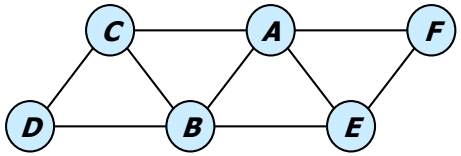
Counting solutions on AND/OR search graphs

ICS-275

Winter 2016

(2007 AIJ paper)

#CSP – OR Search Tree DFS Traversal

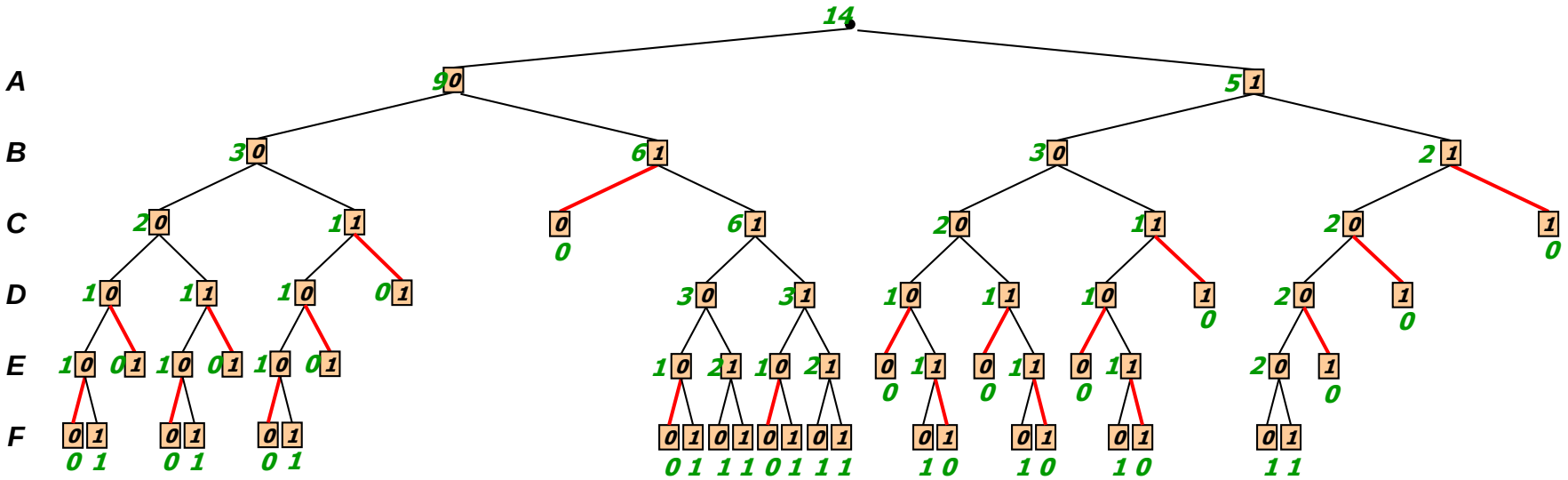


A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

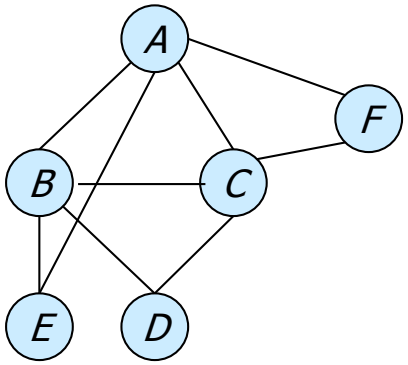


Value of node = number of solutions below it

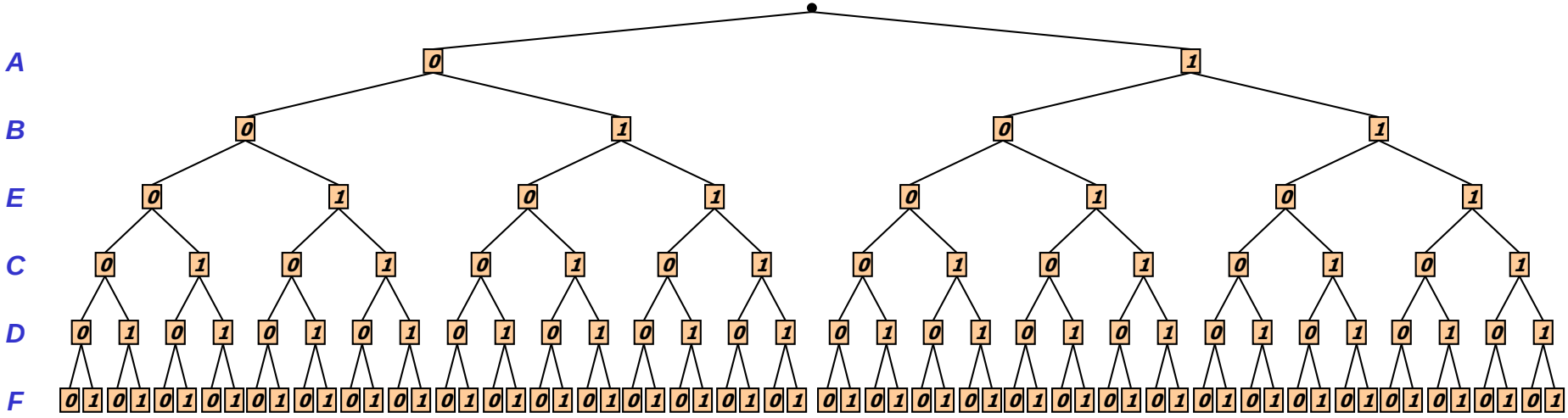
Road Map: Search in Graphical Models

- Improving search by bounded-inference in branching ahead
- Improving search by looking-back
- The alternative AND/OR search space

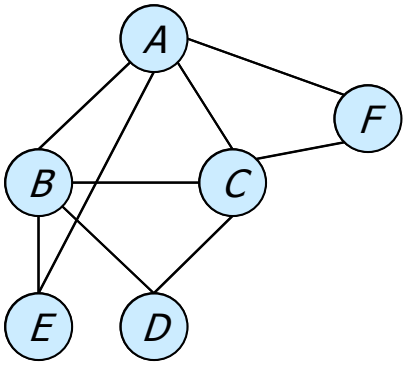
OR Search Space



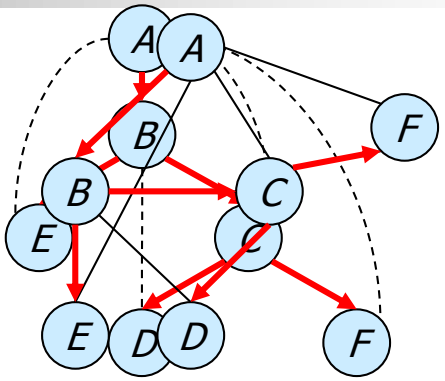
Ordering: A B E C D F



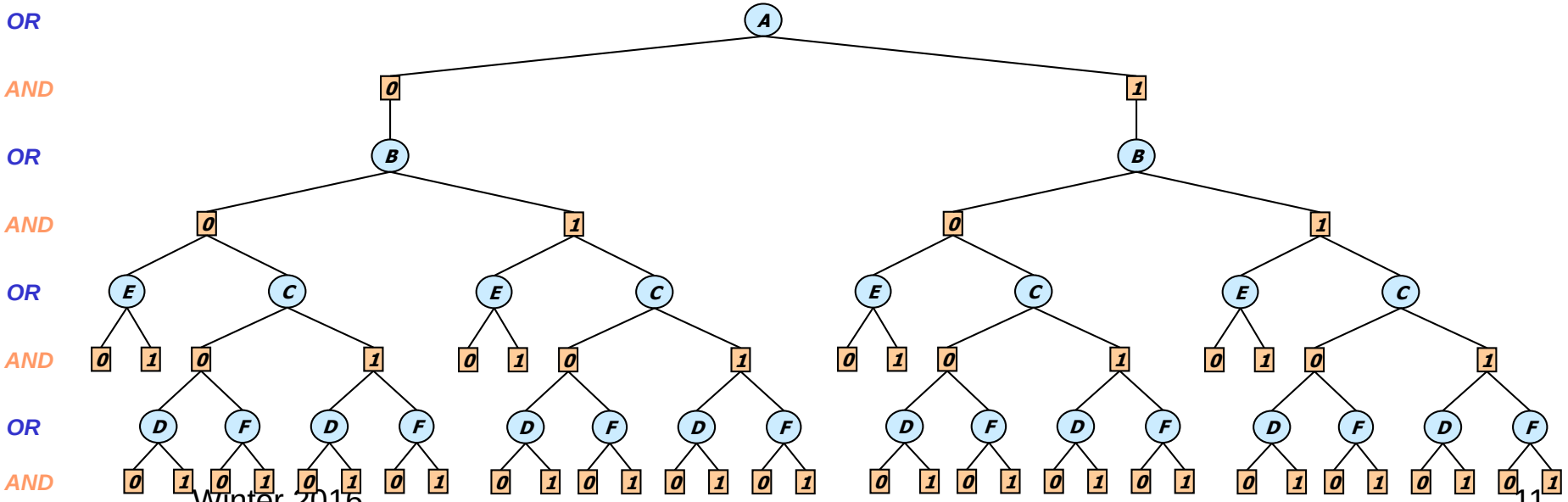
AND/OR Search Space



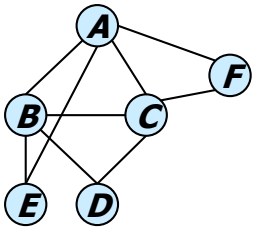
Primal graph

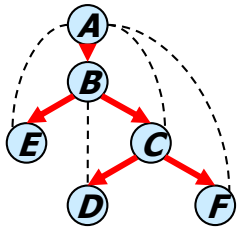


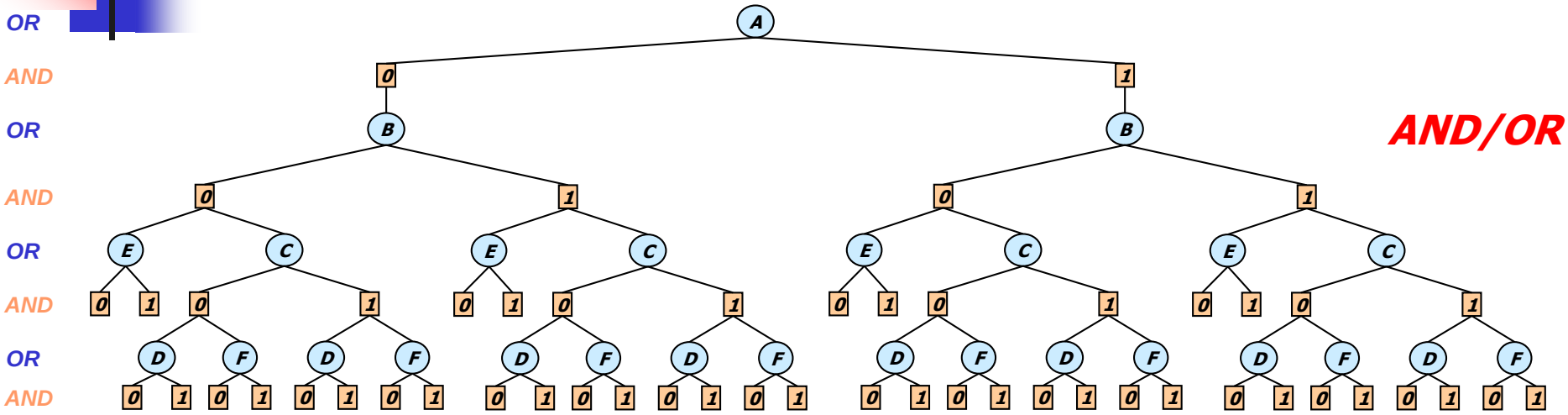
DFS tree



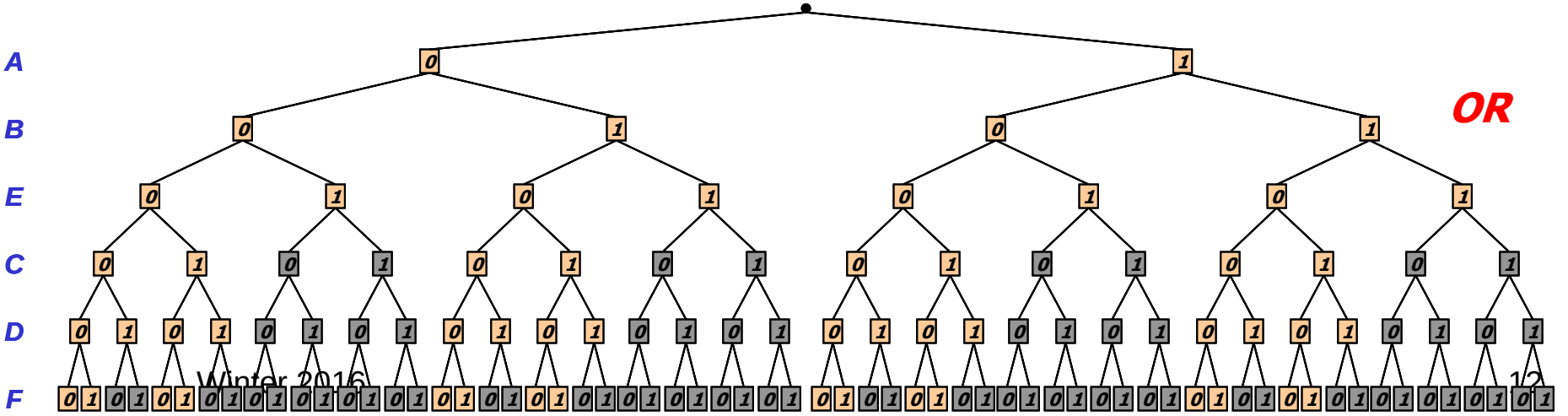
AND/OR vs. OR





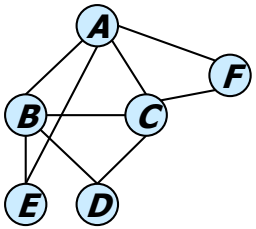


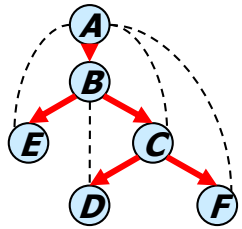
AND/OR size: exp(4), OR size exp(6)

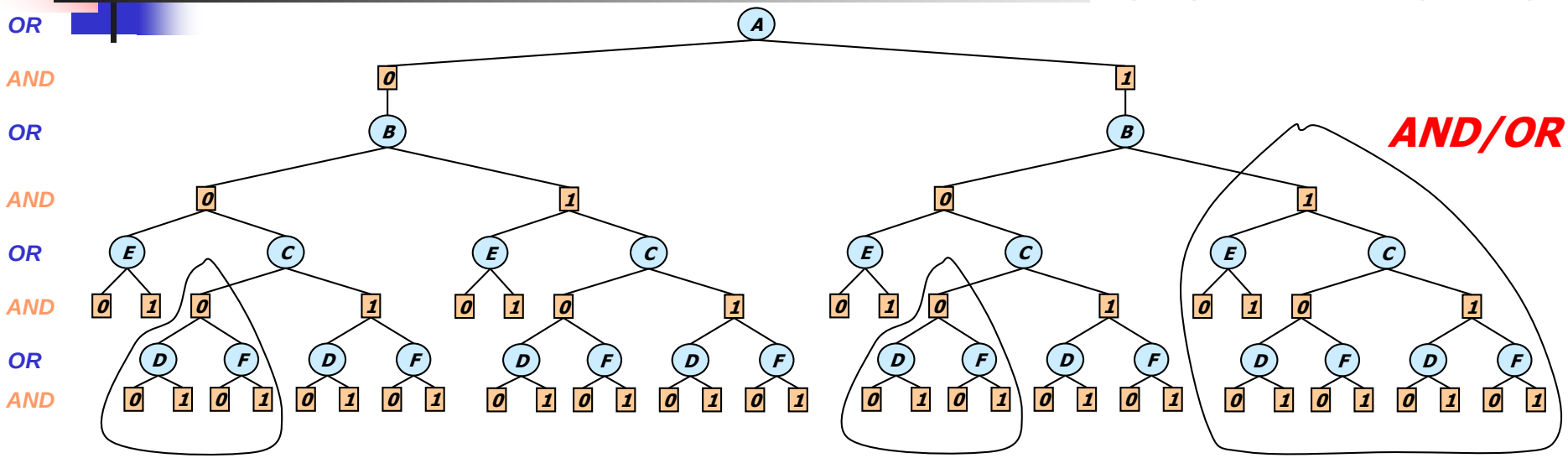


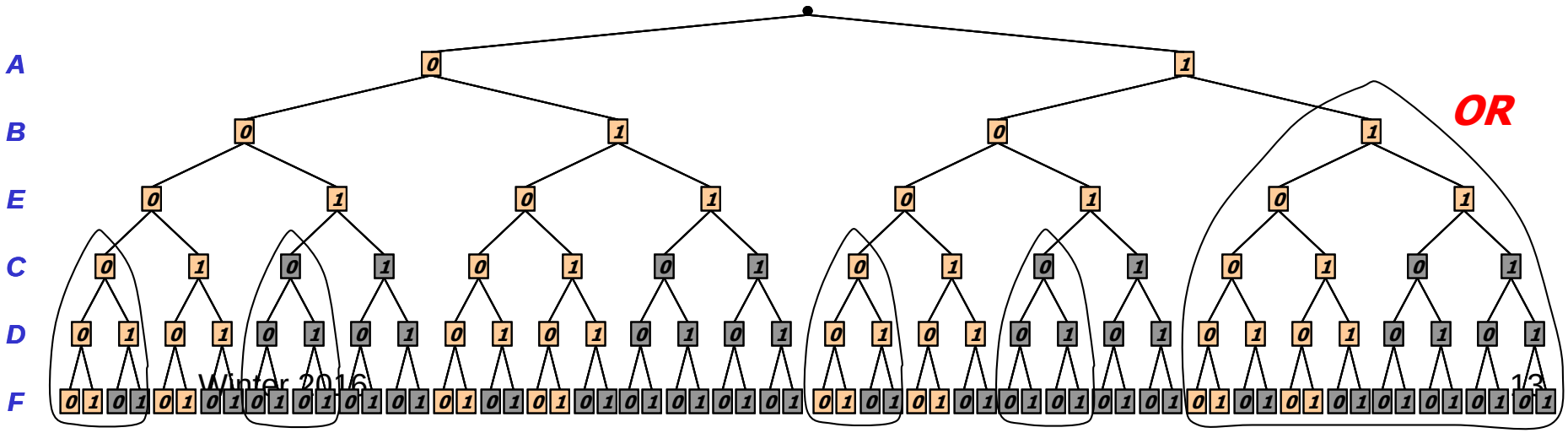
AND/OR vs. OR

No-goods
(A=1,B=1)
(B=0,C=0)



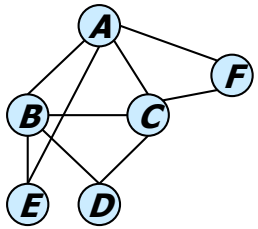


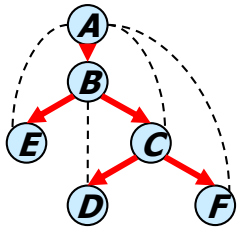


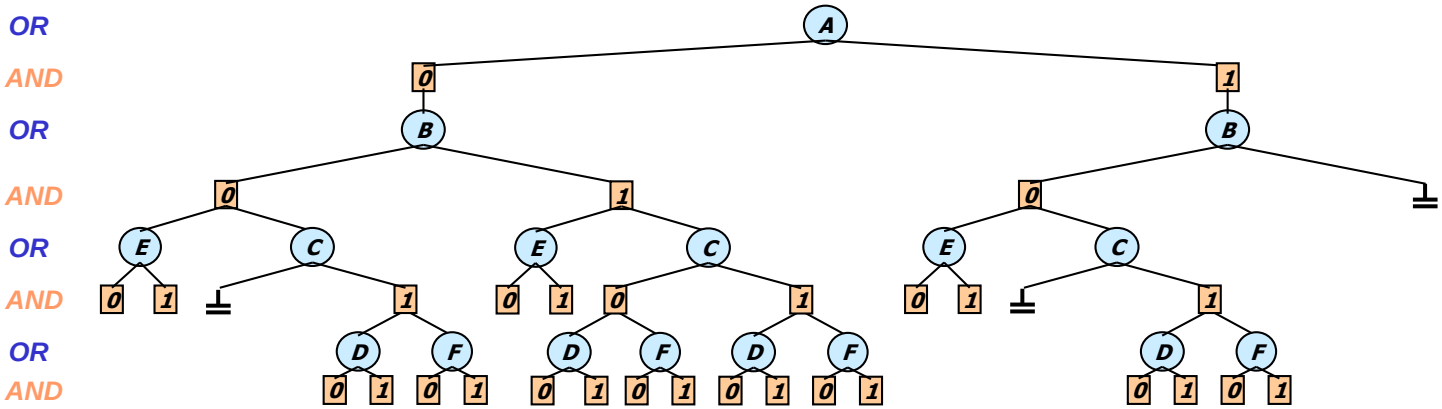


AND/OR vs. OR

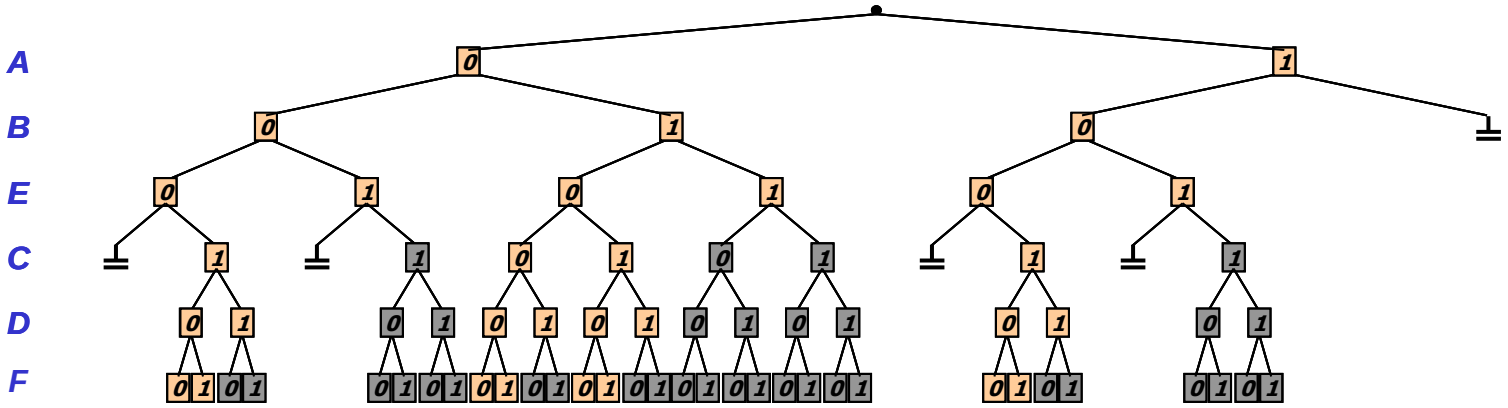
(A=1,B=1)
(B=0,C=0)







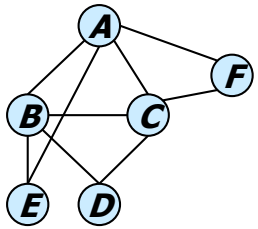
AND/OR

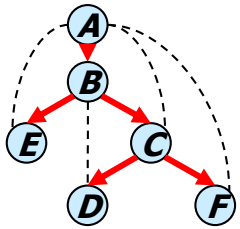


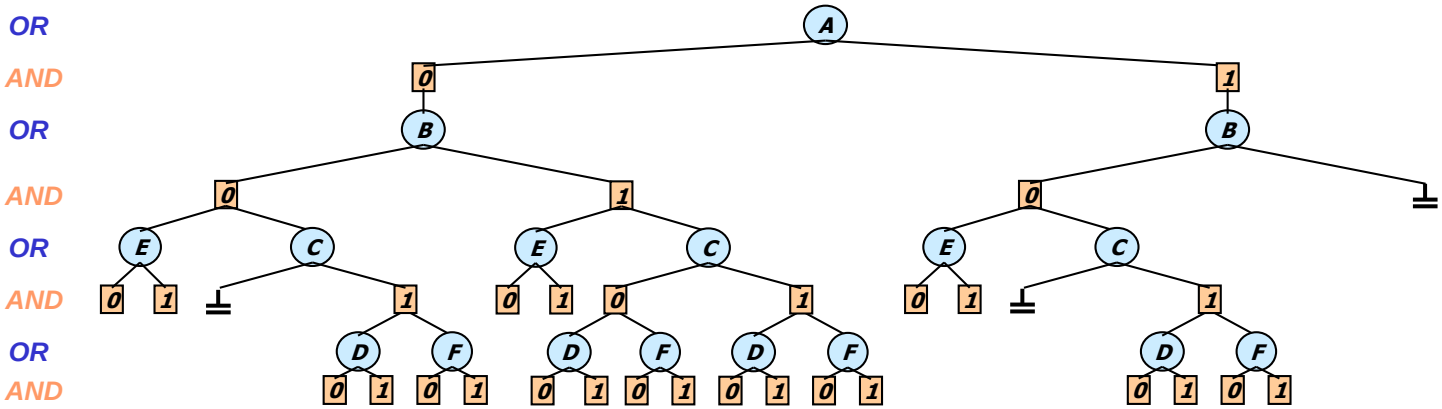
OR

AND/OR vs. OR

(A=1,B=1)
(B=0,C=0)

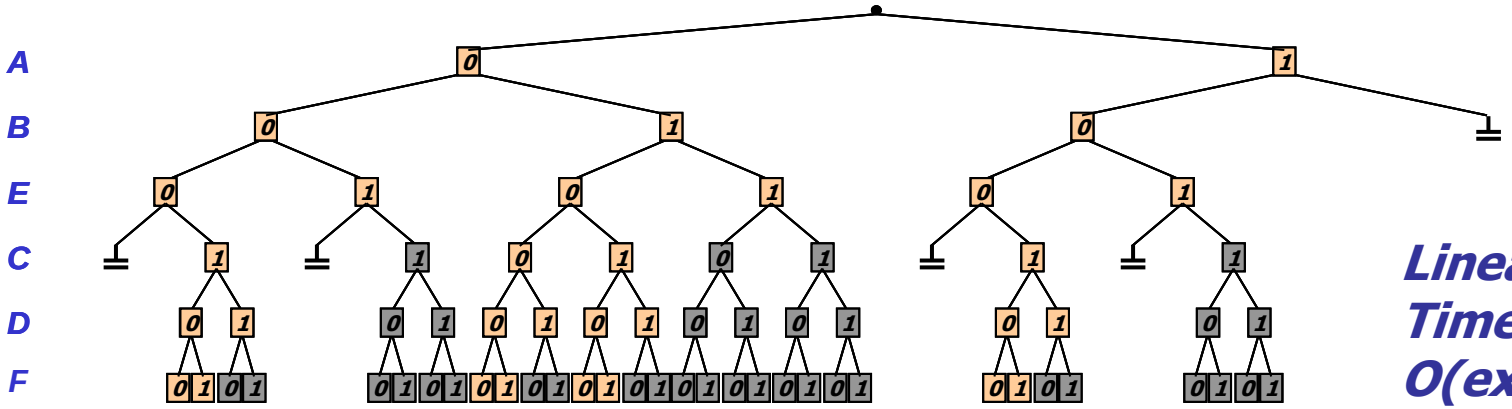






AND/OR

Space: linear
Time:
O(exp(m))
O(w* log n)



OR

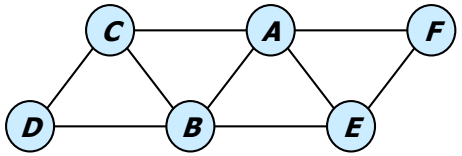
Linear space,
Time:
O(exp(n))

AND/OR vs. OR Spaces

width	depth	OR space		AND/OR space		
		Time (sec.)	Nodes	Time (sec.)	AND nodes	OR nodes
5	10	3.15	2,097,150	0.03	**10,494**	5,247
4	9	3.13	2,097,150	0.01	**5,102**	2,551
5	10	3.12	2,097,150	0.03	**8,926**	4,463
4	10	3.12	2,097,150	0.02	**7,806**	3,903
5	13	3.11	2,097,150	0.10	**36,510**	18,255

Random graphs with 20 nodes, 20 edges and 2 values per node

#CSP – AND/OR Search Tree

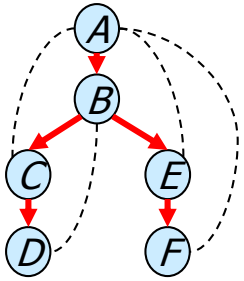


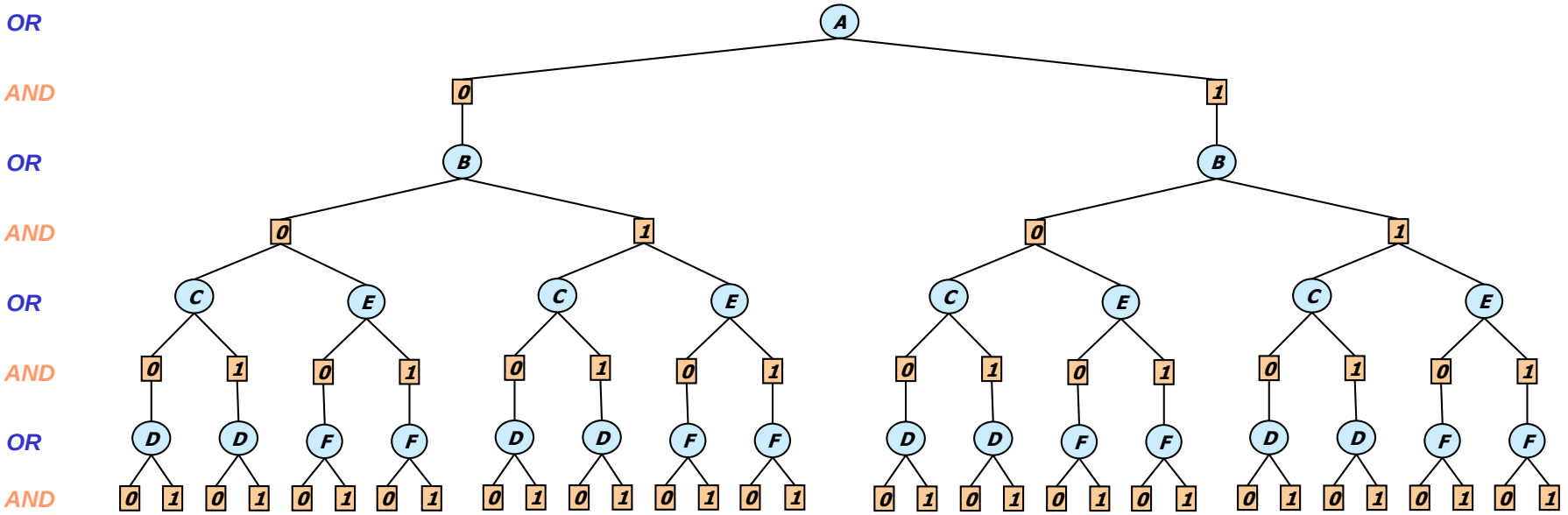
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

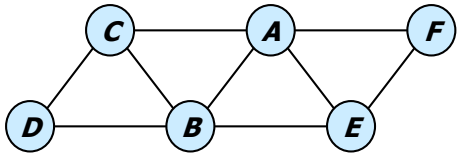
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR Tree DFS

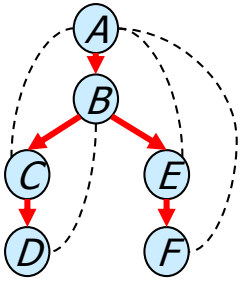


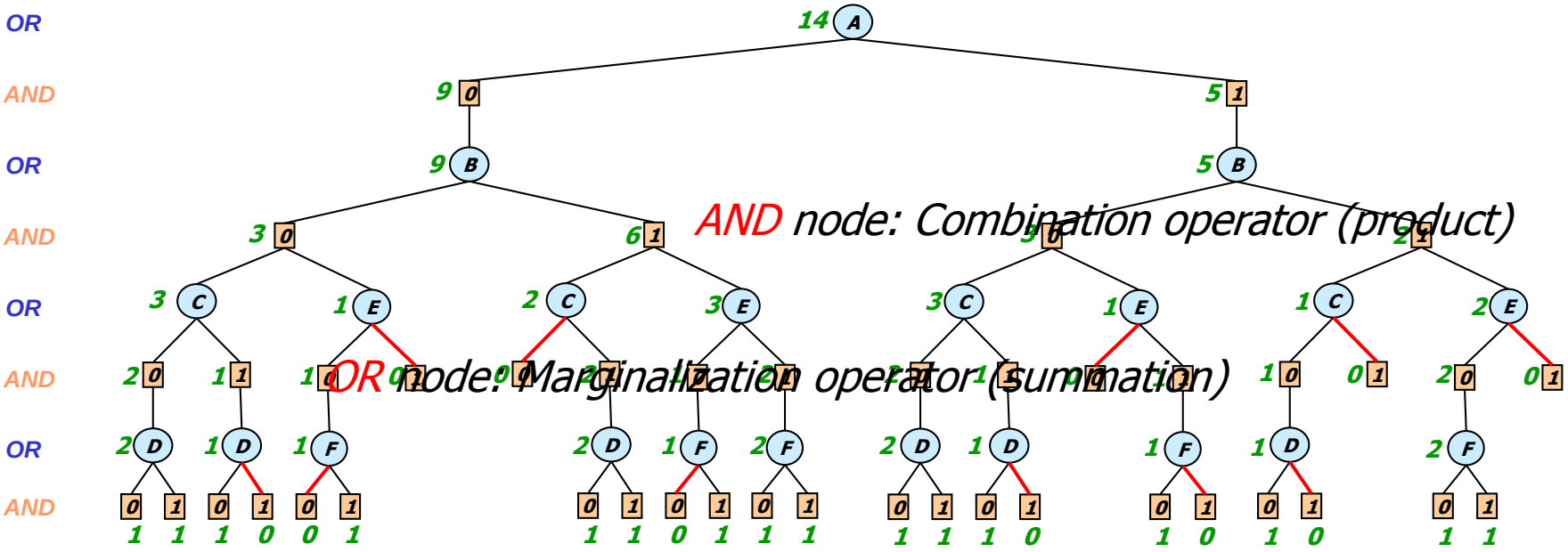
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

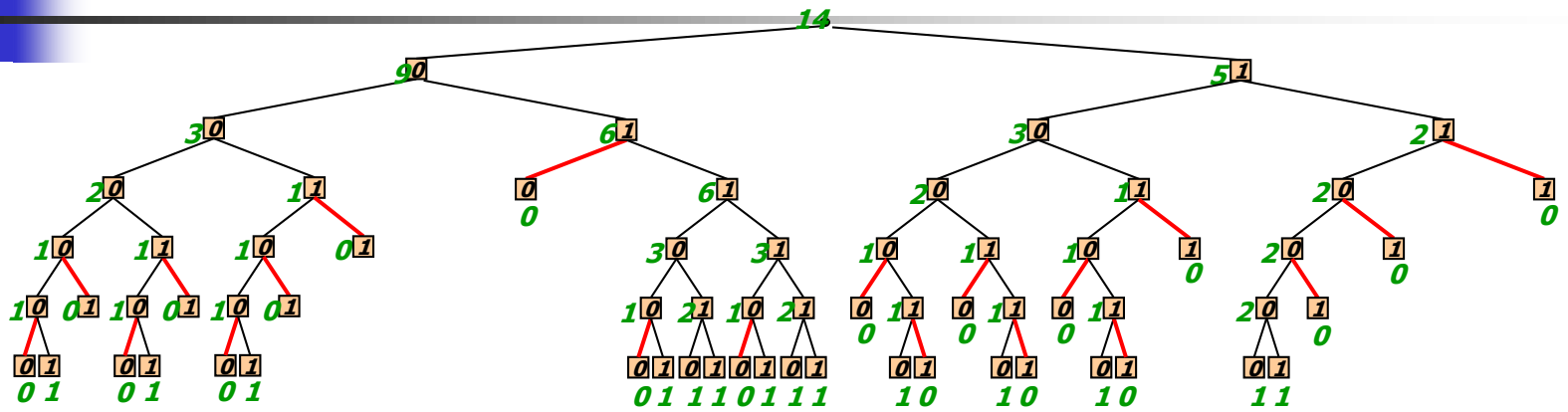
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

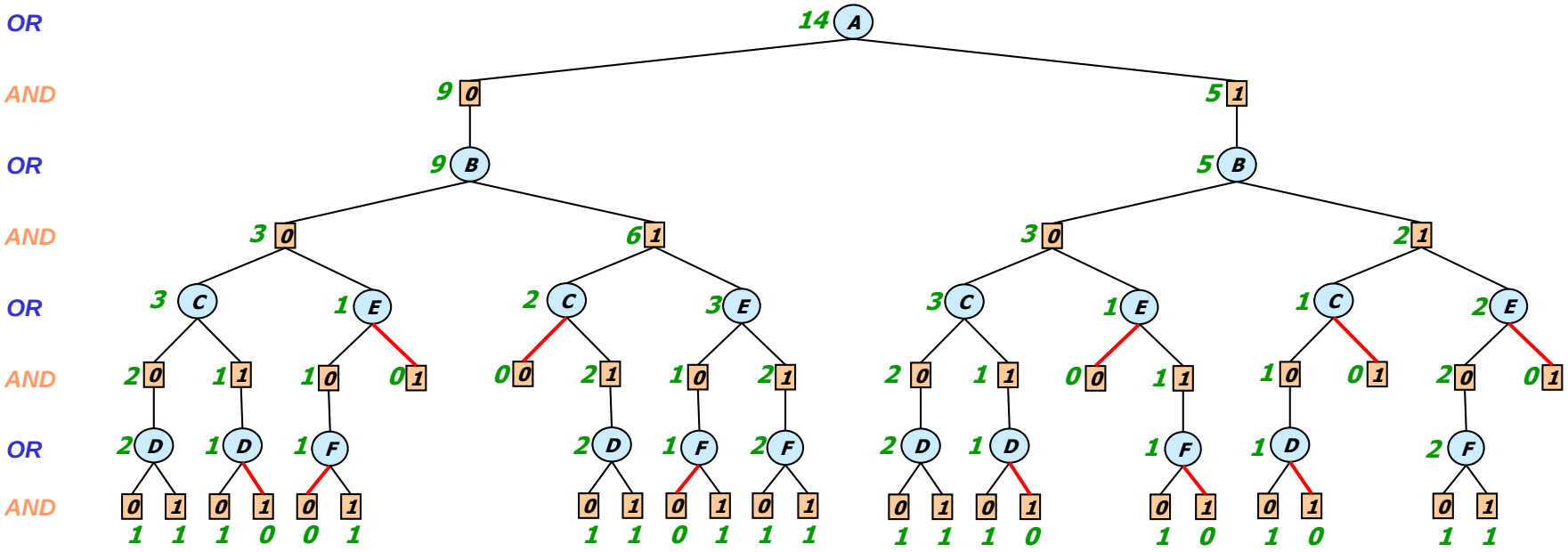
A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





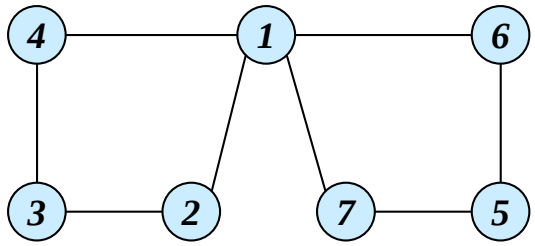
#CSP – Or vs. AND/OR DFS





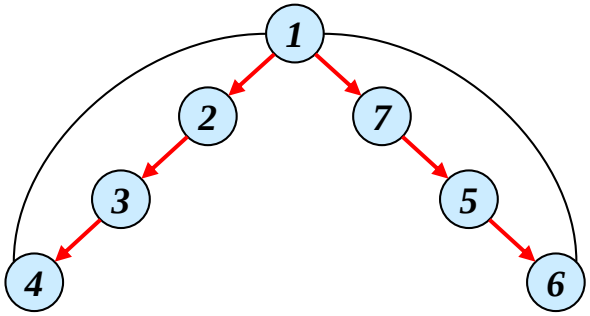
Pseudo-Trees

(Freuder 85, Bayardo 95, Bodlaender and Gilbert, 91)

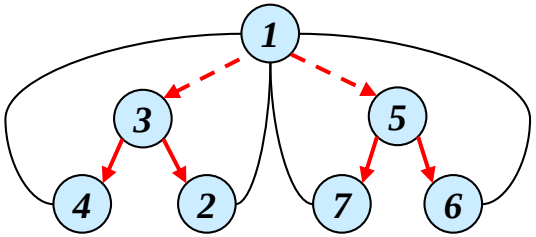


(a) Graph

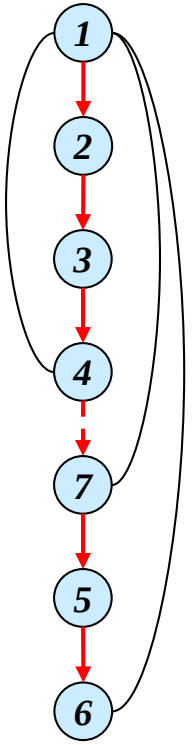
$$m <= w^* \log n$$



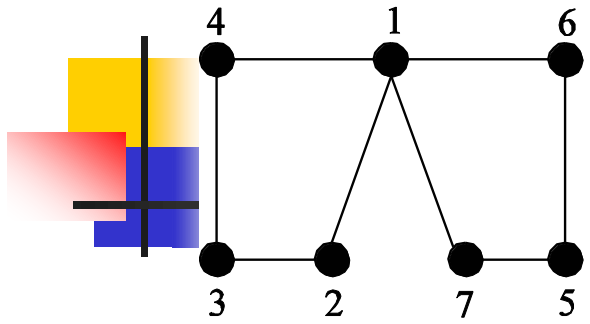
(b) DFS tree
depth=3

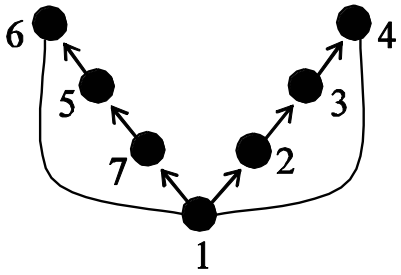


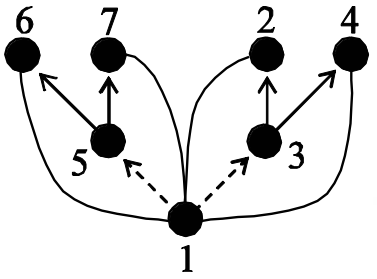
(c) pseudo- tree
depth=2



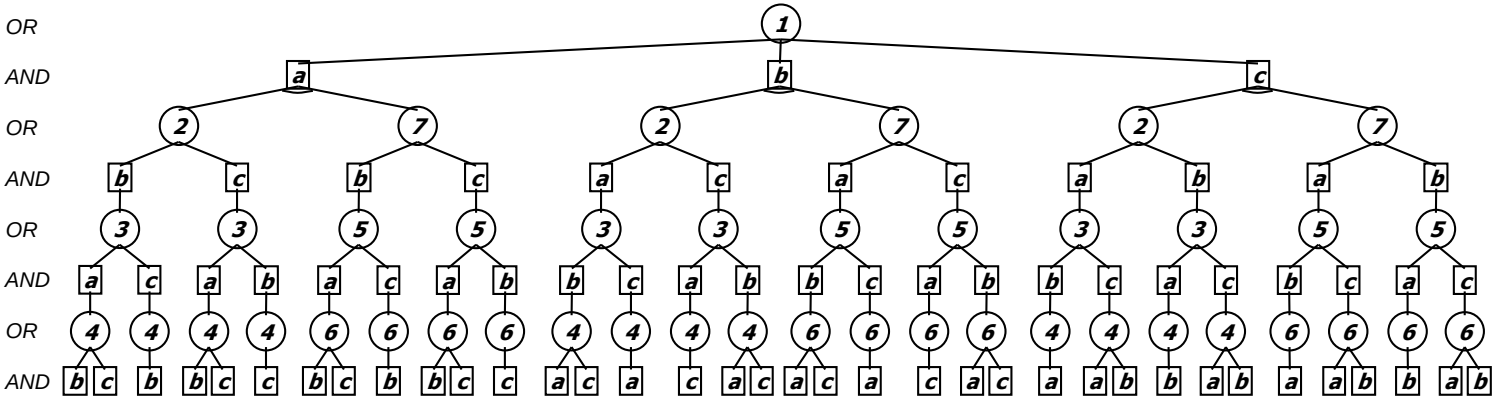
(d) Chain
depth=6

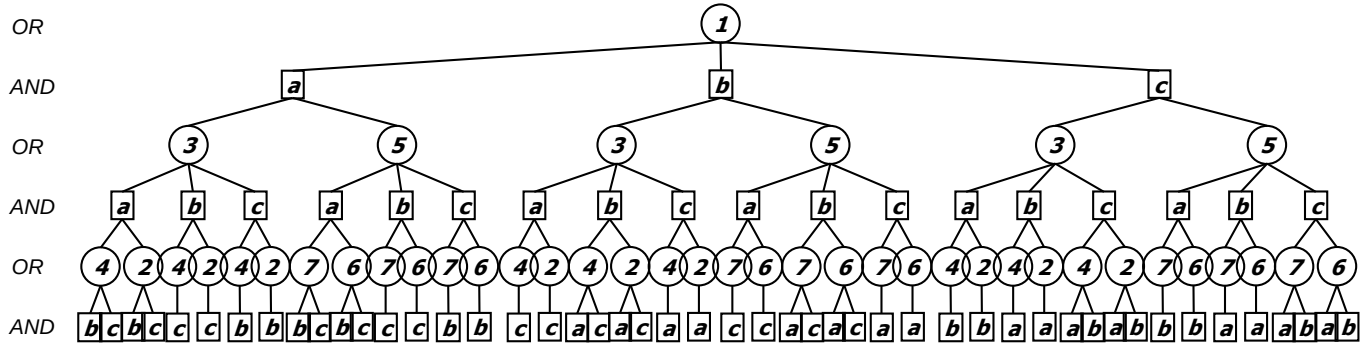

4
1
6
3
2
7
5


6
4
5
3
7
2
1


6
7
2
4
5
3
1

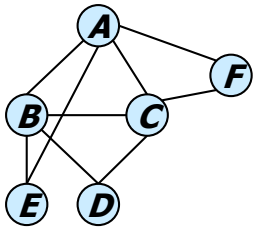
(a) **(b)** **(c)**

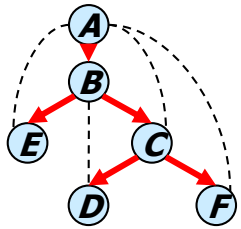

OR
AND
OR
AND
OR
AND
OR
AND

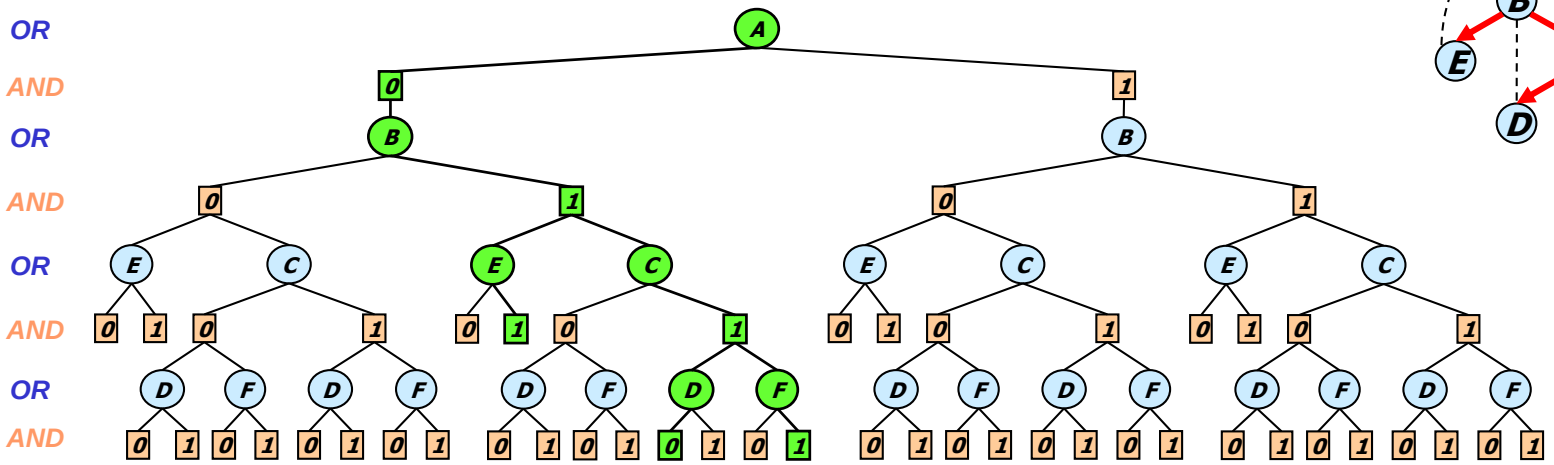

OR
AND
OR
AND
OR
AND

AND/OR search tree for graphical models

- **The AND/OR search tree of R relative to a tree, T, has:**
  - **Alternating levels of: OR nodes (variables) and AND nodes (values)**
- **Successor function:**
  - **The successors of OR nodes X are all its consistent values along its path**
  - **The successors of AND <X,v> are all X child variables in T**
- **A solution is a consistent subtree**
- **Task: compute the value of the root node**

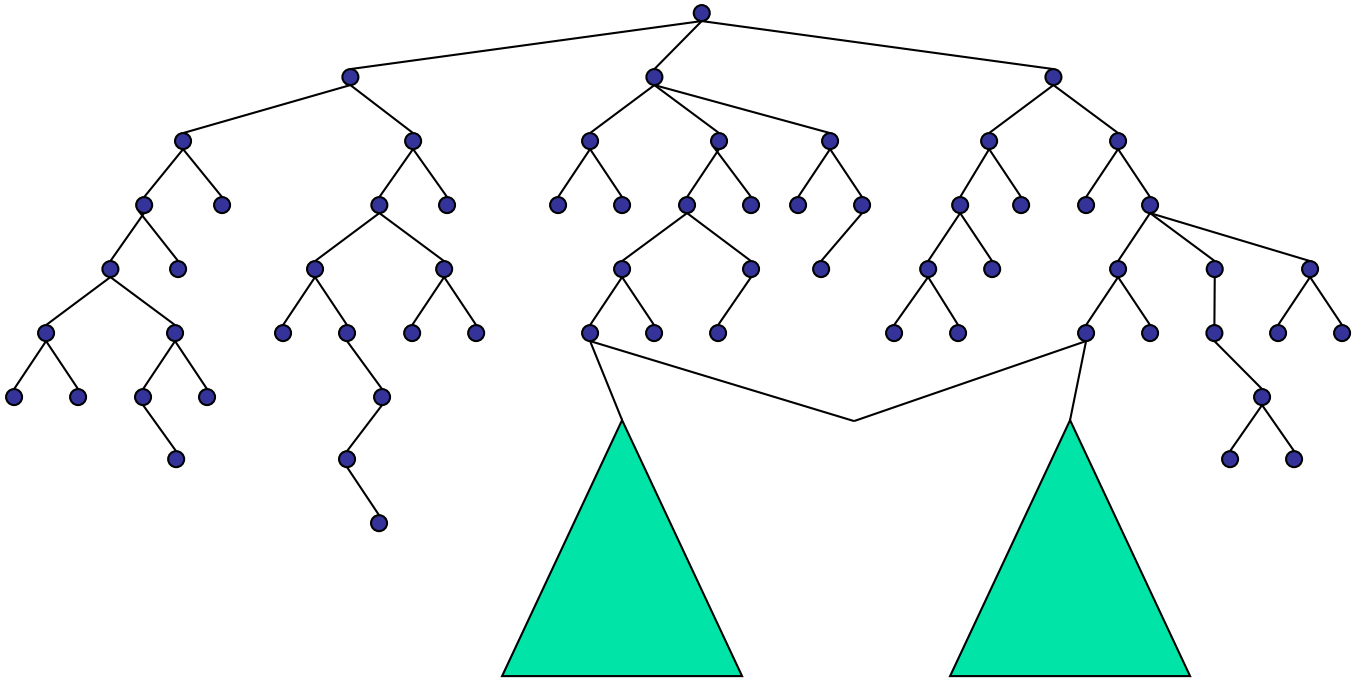






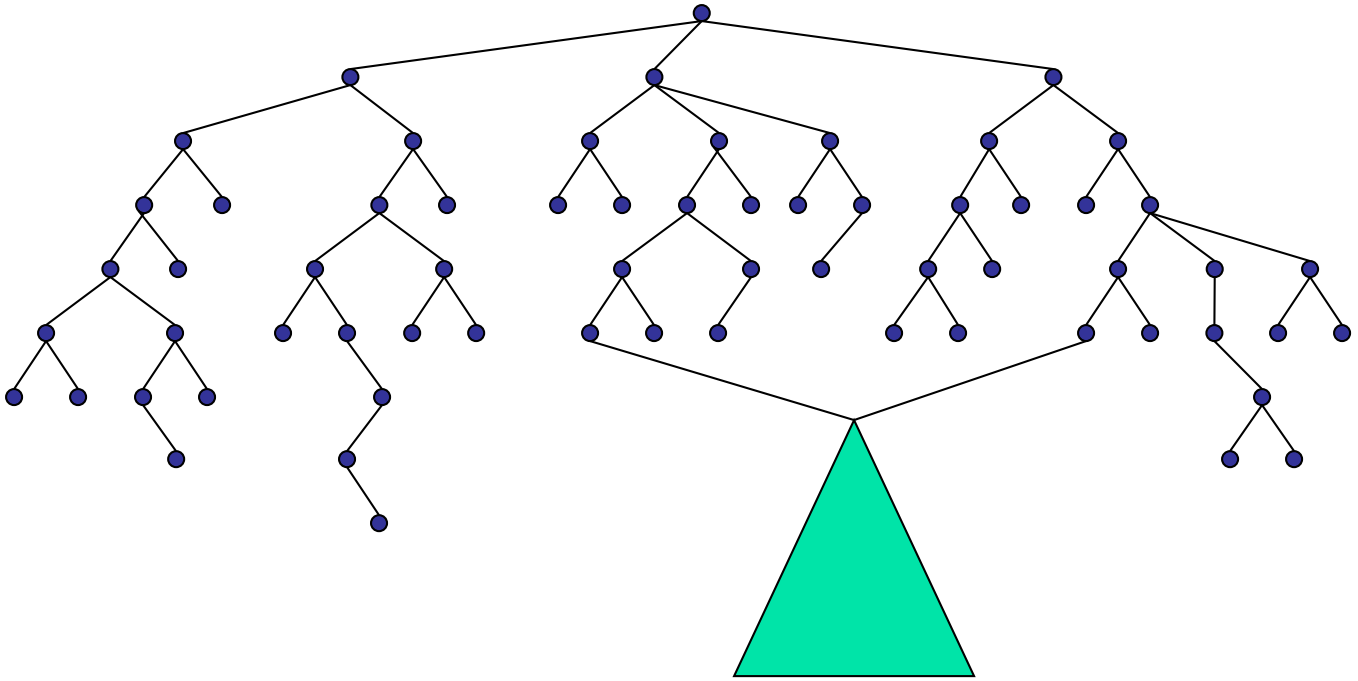
From search trees to search graphs

- Any two nodes that root identical subtrees (subgraphs) can be merged

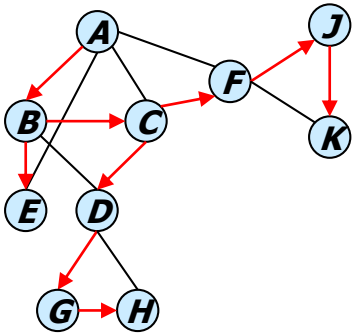


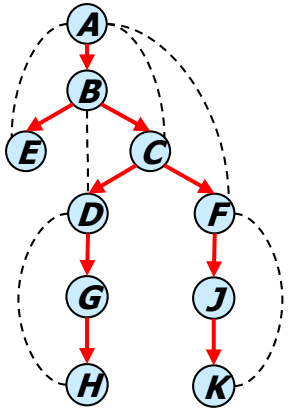
From search trees to search graphs

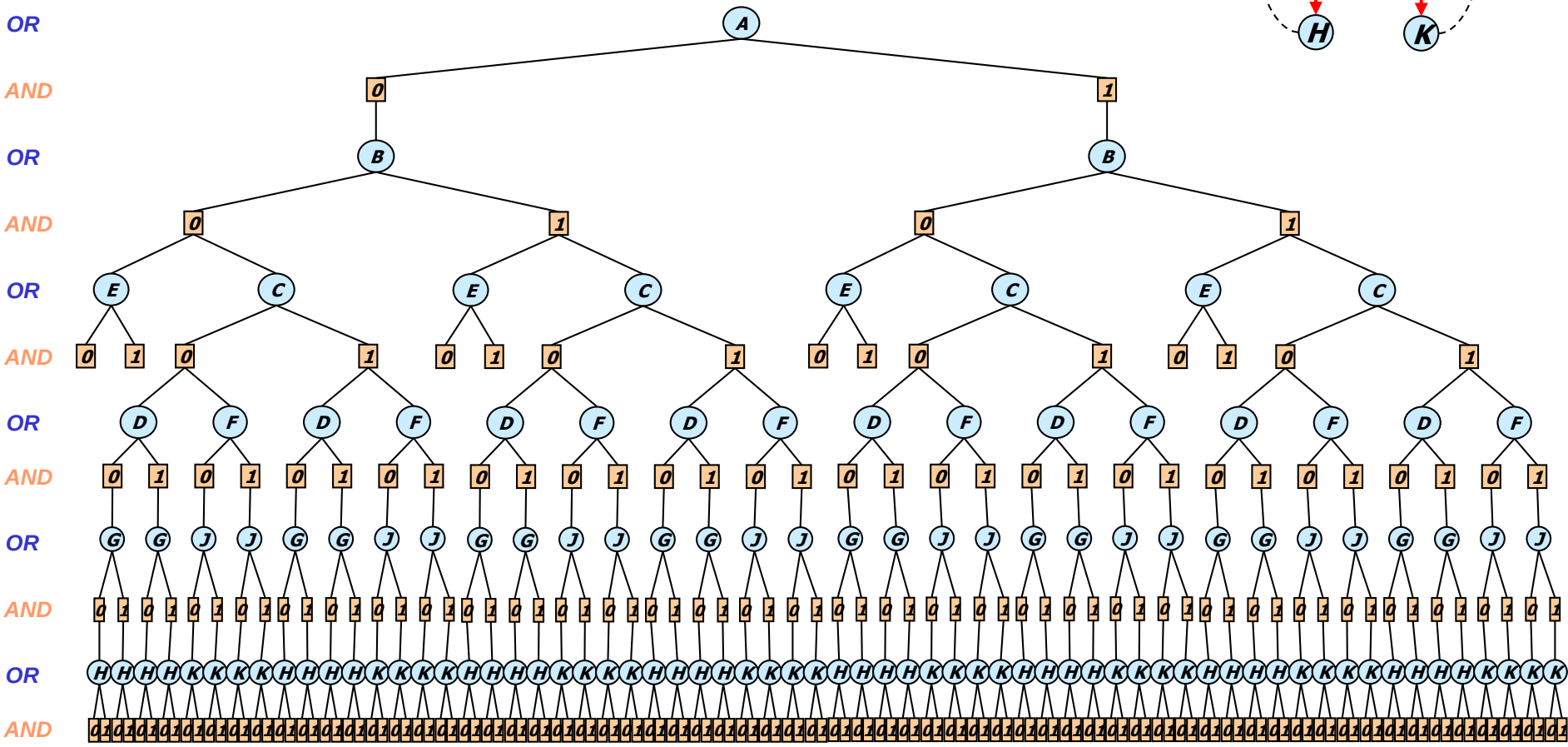
- Any two nodes that root identical subtrees (subgraphs) can be merged



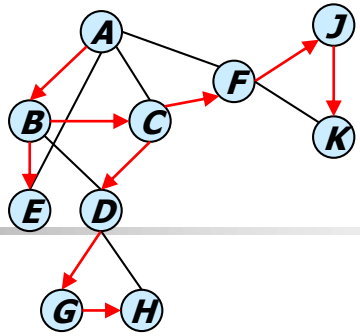
AND/OR Tree

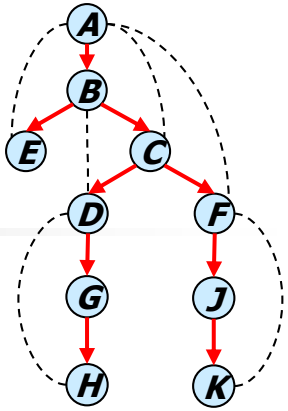

A
B
C
D
E
F
G
H
J
K

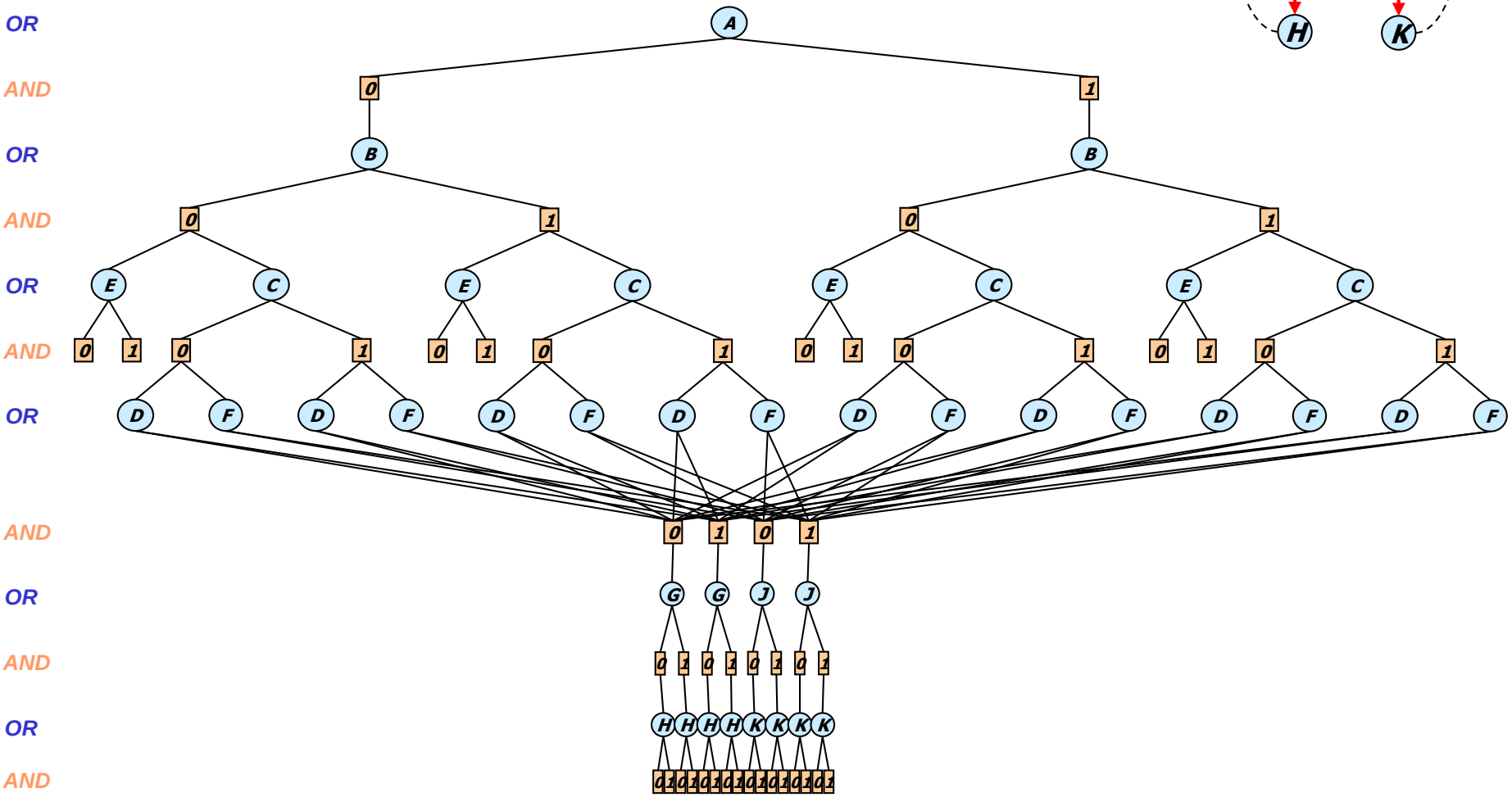

A
B
C
D
E
F
G
H
J
K


OR
AND
OR
AND
OR
AND
OR
AND
OR
AND
OR
AND

An AND/OR Graph: Caching Goods

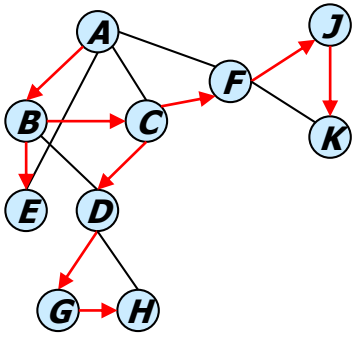


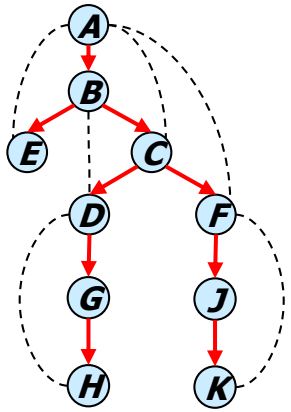




Context-based Caching

- Caching is possible when context is the same
- context = current variable + parents connected to subtree below



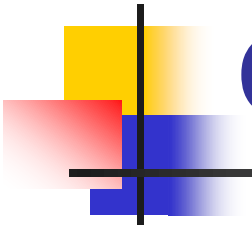


context(B) = {A, B}

context(c) = {A,B,C}

context(D) = {D}

context(F) = {F}

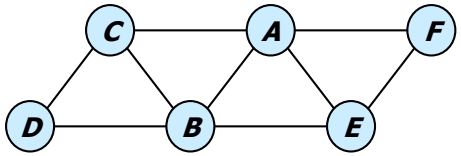


Complexity of AND/OR Graph

- **Theorem:** Traversing the AND/OR search graph is time and space exponential in the induced width/tree-width.

- If applied to the OR graph complexity is time and space exponential in the path-width.

#CSP – AND/OR Tree DFS

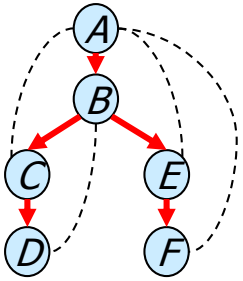


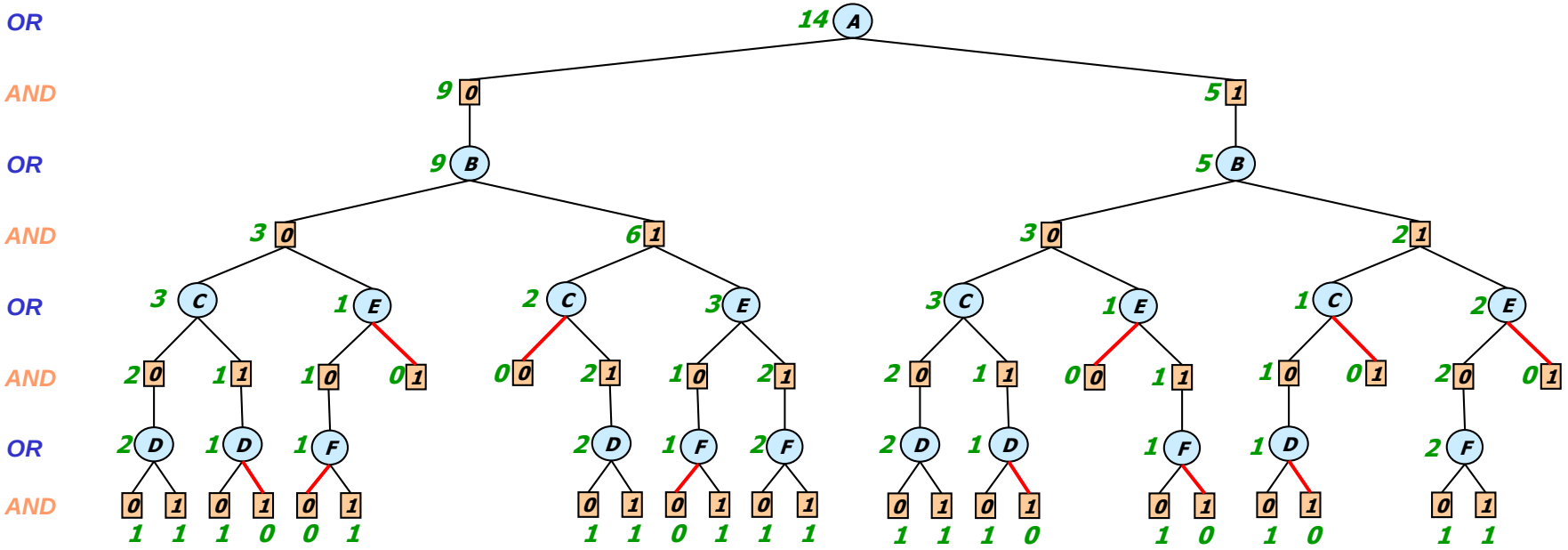
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

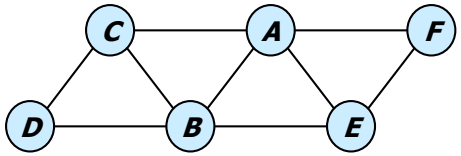
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR Search Graph (Caching Goods)

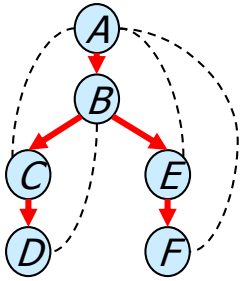


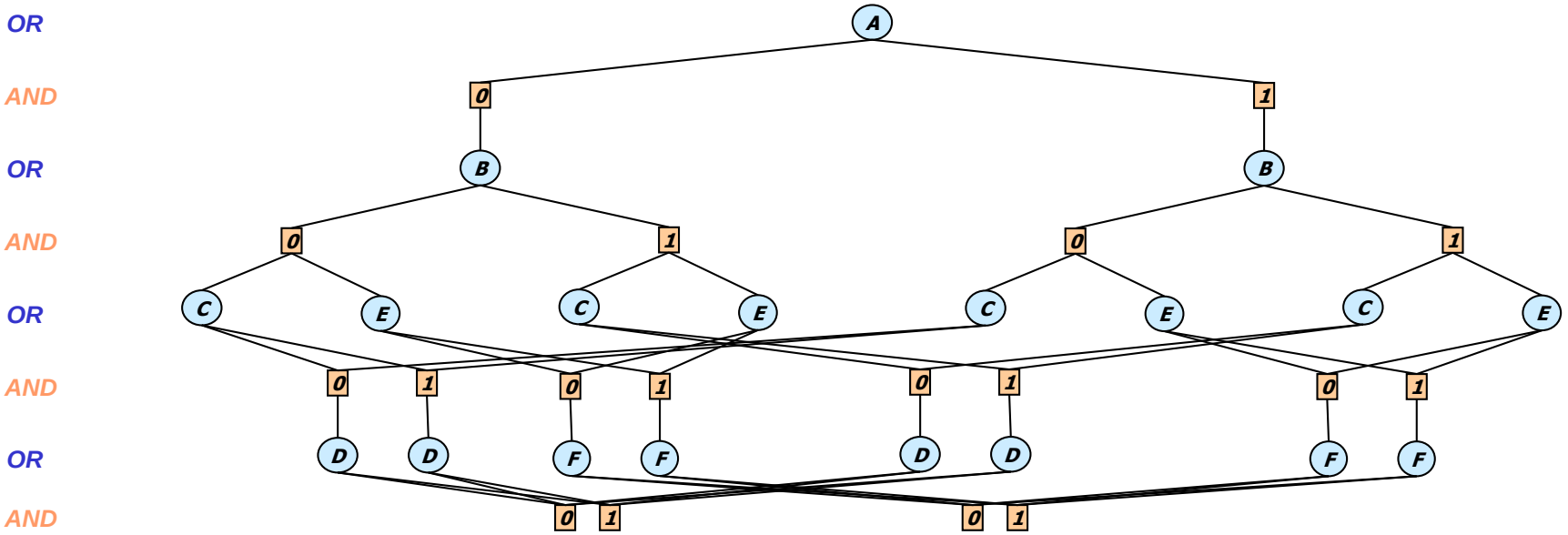
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

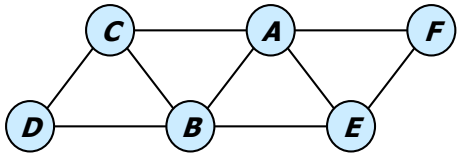
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR Search Graph (Caching Goods)

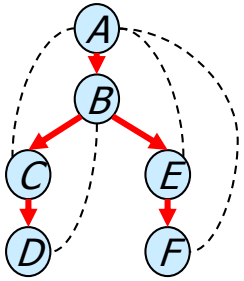


A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

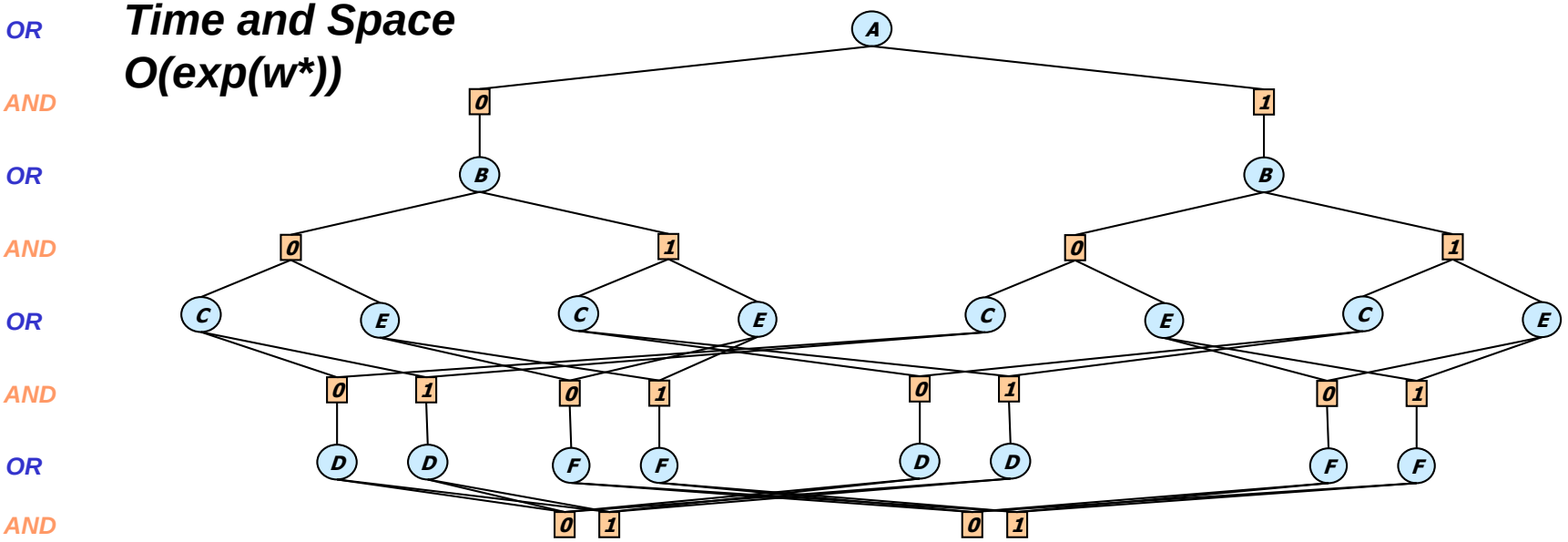
B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

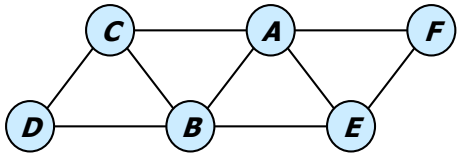
A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Time and Space
O(exp(w*))



#CSP - Searching OR Graph by Good Caching



A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A ***context(A) = [A]***

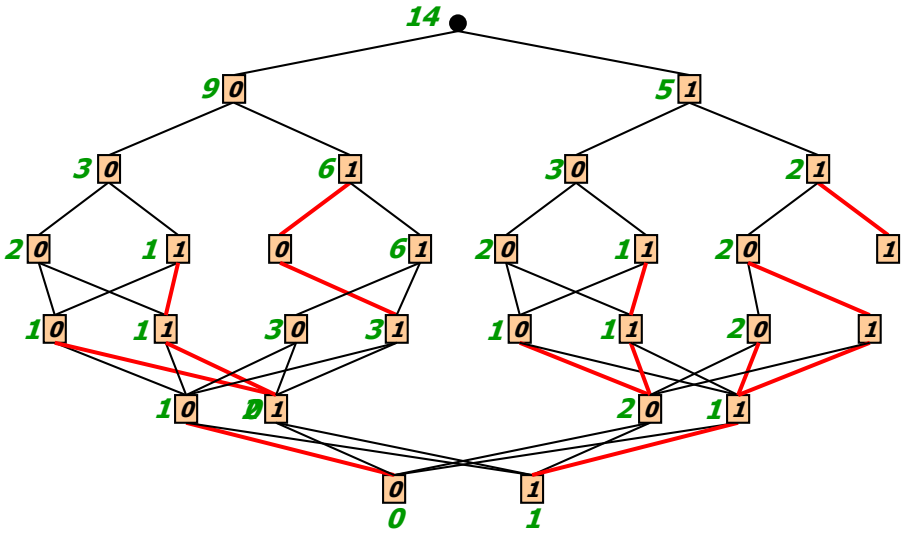
B ***context(B) = [AB]***

C ***context(C) = [ABC]***

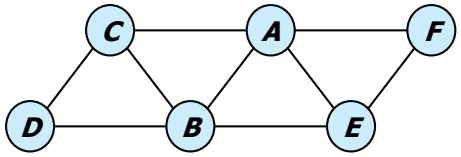
D ***context(D) = [ABD]***

E ***context(E) = [AE]***

F ***context(F) = [F]***



#CSP - Searching OR Graph by Good Caching



A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A ***context(A) = [A]***

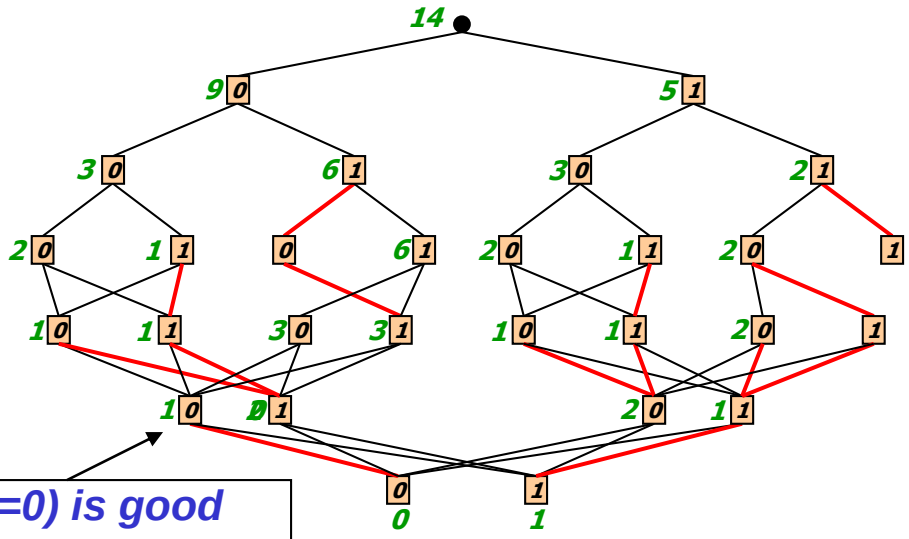
B ***context(B) = [AB]***

C ***context(C) = [ABC]***

D ***context(D) = [ABD]***

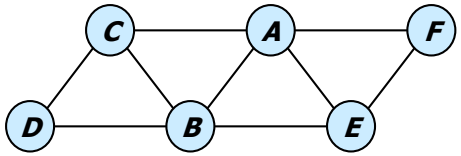
E ***context(E) = [AE]***

F ***context(F) = [F]***



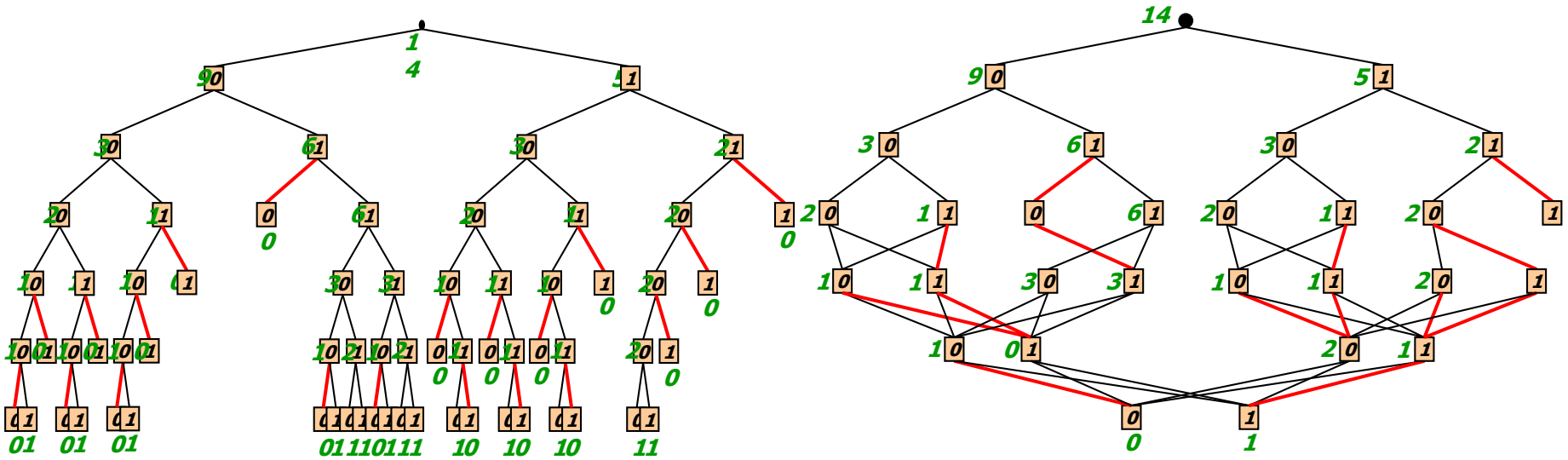
(A=0,E=0) is good
V(A=0,E=0)=1

#CSP - Searching OR Graph by Good Caching

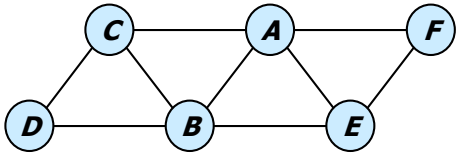


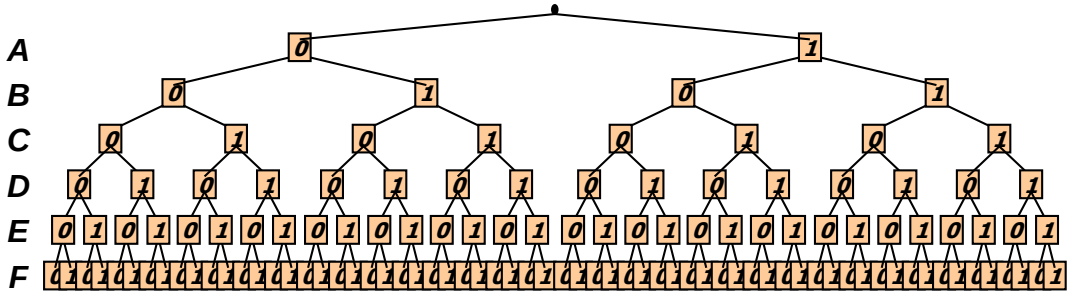
No caching:
O(exp(n))

Good-caching:
O(exp(pw))



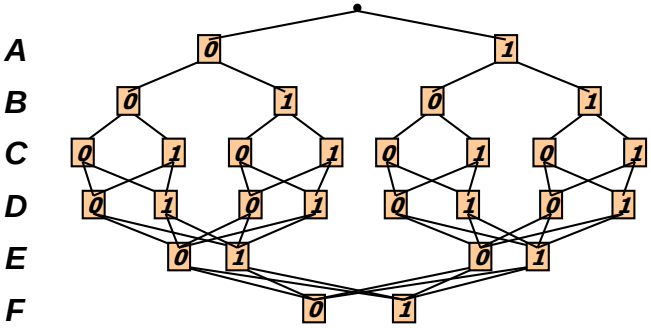
All Four Search Spaces





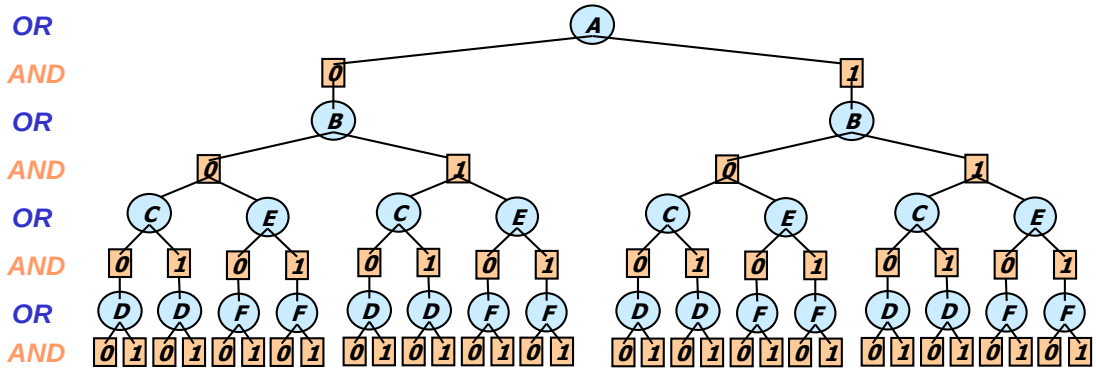
Full OR search tree

126 nodes



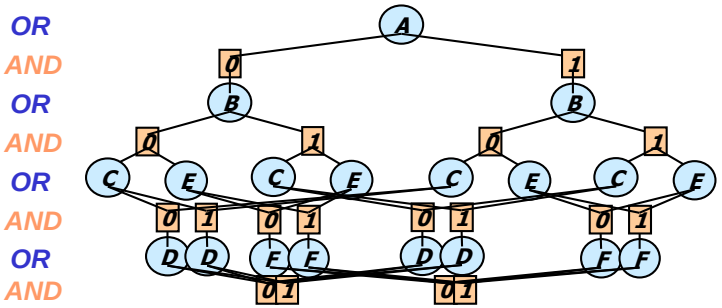
Context minimal OR search graph

28 nodes



Full AND/OR search tree

54 AND nodes



Context minimal AND/OR search graph

18 AND nodes

AND/OR vs. OR DFS Algorithms

k = domain size
m = tree depth
n = # of variables
w = induced width*
pw = path width*

- **AND/OR tree**
  - **Space:** $O(n)$
  - **Time:** $O(n\ k^m)$
    $O(n\ k^{w^* \log n})$

  (Freuder85; Bayardo95; Darwiche01)

- **AND/OR graph**
  - **Space:** $O(n\ k^{w^*})$
  - **Time:** $O(n\ k^{w^*})$

- *OR tree*
  - *Space:* $O(n)$
  - *Time:* $O(k^n)$

- *OR graph*
  - *Space:* $O(n\ k^{pw^*})$
  - *Time:* $O(n\ k^{pw^*})$

Searching AND/OR Graphs

- AO(i): searches depth-first, cache i-context
  - i = the max size of a cache table (i.e. number of variables in a context)

Space: ***O(n)***

Time: ***O(exp(w* log n))***

Space: ***O(exp w*)***

Time: ***O(exp w*)***

AO(i) time complexity?

Impact of AND/OR for Constraint Processing

- **Minor impact for Constraint-satisfaction**
  - **Search with backjumping or without backjumping**
    - **Space: linear, Time: O(exp(logn w*))**
  - **Search with lno-goods learning**
    - **time and space: O(exp(w*))**
  - **Variable-elimination**
    - **time and space: O(exp(w*))**
- **Counting, enumeration**
  - **Search with backjumping**
    - **Space: linear, Time: O(exp(n))**
    - **Space: linear, Time: O(exp(log n w*))**
  - **Search with no-goods caching only**
    - **space: O(exp(w*)) Time: O(exp(n))**
    - **space: O(exp(w*)) Time: O(exp(log n w*))**
  - **Search with goods and no-goods learning**
    - **Time and space: O(exp(path-width), O(exp(log n w*))**
    - **Time and space: O(exp(tree-width), O(exp(w*))**
  - **Variable-elimination**
    - **Time and space: O(exp(w*))**

Bucket-elimination for counting

Algorithm elim-count
Input: A constraint network $\mathcal{R} = (X, D, C)$, ordering d.
Output: Augmented output buckets including the
intermediate count functions and The number of solutions.
1. **Initialize:** Partition C (0-1 cost functions) into ordered buckets $bucket_1, \ldots, bucket_n$,
We denote a function in a bucket N_i, and its scope S_i.)
2. **Backward:** For $p \leftarrow n$ downto 1, do
   Generate the function N^p: $N^p = \sum_{X_p} \prod_{N_i \in bucket_p} N_i$.
   Add N^p to the bucket of the latest variable in $\bigcup_{i=1}^{p} S_i - \{X_p\}$.
3. **Return** the number of solutions, N^1 and the set of output
   buckets with the original and computed functions.

Figure 13.9: Algorithm *elim-count*