AD | BD

Classifying Threshold Functions



A weight of 3 with two inputs can always be reduced to an expression with a weight of 2 so there is no need to consider further configurations of 2 inputs.

Threshold function classes of one and two inputs.

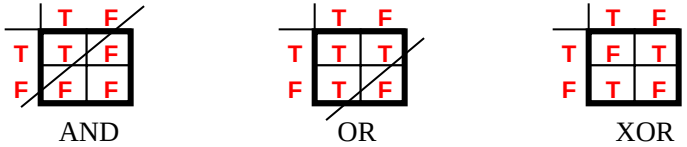
Following a similar procedeure one discovers 5 three input classes and 17 four input classes for 25 classes of 0ne, two, three and four input threshold functions. In the threshold function table functions 1 through 25 represent the 25 classes.

Muroga, Saburo, “Threshold Logic and its Applications”, Wiley-Interscience, 1971

The Function Set Rationale

There is a one to one mapping between the NCL functions and the classes of positive Boolean functions of 4 or fewer inputs

All threshold functions are linearly separable and map directly to the linearly separable Boolean functions.



All linearly separable Boolean functions are positive but not all positive Boolean functions are linearly separable.

There are 28 PN classes of positive Boolean functions of 4 or fewer variables. 25 of these classes are linearly separable and map directly onto the 25 classes of threshold functions with 4 or fewer inputs. There are three positive 4 input Boolean functions that are not linearly separable. These are included in the set of functions as multi function circuits. (26, 27 and 28).

Classifying Boolean functions

PN classes
permutation and negation of inputs
positive function classes are red

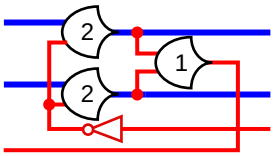
0 variable	M0	0	
	M1	1	
1 variable	M2	x_1	**1. A**
2 variable	M3	x_1x_2	**3. AB**
	M4	$x_1 \vee x_2$	**2. A + B**
	M5	$x_1 \oplus x_2 = x_1\bar{x}_2 \vee \bar{x}_1x_2$	
3 variable	M6	$x_1x_2 \vee x_2x_3 \vee x_1x_3$	**5. AB + BC + AC**
	M7	$x_1 \oplus x_2 \oplus x_3 = x_1(x_2\bar{x}_3 \vee \bar{x}_2x_3) \vee \bar{x}_1(\bar{x}_2\bar{x}_3 \vee x_2x_3)$	
	M8	$x_1x_2x_3$	**6. ABC**
	M9	$x_1 \vee x_2 \vee x_3$	**4. A + B + C**
	M10	$x_1(x_2 \vee x_3)$	**8. AB + AC**
	M11	$x_1 \vee x_2x_3$	**7. A + BC**
	M12	$x_1x_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3$	
	M13	$(x_1 \vee x_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$	
	M14	$\bar{x}_1x_2x_3 \vee x_1\bar{x}_2 \vee x_1\bar{x}_3$	
	M15	$x_1(x_2x_3 \vee \bar{x}_2\bar{x}_3)$	
	M16	$x_1 \vee x_2\bar{x}_3 \vee \bar{x}_2x_3$	
	M17	$x_1x_2 \vee x_2x_3 \vee \bar{x}_1x_3$	
	M18	$\bar{x}_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3$	
	M19	$x_1x_2 \vee x_2x_3 \vee x_1x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3$	
	M20	$x_1\bar{x}_2\bar{x}_3 \vee x_2x_3$	
	M21	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(x_2 \vee x_3)$	

Muroga, Saburo, “Threshold Logic and its Applications”, Wiley-Interscience, 1971

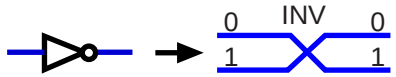
Threshold Functions and Boolean Equations

The question for each signal in an NCL expression is when should it transition to DATA. This corresponds to the Boolean Truth function which asks when it should transition to TRUE. Through this correspondence of form it is convenient to specify the transition to DATA function for each NCL signal as a Boolean sum of products equation.

The only signal inversion in NCL, in the oscillation link on the closure path forming the oscillation, is considered an integral part of the link and is not considered a part of the general logic.

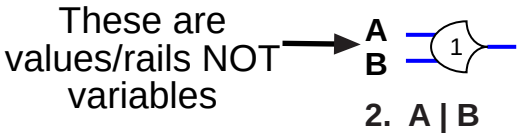


Variable value inversion in the multi-rail representation is expressed by relabling or crossing rails.

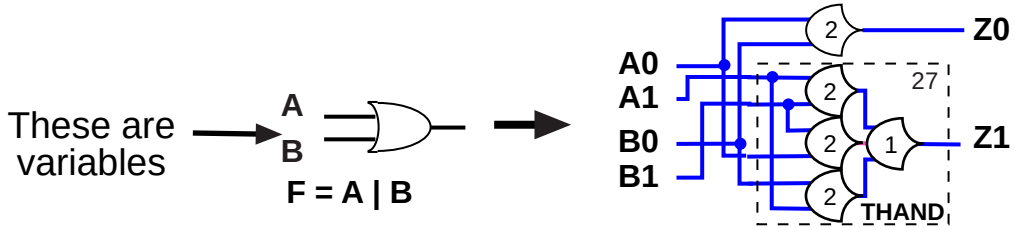


Consequently, a Boolean equation characterizing a transition to DATA function does not contain inversion so is positive.

An NCL function does not implement its associated Boolean equation in the tradition sense of a function of binary data variables.



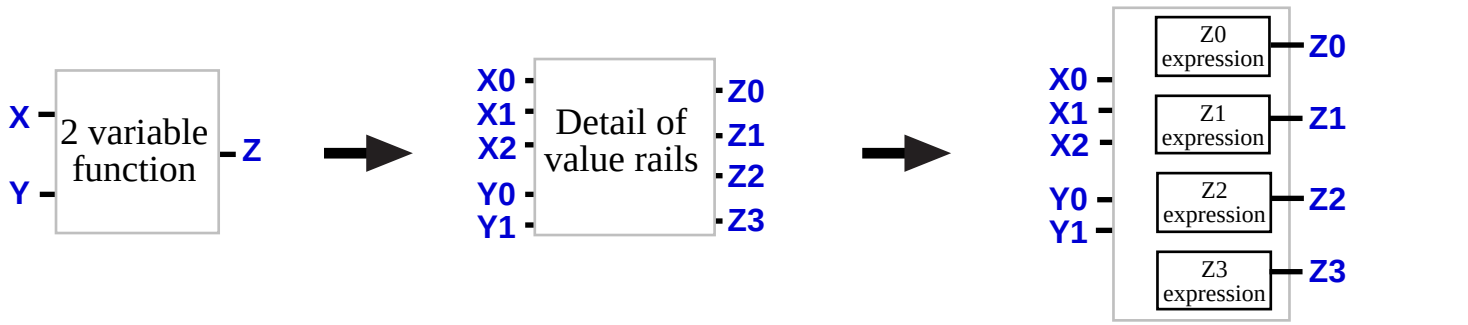
NCL 1 of 2 threshold function and its transition to DATA Boolean equation



The Boolean OR and its 2NCL combinational expression

Boolean Logic as Specification Language

An interaction of multi-rail variables can be completely characterized with a positive Boolean equation for each output rail specifying all the combinations of input variables that will transition it to DATA. The set of positive Boolean equations must span all combinations of input rails.



a. Variable level function b. value level of function c. internal structure of function

An NCL circuit can then be constructed by mapping these output equations directly to NCL functions through their associated Boolean equations.

A Multi-Value Example

A binary-trinary-quaternary adder function table

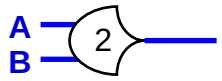
	X/0	X/1	X/2
Y/0	Z/0	Z/1	Z/2
Y/1	Z/1	Z/2	Z/3

output equations

token Y{1,0}, X{2, 1, 0}, Z{3, 2, 1, 0};

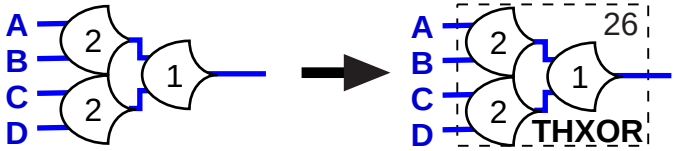
Z/0 = X/0 & Y/0;
Z/1 = X/1 & Y/0 | X/0 & Y/1;
Z/2 = X/2 & Y/0 | X/1 & Y/1;
Z/3 = X/2 & Y/1;

Z/0 = X/0 & Y/0;
Z/3 = X/2 & Y/1;



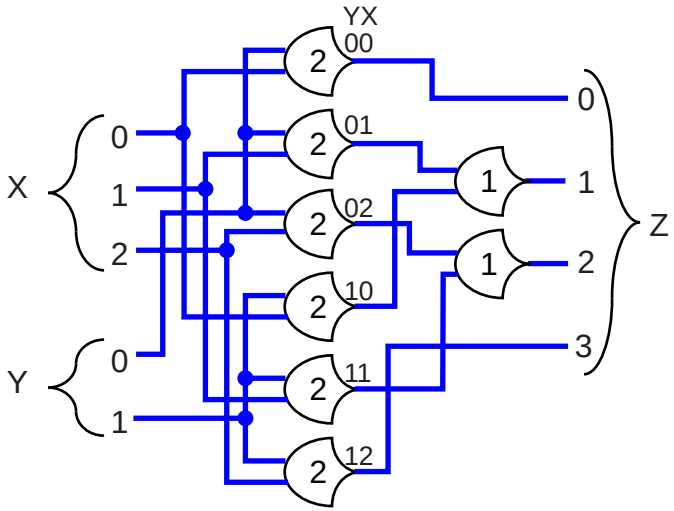
3. AB

Z/1 = X/1 & Y/0 | X/0 & Y/1;
Z/2 = X/2 & Y/0 | X/1 & Y/1;

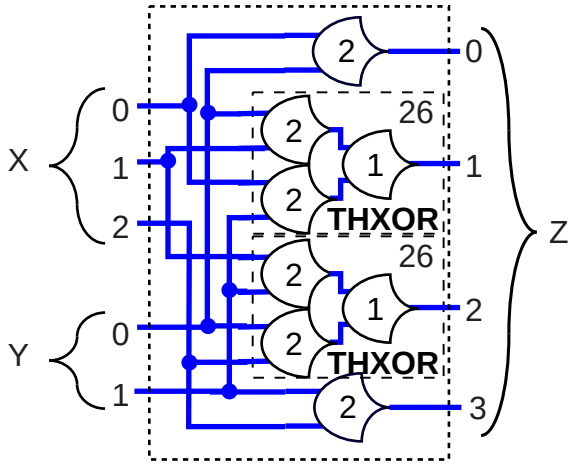


26. AB | CD

Canonical minterm circuit



Function optimized circuit



A Multi-Rail Condition

XY \ C	A	O	X
00	Z/0	Z/0	Z/0
01	Z/0	Z/1	Z/1
10	Z/0	Z/1	Z/1
11	Z/1	Z/1	Z/0

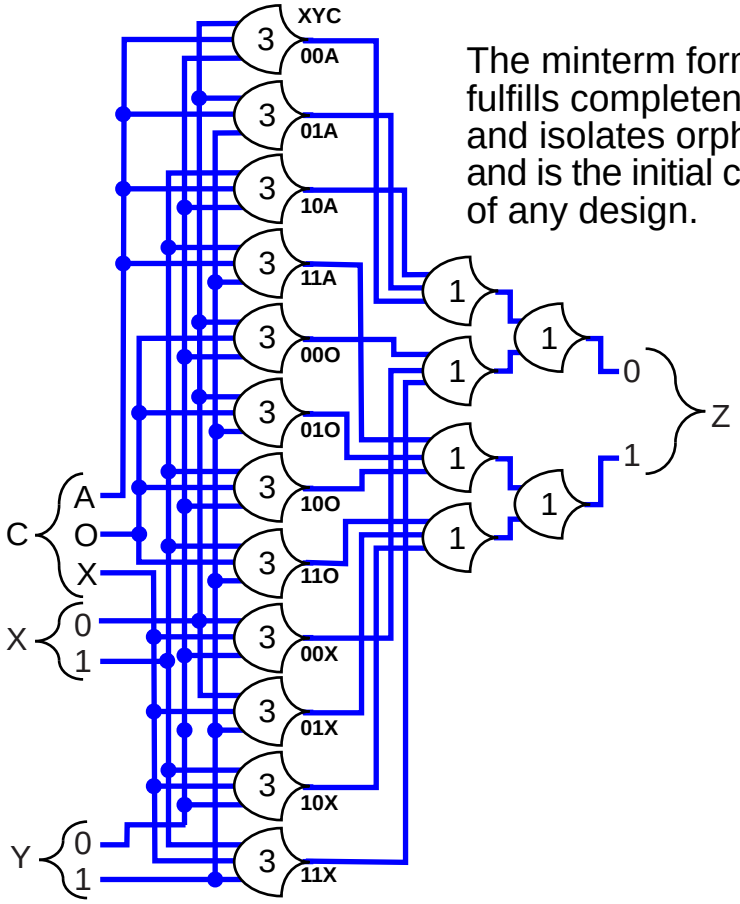
Perform AND, OR, XOR on two binary variables conditional on a command variable.

The command variable is just another multi-rail variable to combinationally interact. It has no special status or structure.

token Y{1,0}, X{1, 0}, C{A, O, X}, Z{1, 0}; output equations

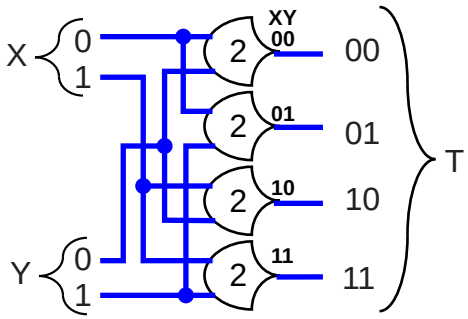
Z/0 = X/0 & Y/0 & C/A | X/0 & Y/0 & C/O | X/0 & Y/0 & C/X | X/0 & Y/1 & C/A | X/1 & Y/0 & C/A | X/1 & Y/1 & C/X
Z/1 = X/1 & Y/1 & C/A | X/1 & Y/1 & C/O | X/1 & Y/0 & C/O | X/0 & Y/1 & C/O | X/1 & Y/0 & C/X | X/0 & Y/1 & C/X

Canonical minterm circuit



The minterm form fulfills completeness and isolates orphans and is the initial circuit of any design.

factor out X Y with intermediate variable T, assign it convenient value names then substitute T in the equations and refactor



token Y{1,0}, X{1, 0}, C{A, O, X}, Z{1, 0}, T{00, 01, 10, 11};

T/00 = X/0 & Y/0;
T/01 = X/0 & Y/1;
T/10 = X/1 & Y/0;
T/11 = X/1 & Y/1;
Z/0 = T/00 & C/A | T/00 & C/O | T/00 & C/X | T/01 & C/A | T/10 & C/A | T/11 & C/X
Z/1 = T/11 & C/A | T/11 & C/O | T/10 & C/O | T/01 & C/O | T/10 & C/X | T/01 & C/X

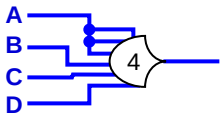
Combining Terms

Combine terms to 4 variable terms

token Y{1,0}, X{1, 0}, C{A, O, X}, Z{1, 0}, T{00, 01, 10, 11};

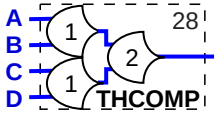
T/00 = X/0 & Y/0;
T/01 = X/0 & Y/1;
T/10 = X/1 & Y/0;
T/11 = X/1 & Y/1;
Z/0 = T/00 & C/A | T/00 & C/O | T/00 & C/X | T/01 & C/A | T/10 & C/A | T/11 & C/X
Z/1 = T/11 & C/A | T/11 & C/O | T/10 & C/O | T/01 & C/O | T/10 & C/X | T/01 & C/X

T/00 & C/A | T/00 & C/O | T/00 & C/X →



17. AB | AC | AD

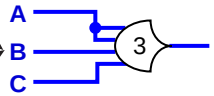
T/10 & C/O | T/01 & C/O | T/10 & C/X | T/01 & C/X →



28. AC | BC | AD | BD

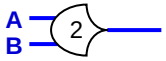
T/01 & C/A | T/10 & C/A

T/11 & C/A | T/11 & C/O

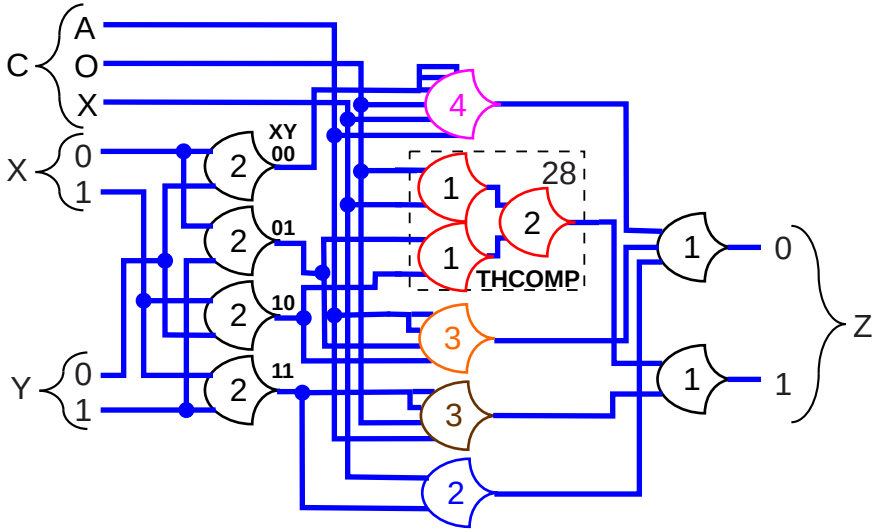


8. AB | AC

T/11 & C/X →



3. AB



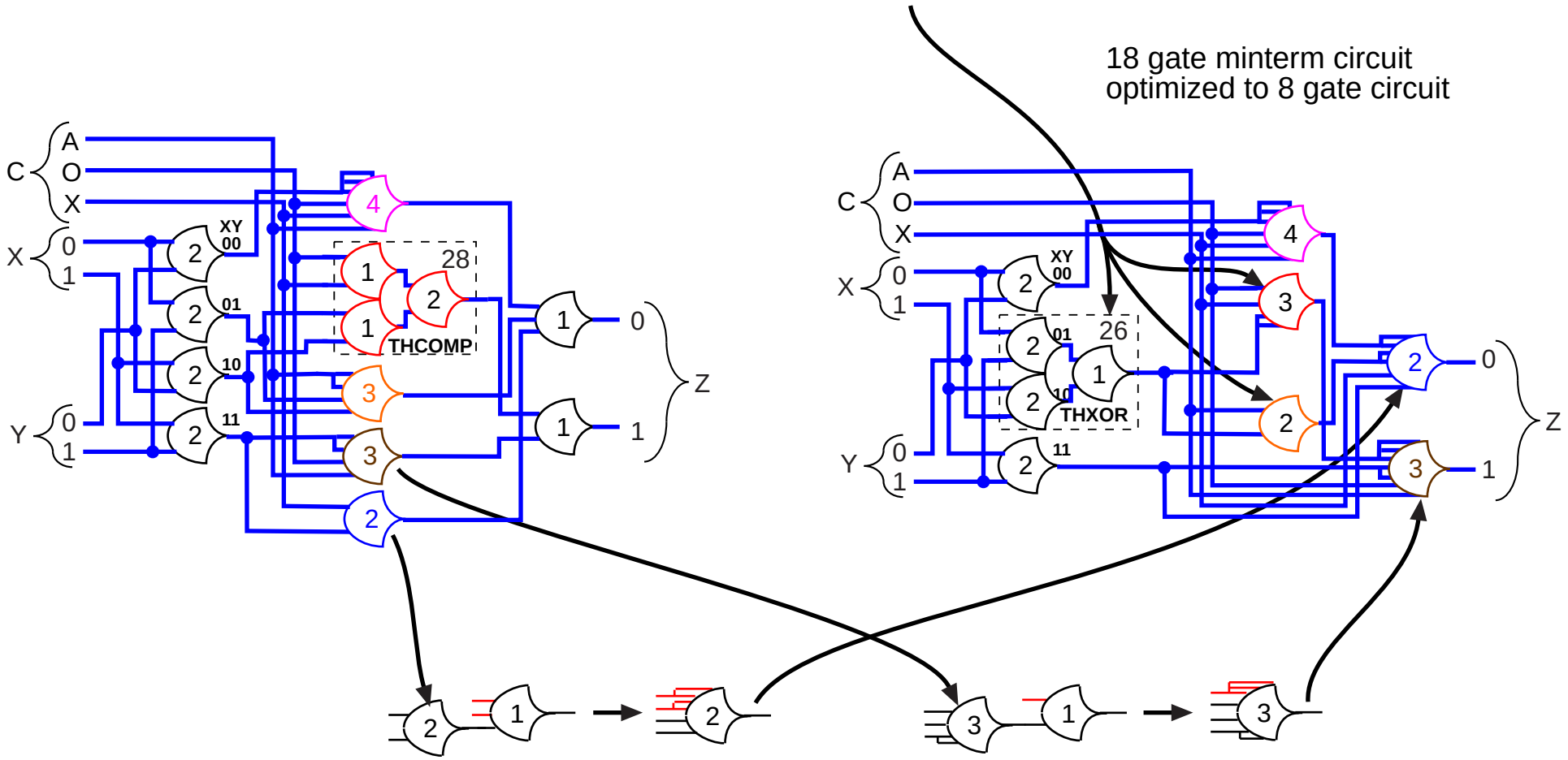
Merging Gates

T/01 and T/10 both go to all the same gates and can be combined into a single rail T/0110. Now that there is only one rail the downstream gates simplify.

token Y{1,0}, X{1, 0}, C{A, O, X}, Z{1, 0}, T{00, 0110, 11};

T/00 = X/0 & Y/0;
T/0110 = X/0 & Y/1 | X/1 & Y/0;
T/11 = X/1 & Y/1;
Z/0 = T/00 & C/A | T/00 & C/O | T/00 & C/X | T/0110 & C/A | T/11 & C/X
Z/1 = T/11 & C/A | T/11 & C/O | T/0110 & C/O | T/0110 & C/X

18 gate minterm circuit
optimized to 8 gate circuit



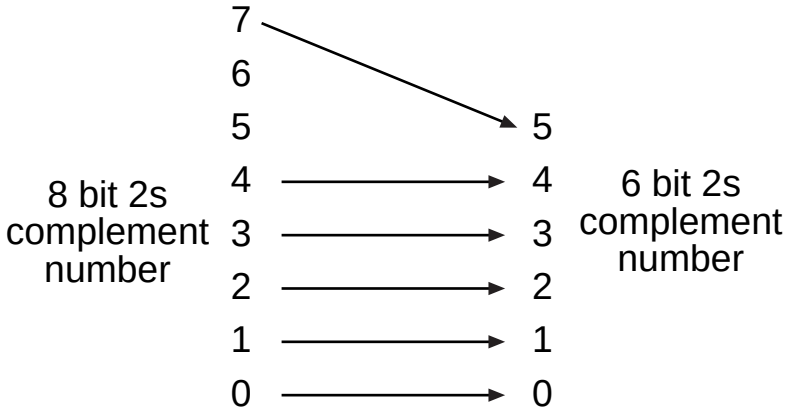
Merge gates into 4 variable functions

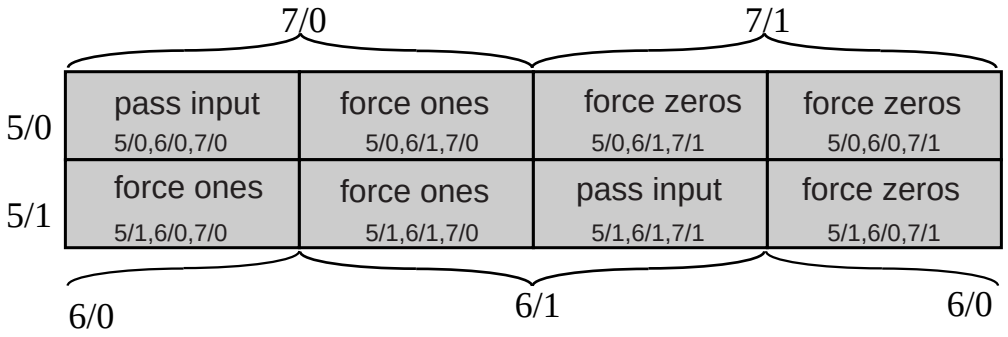
Twos Complement 8 Bit to 6 Bit Clipper

If sign bit 7 is 1 and either bit 5 or bit 6 is 0 then the 8 bit number is a negative number larger that 6 bits. Force to smallest negative 6 bit number (magnitude all zeros).

If sign bit 7 is 0 and either bit 5 or bit 6 is 1 then the 8 bit number is a positive number larger that 6 bits. Force to largest positive 6 bit number (magnitude all ones).

If sign bit 7 and bits 5 and bit 6 match then the magnitude of the number is within 6 bits. Pass the magnitude bits.



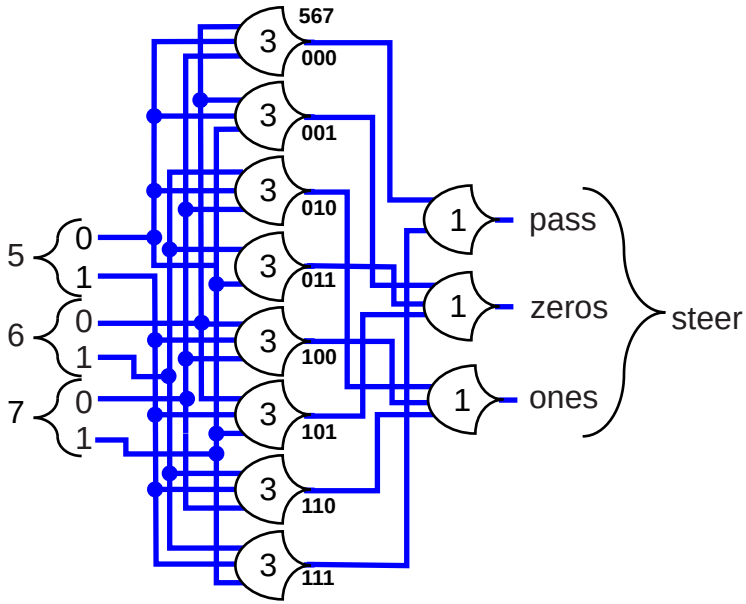


output equations

token steer{pass, ones, zeros}, 5{1, 0}, 6{1, 0}, 7{1, 0};

steer/pass = 5/0 & 6/0 & 7/0 | 5/1 & 6/1 & 7/1;
steer/ones = 5/1 & 6/0 & 7/1 | 5/0 & 6/0 & 7/1 | 5/0 & 6/1 & 7/1;
steer/zeros = 5/1 & 6/0 & 7/0 | 5/1 & 6/1 & 7/0 | 5/0 & 6/1 & 7/0;

Canonical minterm circuit



Sum Partitioned Factoring

steer/pass = 5/0 & 6/0 & 7/0 | 5/1 & 6/1 & 7/1;

steer/ones = 5/1 & 6/0 & 7/1 | 5/0 & 6/0 & 7/1 | 5/0 & 6/1 & 7/1;

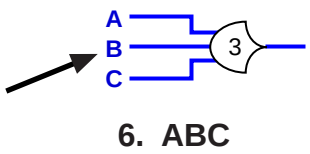
steer/zeros = 5/1 & 6/0 & 7/0 | 5/1 & 6/1 & 7/0 | 5/0 & 6/1 & 7/0;

5/1 & 6/1 & 7/1

5/0 & 6/0 & 7/0

5/1 & 6/0 & 7/1

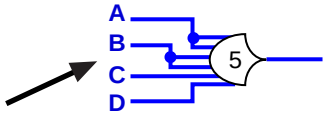
5/0 & 6/1 & 7/0



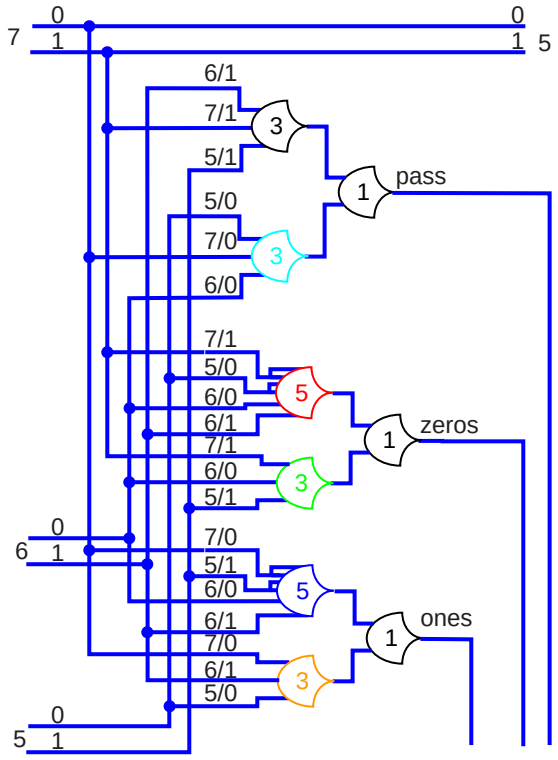
6. ABC

5/0 & 6/0 & 7/1 | 5/0 & 6/1 & 7/1

5/1 & 6/0 & 7/0 | 5/1 & 6/1 & 7/0

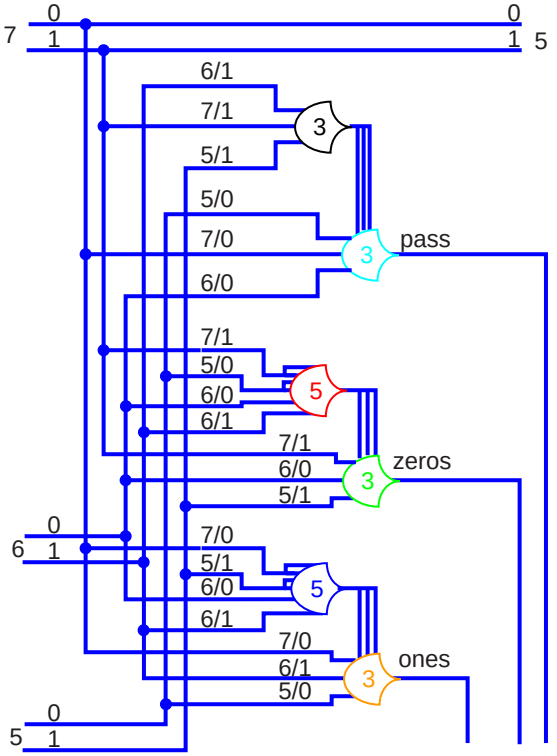


21. ABC | ABD



directly mapped expression

merge operators



merged operators

11 gates into 6 gates

Product Partitioned Factoring

Combine bit 5 and bit 6

5 \ 6	6/0	6/1
5/0	00	10
5/1	01	11

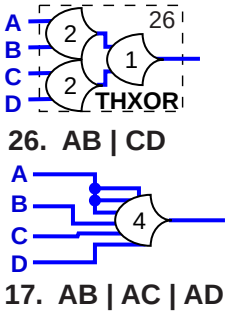
T

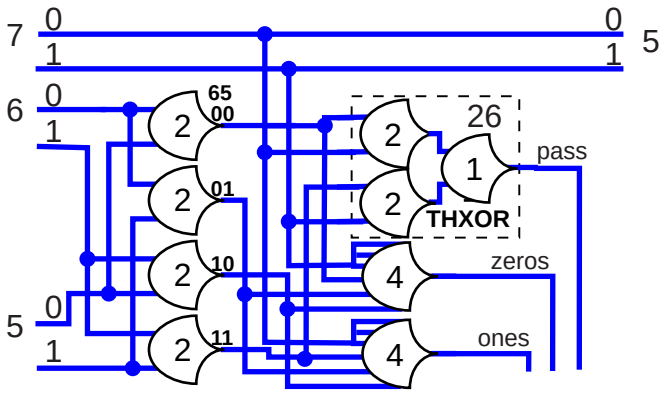
Combine bit 7

T	7/0	7/1
T/00	pass input	force zeros
T/01	force ones	force zeros
T/10	force ones	force zeros
T/11	force ones	pass input

token steer{pass, ones, zeros}, 5{1, 0}, 6{1, 0}, 7{1, 0}, T{00, 01, 10, 11};

T/00 = 5/0 & 6/0;
T/01 = 5/1 & 6/0;
T/10 = 5/0 & 6/1;
T/11 = 5/1 & 6/1;
steer/pass = T/00 & 7/0 | T/11 & 7/1;
steer/ones = T/10 & 7/1 | T/00 & 7/1 | T/01 & 7/1;
steer/zeros = T/10 & 7/0 | T/11 & 7/0 | T/01 & 7/0;

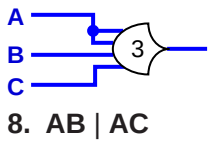


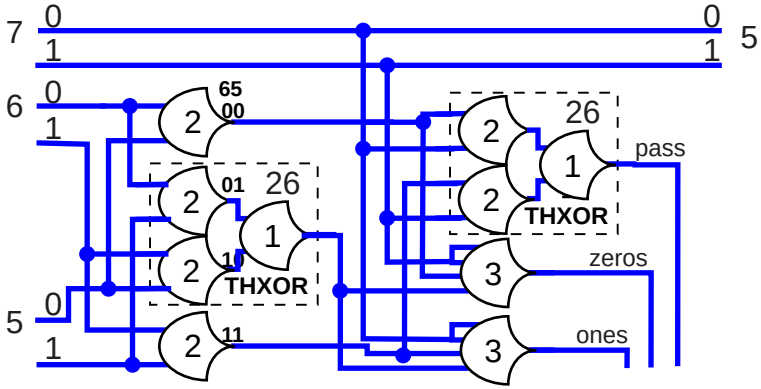


As is often the case 7/01 and 7/10 go to all the same places and can be turned into a single rail simplifying downstream gates

token steer{pass, ones, zeros}, 5{1, 0}, 6{1, 0}, 7{1, 0}, T{00, 0110, 11};

T/00 = 5/0 & 6/0;
T/0110 = 5/1 & 6/0 | 5/0 & 6/1;
T/11 = 5/1 & 6/1;
steer/pass = T/00 & 7/0 | T/11 & 7/1;
steer/ones = T/0110 & 7/1 | T/00 & 7/1;
steer/zeros = T/0110 & 7/0 | T/11 & 7/0;





11 gates into 6 gates

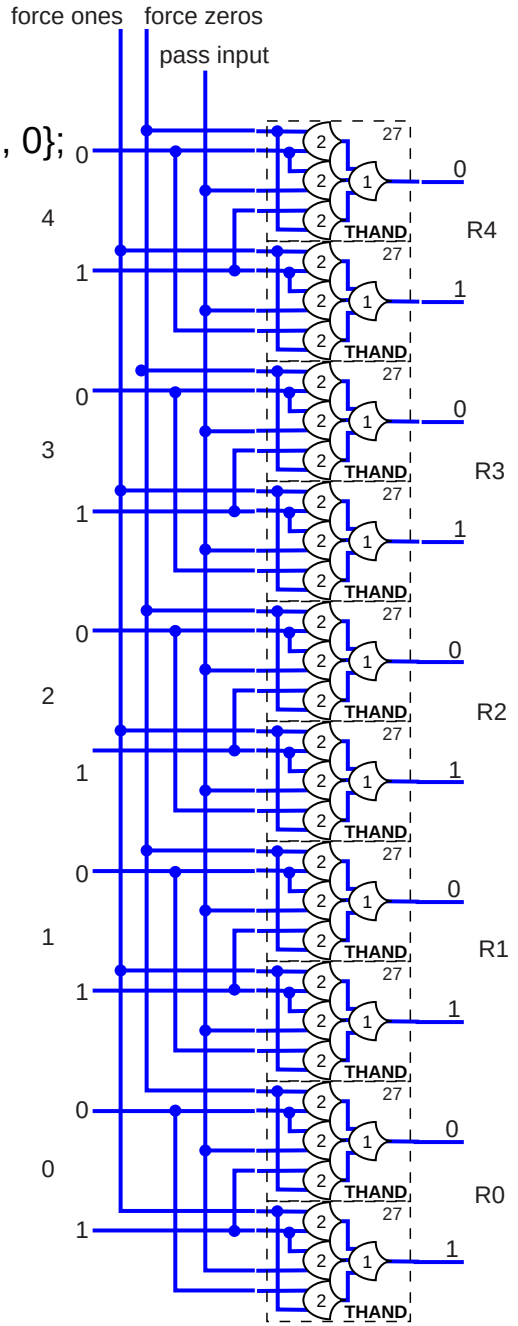
Clipper Data Path

n	steer /pass	steer /zeros	steer /ones	R
/0	/0	/0	/1	
/1	/1	/0	/1	

token steer{pass, ones, zeros}, 0{1, 0}, 1{1, 0}, 2{1, 0}, 3{1, 0}, 4{1, 0}, 5{1, 0}, 6{1, 0}, 7{1, 0};
token R0{1, 0}, R1{1, 0}, R2{1, 0}, R3{1, 0}, R4{1, 0}, R5{1, 0};

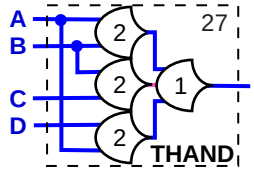
steer/pass = 5/0 & 6/0 & 7/0 | 5/1 & 6/1 & 7/1;
steer/ones = 5/1 & 6/0 & 7/1 | 5/0 & 6/0 & 7/1 | 5/0 & 6/1 & 7/1;
steer/zeros = 5/1 & 6/0 & 7/0 | 5/1 & 6/1 & 7/0 | 5/0 & 6/1 & 7/0;
R5/0 = 7/0;
R5/1 = 7/1;
R4/0 = steer/pass & 4/0 | steer/zeros & (4/0 | 4/1);
R4/1 = steer/pass & 4/1 | steer/ones & (4/0 | 4/1);
R3/0 = steer/pass & 3/0 | steer/zeros & (3/0 | 3/1);
R3/1 = steer/pass & 3/1 | steer/ones & (3/0 | 3/1);
R2/0 = steer/pass & 2/0 | steer/zeros & (2/0 | 2/1);
R2/1 = steer/pass & 2/1 | steer/ones & (2/0 | 2/1);
R1/0 = steer/pass & 1/0 | steer/zeros & (1/0 | 1/1);
R1/1 = steer/pass & 1/1 | steer/ones & (1/0 | 1/1);
R0/0 = steer/pass & 0/0 | steer/zeros & (0/0 | 0/1);
R0/1 = steer/pass & 0/1 | steer/ones & (0/0 | 0/1);

The force rails have to wait on the input data flow to insure completeness. A baton is being handed off and we have to make sure we pick up the baton



C B A D

n/0 = steer/pass & n/0 | steer/zeros & n/0 | steer/zeros & n/1;
n/1 = steer/pass & n/1 | steer/ones & n/0 | steer/ones & n/1;



27. AB | BC | AD

Do Not Optimize Boolean Expression

Optimizing the Boolean expression before mapping to NCL destroys the completeness relations and orphan isolation.