1.1

Multi-valued Decision Diagrams

Jean-Charles Régin

Master Computer Science1st Year

Remerciements

- Guillaume Perez
- Francois Pachet
- Pierre Roy

Plan

- Multi-valued Decision Diagram
- Construction
- Operations (intersection, union, difference, negation)
- Reduction algorithm
- In-place Operations

- Modeling tools

Decision Diagrams

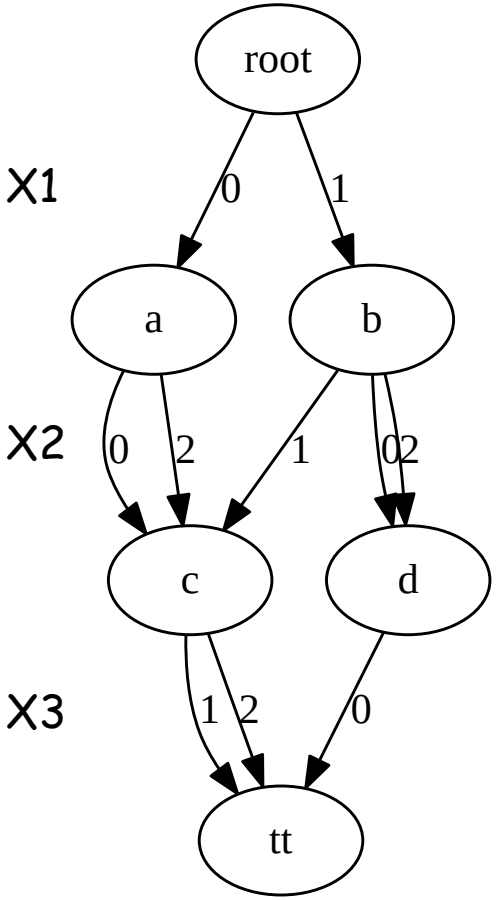
- Old techniques (BDD created in 1959)
- A compressed representation of a set of relations
- We consider here only Multi-values Decision Diagram (MDD)

Multivalued Decision Diagram

- BDD generalization

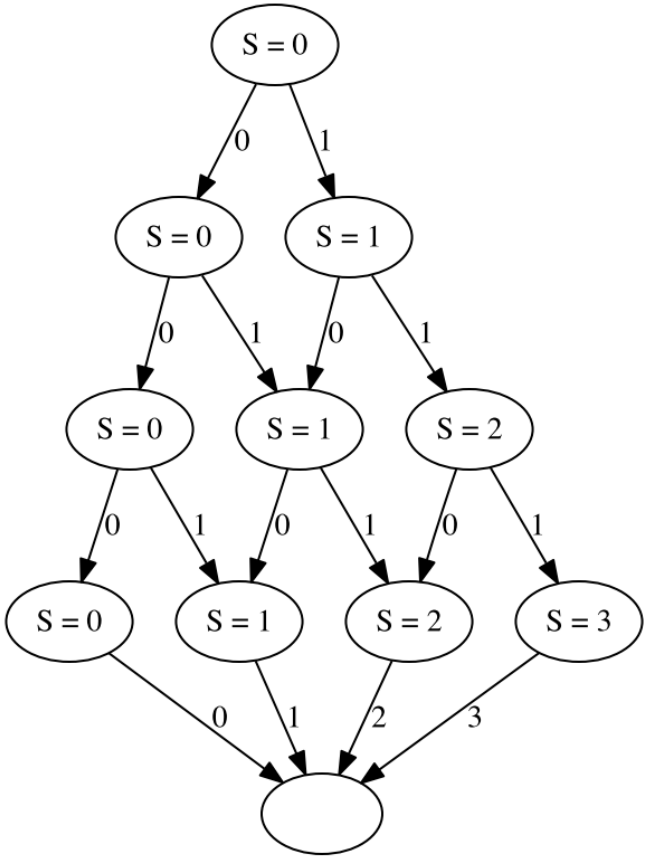
$$f : \{0...d-1\}^r \rightarrow \{true, false\}$$

- Each **level** represents a **variable**
- Each **path** from node ***root*** to node ***tt*** represents a **valid assignment**
- The MDD models **all the tuples** satisfying the constraint
- Compression may gain an **exponential factor**



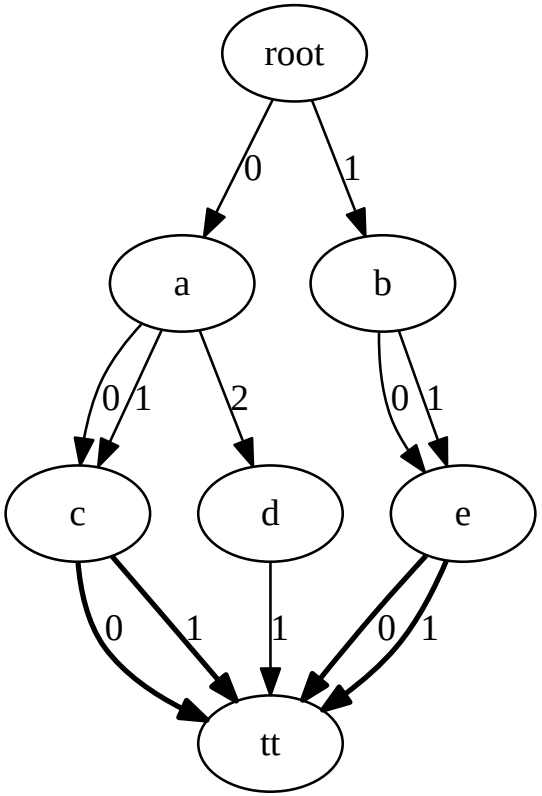
MDD and compression

- Sum of 3 variables
- Corresponds to an automata
- The last layer is the sum value



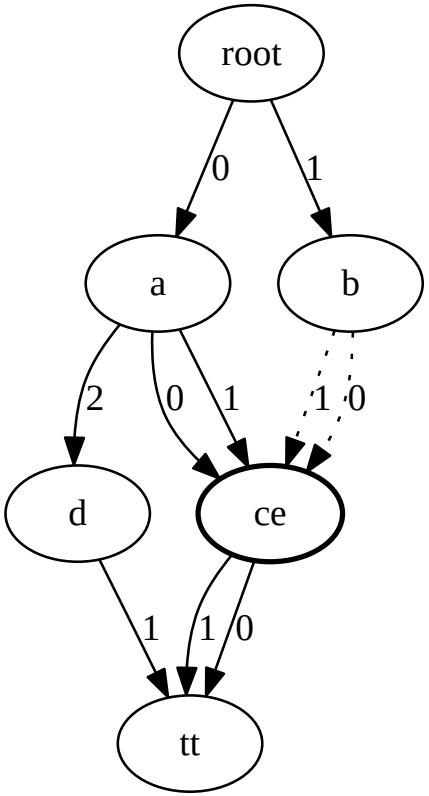
Reduction

- Operation which **merges *equivalent* nodes**
- **Two nodes are equivalent if they have**
  - **the same outgoing edges (same destination + same labels)**
- **Minimization** of finite automata



Reduction

- Operation which **merges *equivalent* nodes**
- **Two nodes are equivalent if they have**
  - **the same outgoing edges (same destination + same labels)**
- **Minimization** of finite automata

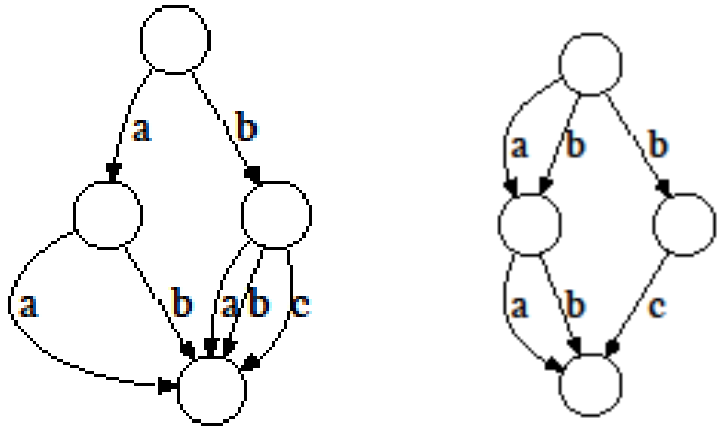


Reduction

- Reduction may gain an exponential factor
- Consequence:
  - MDD can be exponentially smaller than an equivalent automaton

MDDs: determinism?

- Attention, the MDDs we consider are deterministic (only one outgoing arc of a node can a have a given label)



Plan

- Multi-valued Decision Diagram
- **Construction**
- Operations (intersection, union, difference, negation)
- Reduction algorithm
- In-place Operations

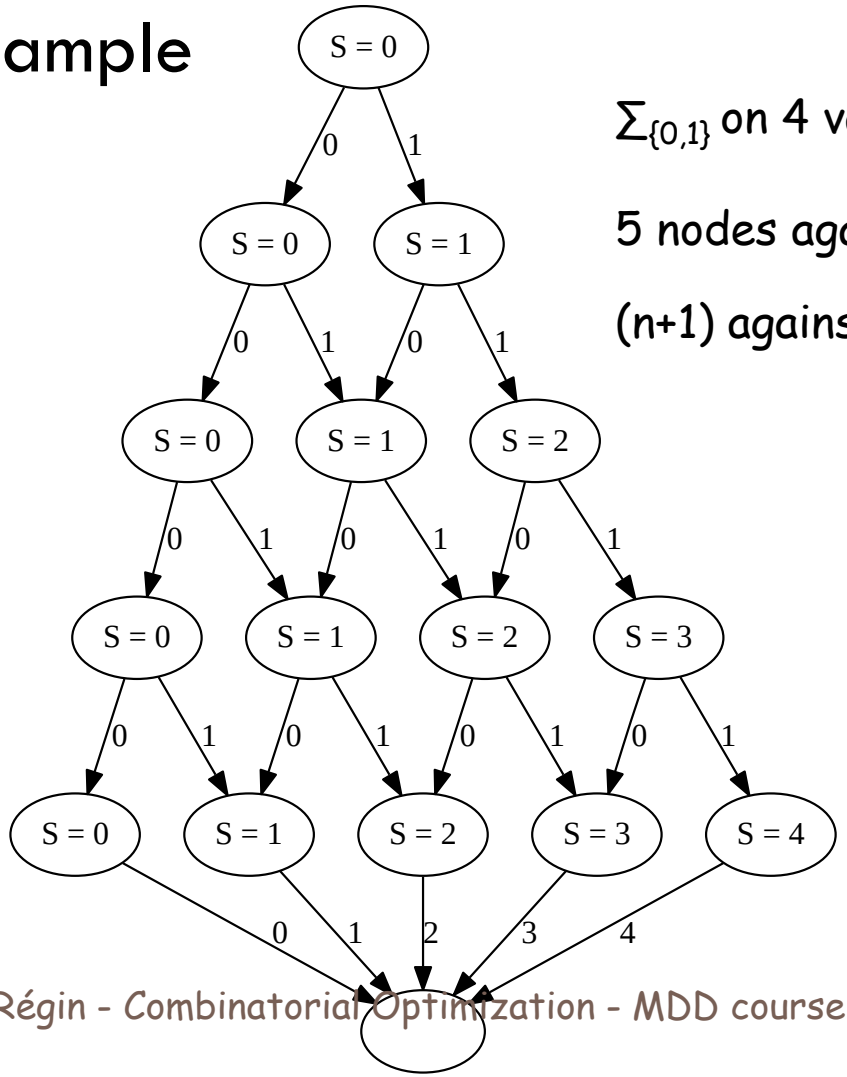
- Modeling tools

MDD: creation

- MDD **can be created without enumerating the solution set**

Ordering effect

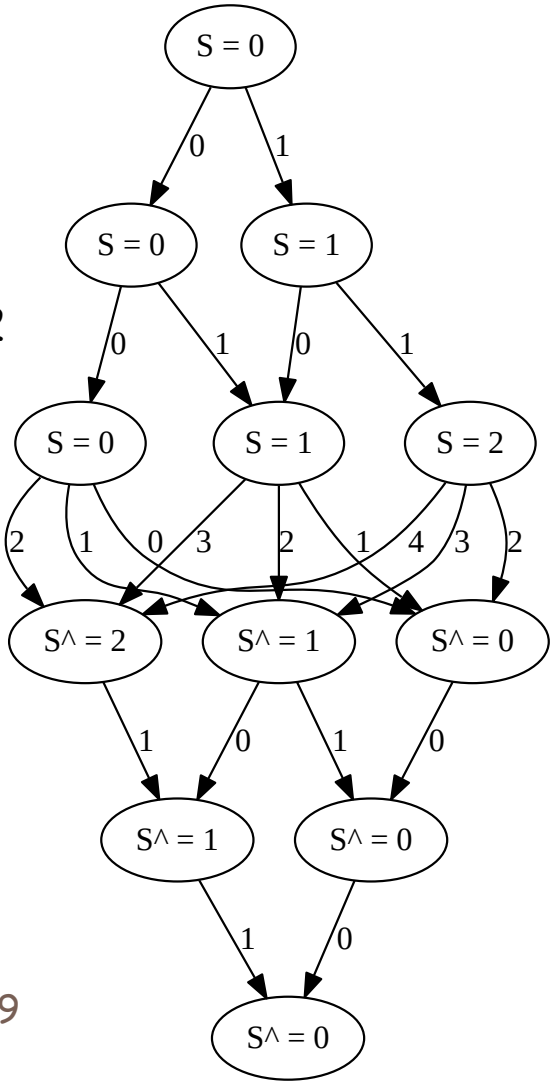
- Example



$\Sigma_{\{0,1\}}$ on 4 variables

5 nodes against 3

(n+1) against (n+1)/2



MDDs construction

- Unrool the automaton until reaching a given depth
- Exemple : a sum constraint
  - Define the state function

- Exercice: draw the MDD representing the solution set of $x1+x2+x3 \in [5,9]$ with the domain of the variables being $D(x1)=\{1,3,7\}$, $D(x2)=\{0,2,4\}$, $D(x3)=\{2,3,4\}$

MDD construction

- Draw the MDD with 3 variables whose domains are {a,b,c} such that
  - There is at most one a
- Draw the MDD with 3 variables whose domains are {a,b,c} such that
  - There is at least one b

- Thanks in term of state
  - Here states correspond to #a and #b

MDD construction

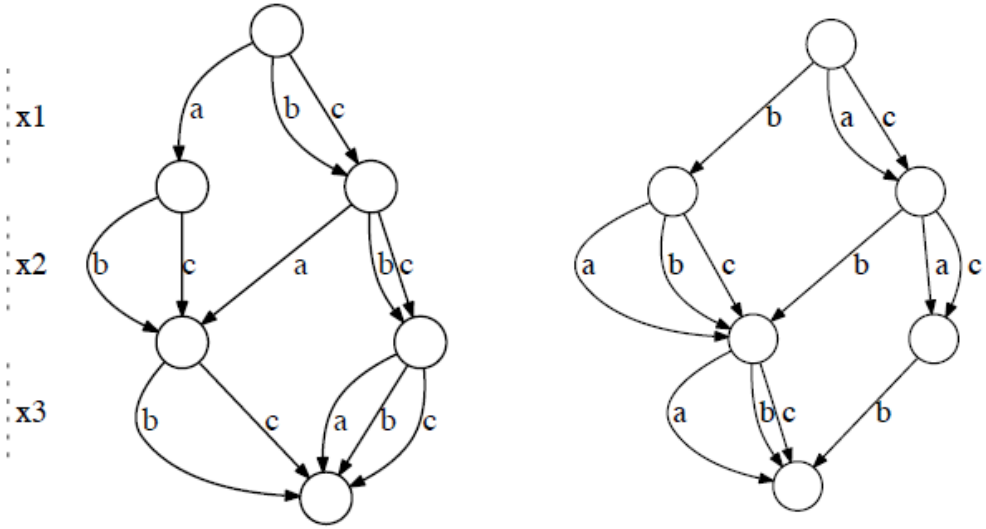


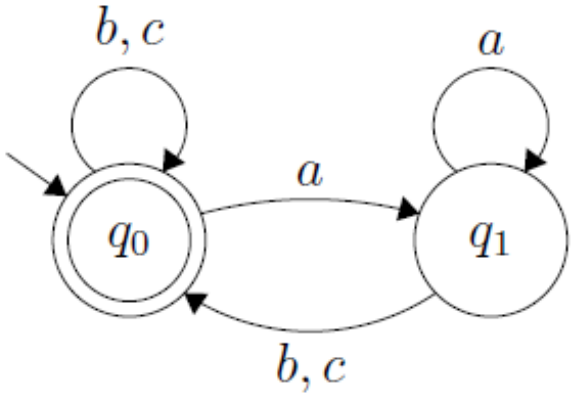
FIGURE 1.1 – On the left, the MDD, defined over three variables (x1,x2,x3) associated with the domain (a, b, c), representing the constraint *at most one* a. On the right the MDD, defined over three variables (x1,x2,x3) associated with the domain (a, b, c), representing the constraint *at least one* b.

Automaton vs MDD

- Let $\Sigma = \{a,b,c\}$. Draw the automaton preventing accepted word to finish by an a

Automaton vs MDD

- Let $\Sigma = \{a,b,c\}$. Draw the automaton preventing accepted word to finish by an a

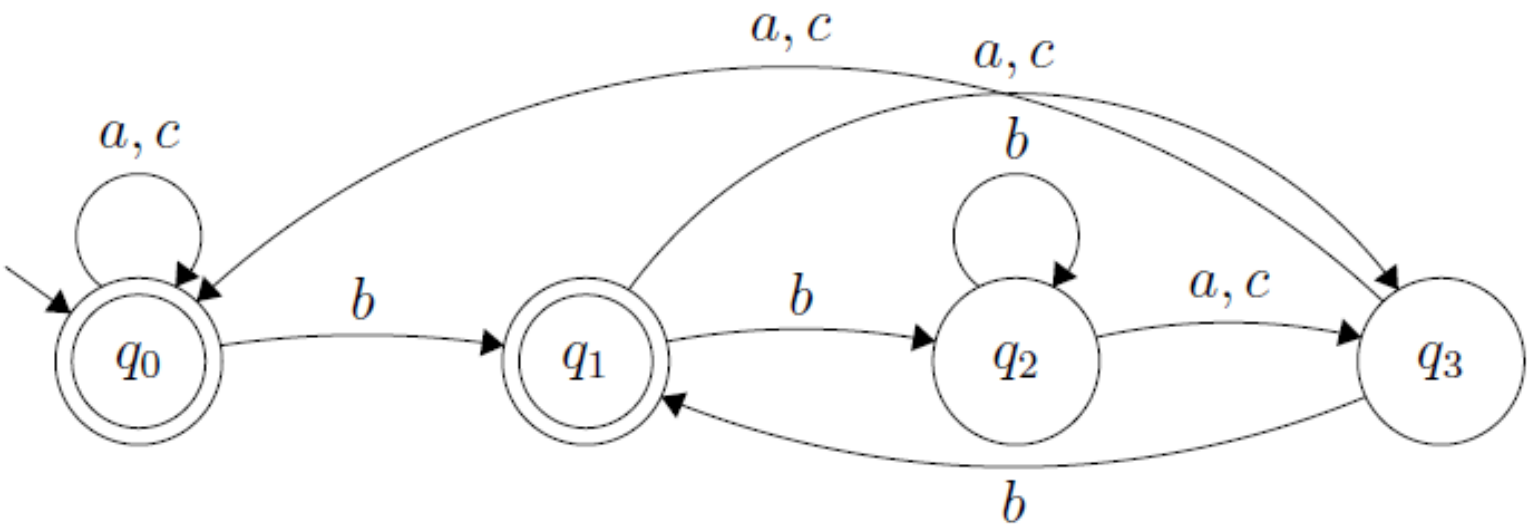


Automaton vs MDD

- Draw the automaton preventing accepted words to have b in the before last position

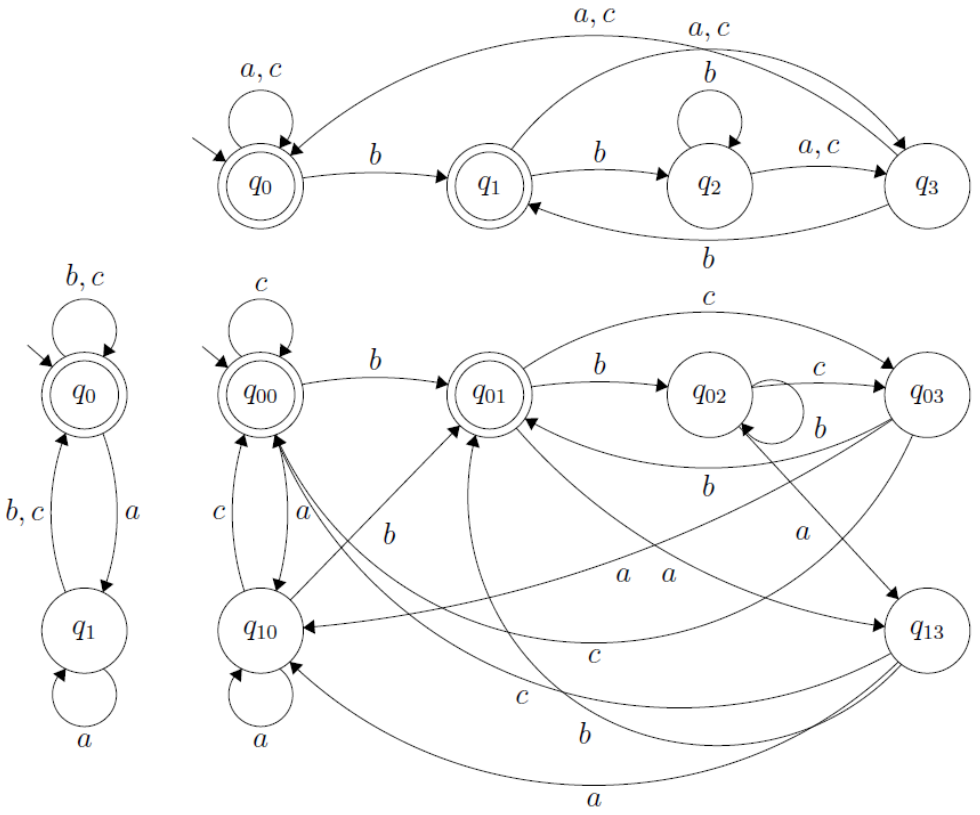
Automaton vs MDD

- Draw the automaton preventing accepted words to have b in the before last position



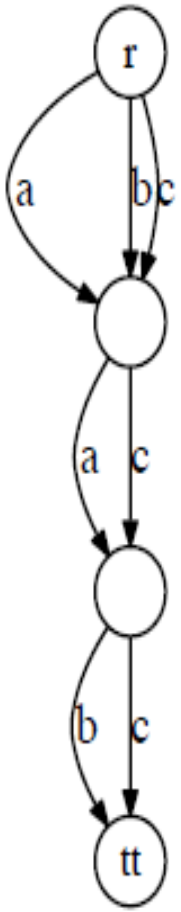
Automaton vs MDD

- The automaton representing the words that cannot finish by an a and not having b in before last position can be obtained by making the product of the two previous automaton



Automaton vs MDD

- The automaton is almost unreadable for humans
- MDD:



Automaton vs MDD

- The minimal DFA that recognizes Ln={w in {0,1}* | the n-th to last symbol in w is 1} has at least 2^n states
- The corresponding MDD has only 2n nodes
  - An exponential factor is gained.
  - This is due to the reduction process

Dynamic programming

- We identify a collections of subproblems, usually encapsulated
- We solve them one by one, smallest first, using the answers to small problems to help figure out larger ones, until the whole problem is solved.
- Each of these subproblems is solved just once, and its solution is stored.
- The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution

- **Principle: saving computation time at the expense of memory consumption.**

Dynamic programming

- Knapsack with repetition
- Pick items (several times) s.t.
  - Sum of weight < W
  - Sum of cost is MAX
- W=10
- Solution ?
  - 1 of item 1 and 2 of item 4
- DP: **K(w)=max{K(w-wi)+vi} for i=1..n, wi ≤ w**
- K(0)=0; K(1)=0; K(2)=K(0)+9=9;
  K(3)=max(K(1)+v4,K(0)+v2)=max(9,14)=14
  K(4)=max(K(1)+14,K(0)+16,K(2)+9)=18;
  K(5)=23; K(6)=30; K(7)=32; K(8)=39; K(9)=44; K(10)=48

Item	Weight	Value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

Dynamic programming

- Knapsack **without** repetition
- Pick items (no repetition) s.t.
  - Sum of weight < W
  - Sum of cost is MAX
- W=10
- Solution ?
  - item 1 and item 3
- DP: **K(w,j)=max value for capacity w with items 1..j**
  K(w,j)= max{K(w-wj,j-1) + vj, K(w,j-1)}
  we add the item or not

Item	Weight	Value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

Dynamic programming vs MDD

- Avec les MDD on engendre les étapes facilement une après l'autre
- On n'a pas besoin de définir formellement ce qui se passe entre deux étapes. On a juste à essayer les valuers possibles pour la prochaine variable.
- On peut construire des chemins partiels qui ne conduisent à rien. Ils seront ensuite éliminés par la réduction
- C'est donc plus simple et plus facile à faire

MDD

- We can solve these two variants of the same problem with MDDs
  - Knapsack with repetition
  - Knapsack without repetition

Table constraints

- Constraint defined by the **explicit** list of combinations **satisfying** the constraint
  - x1=**a**, x2=**a**, x3=**b**, x4=**a**, x5=**b**
    valid assignment for this constraint

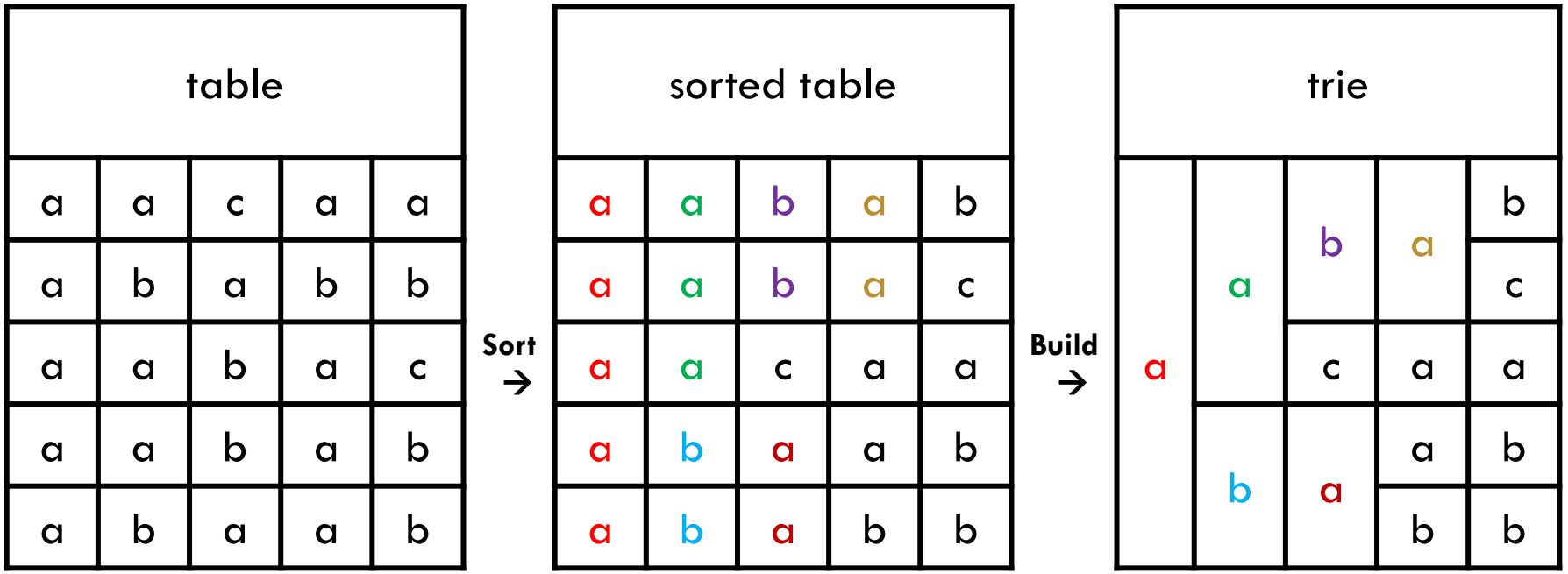
x1	x2	x3	x4	x5
a	a	c	a	a
a	b	a	b	b
a	a	b	a	c
a	**a**	**b**	**a**	**b**
a	b	a	a	b

Table

- **Complexity:**
  - Structure representing the table
  - Simple table: **|T| * |X|**
  - **Reduce this complexity**

Table

- How sorting can be useful?



- Time complexity **O(n+m+d+|T|*r) Optimal**
- Previous one O(n * d + |T|*r)

MDD

- Definition without enumerating the solutions
- Strong compression of solution set
- So what?
- Operations!

Plan

- Multi-valued Decision Diagram
- Construction
- **Operations (intersection, union, difference, negation)**
- Reduction algorithm
- In-place Operations

- Modeling tools

MDD: operations

- Intersection, union, difference, negation etc…
- Operations are performed **without decompression**
- Intersection of 2 MDDs is equivalent to make the conjunction of the 2 constraints represented by the MDDs
- Relation between MDD operations and constraints combination
  - Intersection : conjunction
  - Union : disjunction
  - Negation : negation

MDD intersection

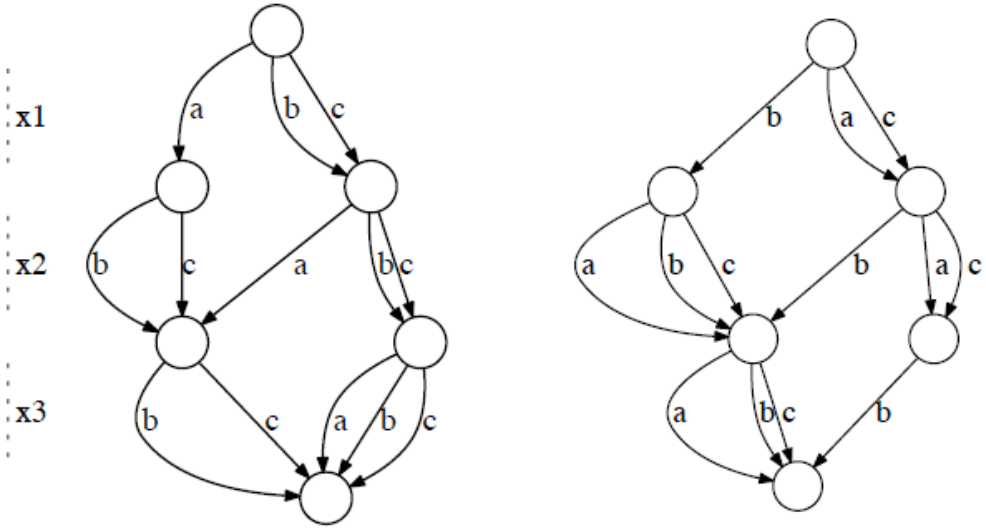


FIGURE 1.1 – On the left, the MDD, defined over three variables (x1,x2,x3) associated with the domain (a, b, c), representing the constraint *at most one* a. On the right the MDD, defined over three variables (x1,x2,x3) associated with the domain (a, b, c), representing the constraint *at least one* b.

MDD Intersection

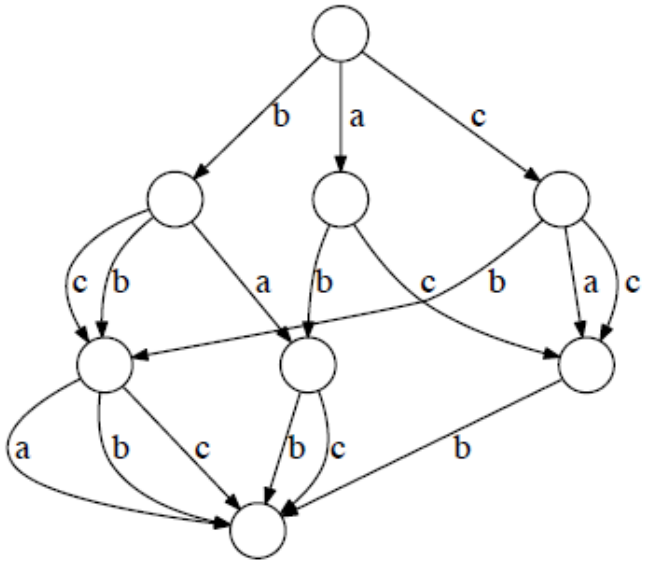
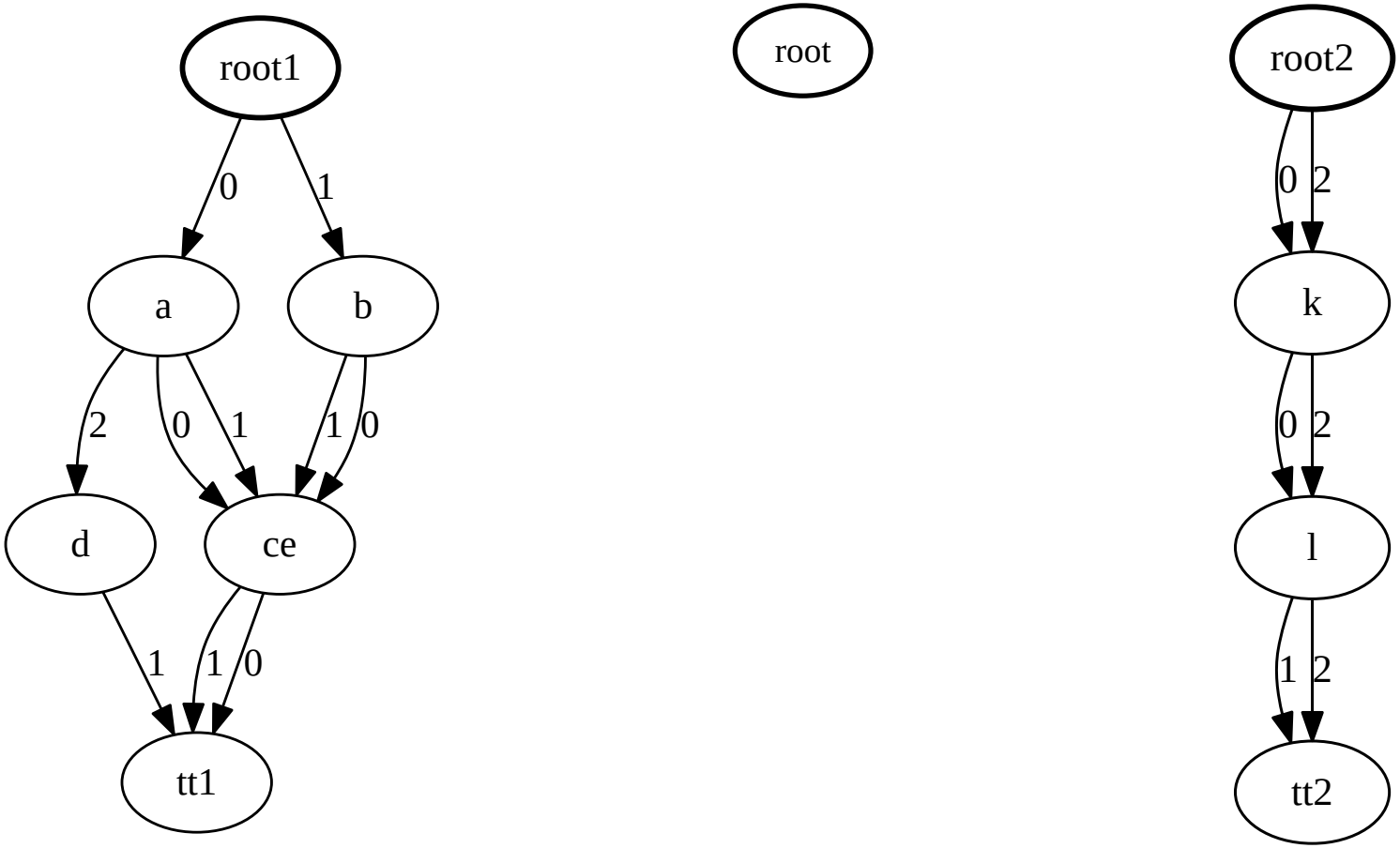


FIGURE 1.2 – The intersection of the two MDDs from Figure 1.1. Thus all the solutions do not contain more than one a but do contain at least one b.

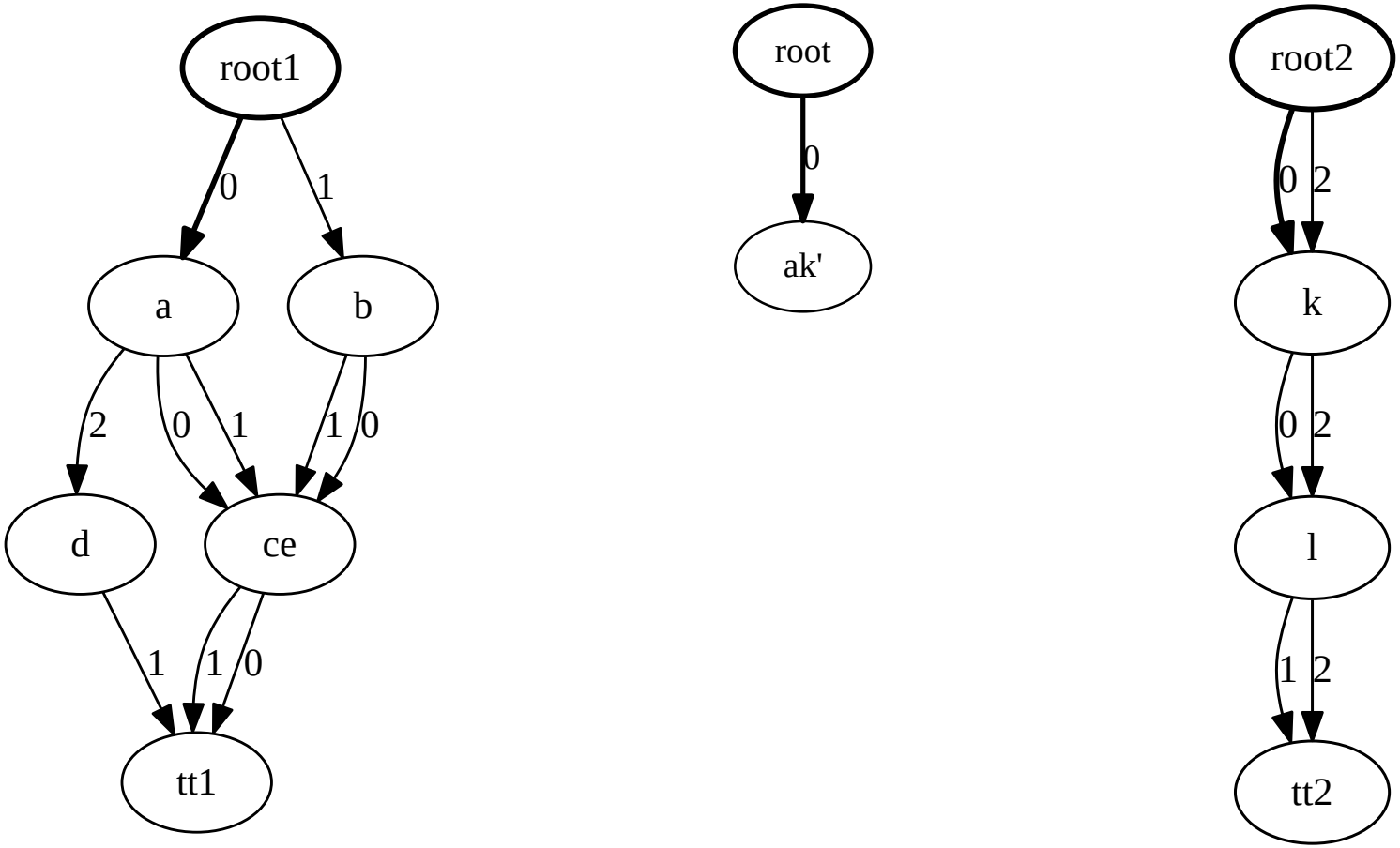
MDD Intersection

- Draw (from scratch) the MDD equals to the intersection of the 2 previous MDD
- In this MDD
  - There is at most one a
  - There is at least one b

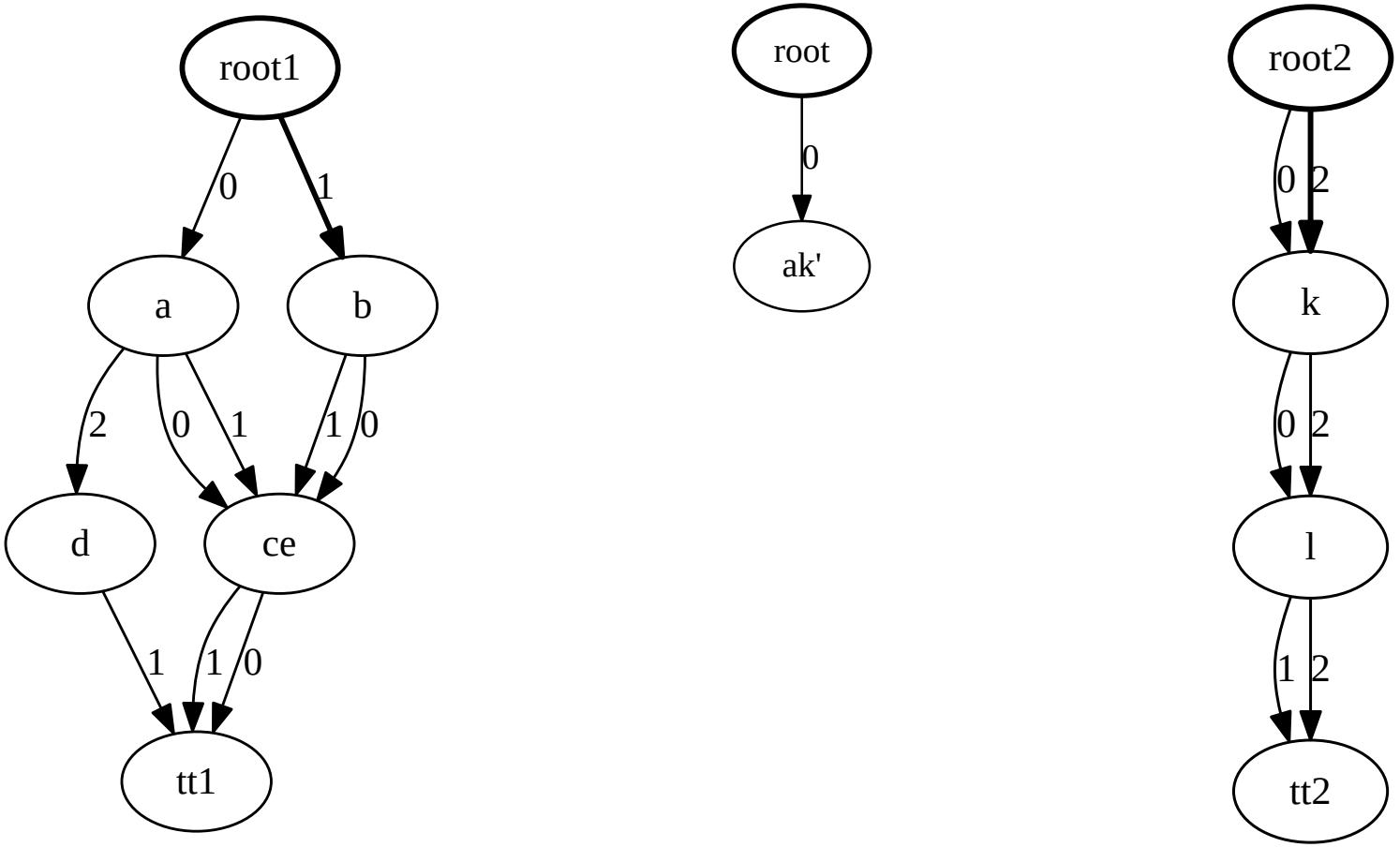
Intersection



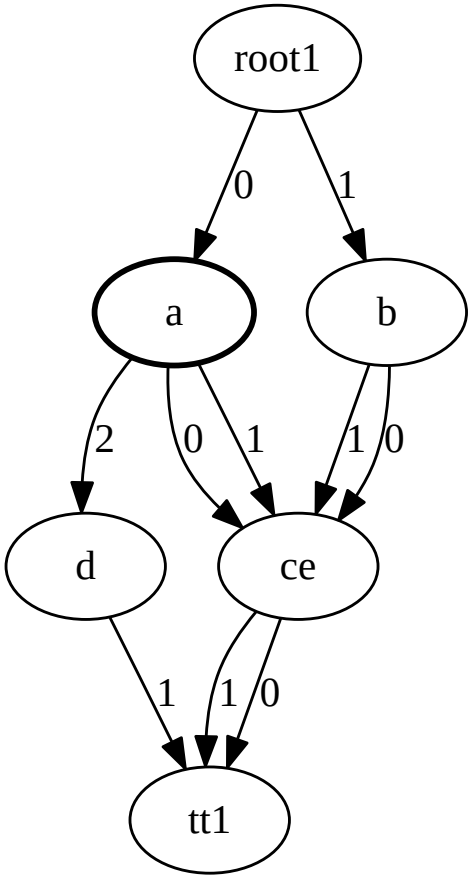
Intersection

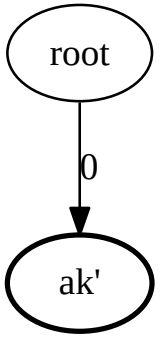


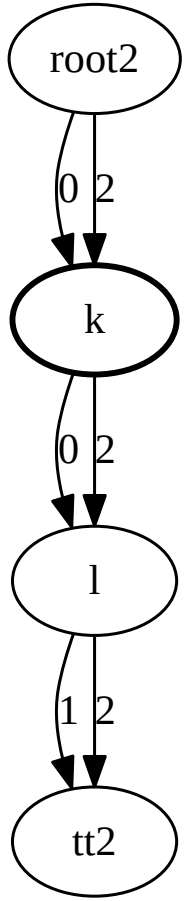
Intersection



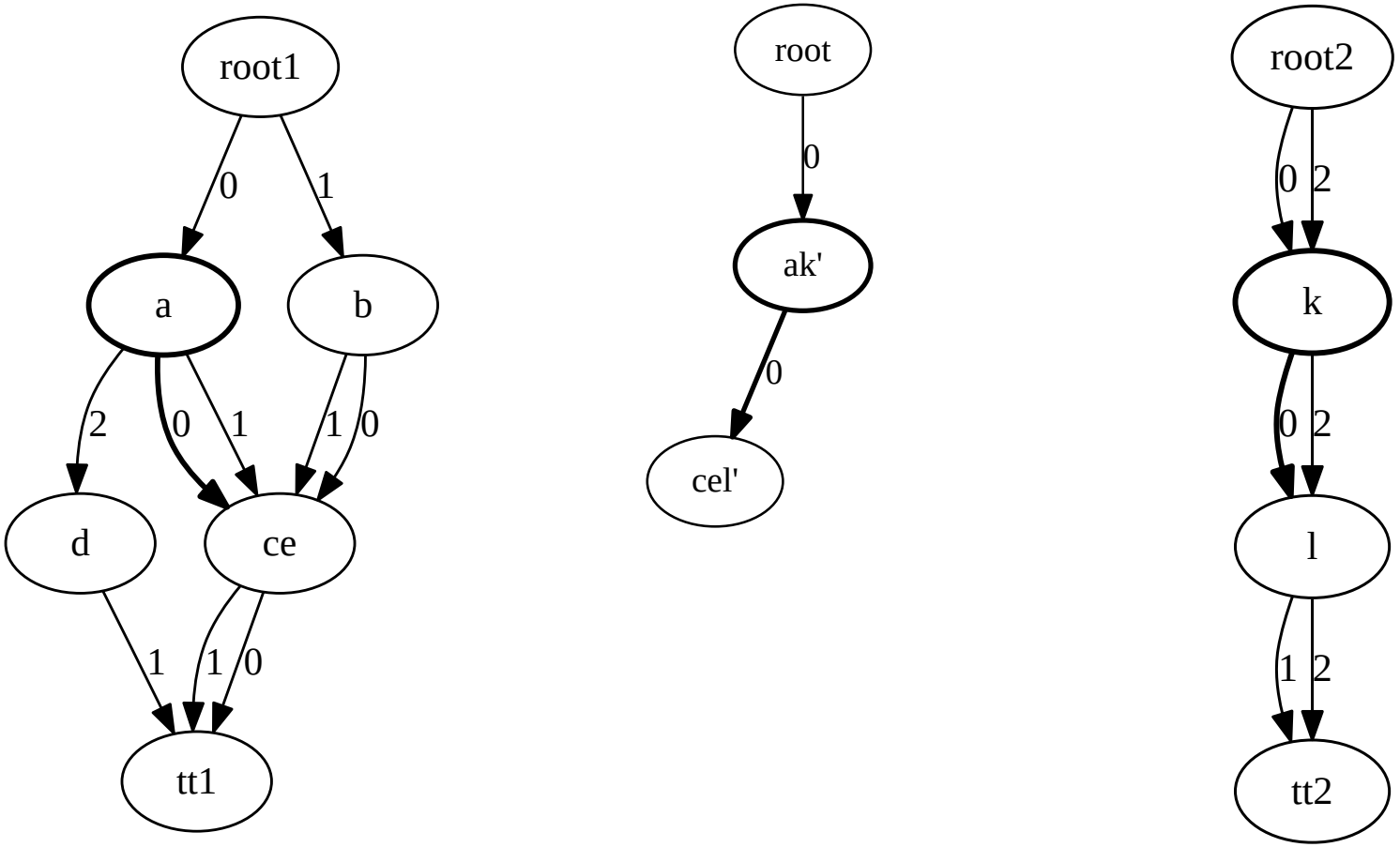
Intersection



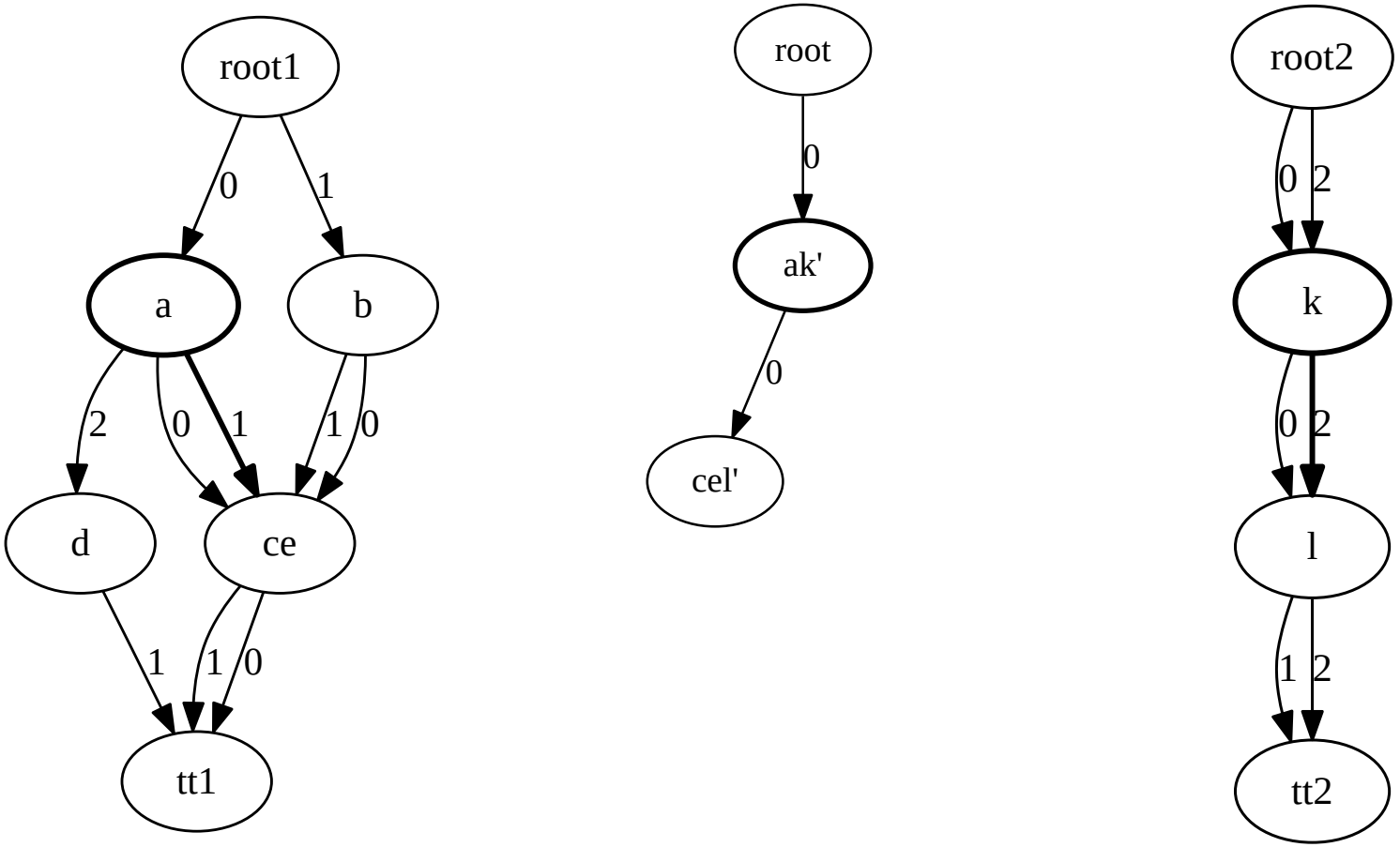




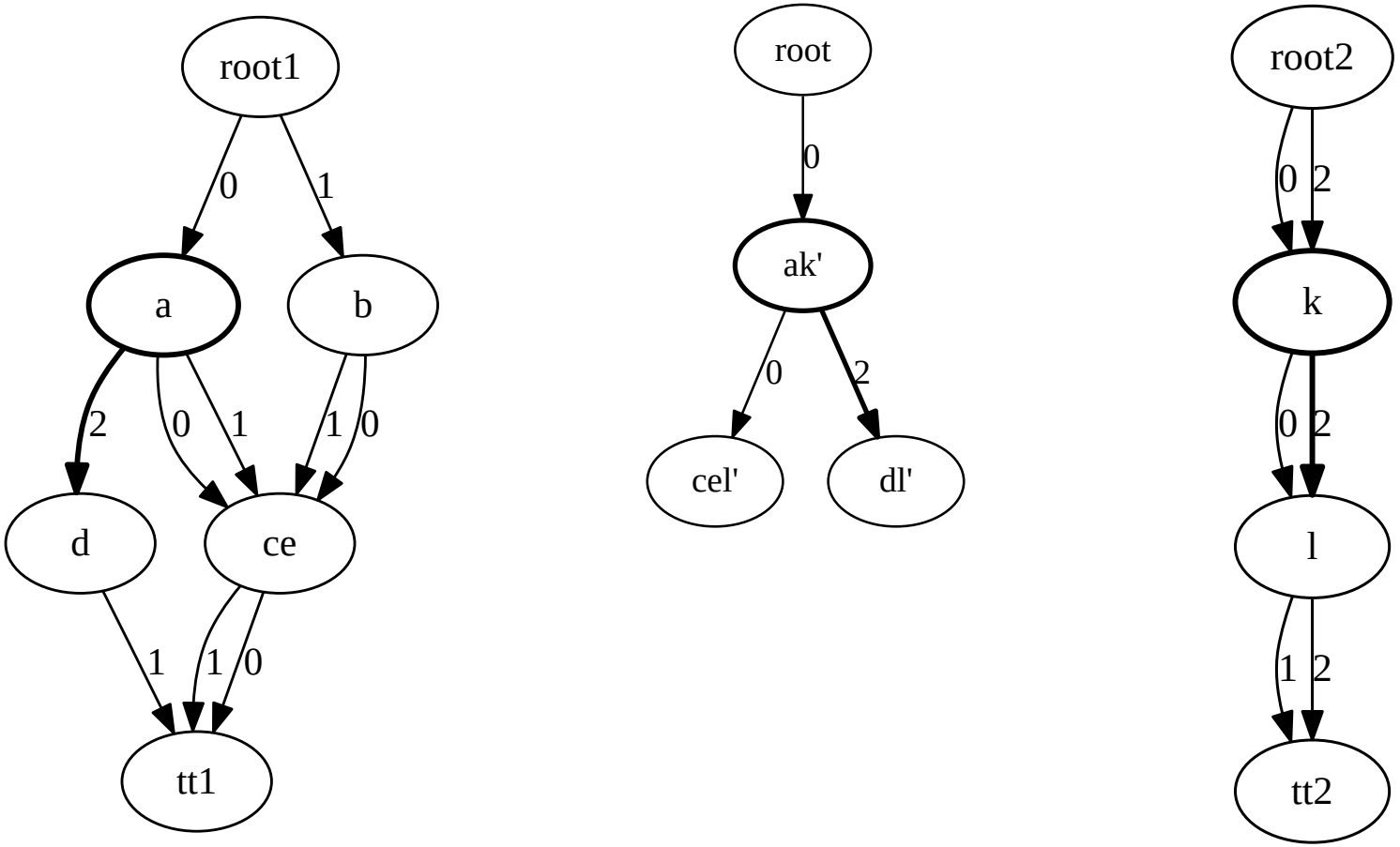
Intersection



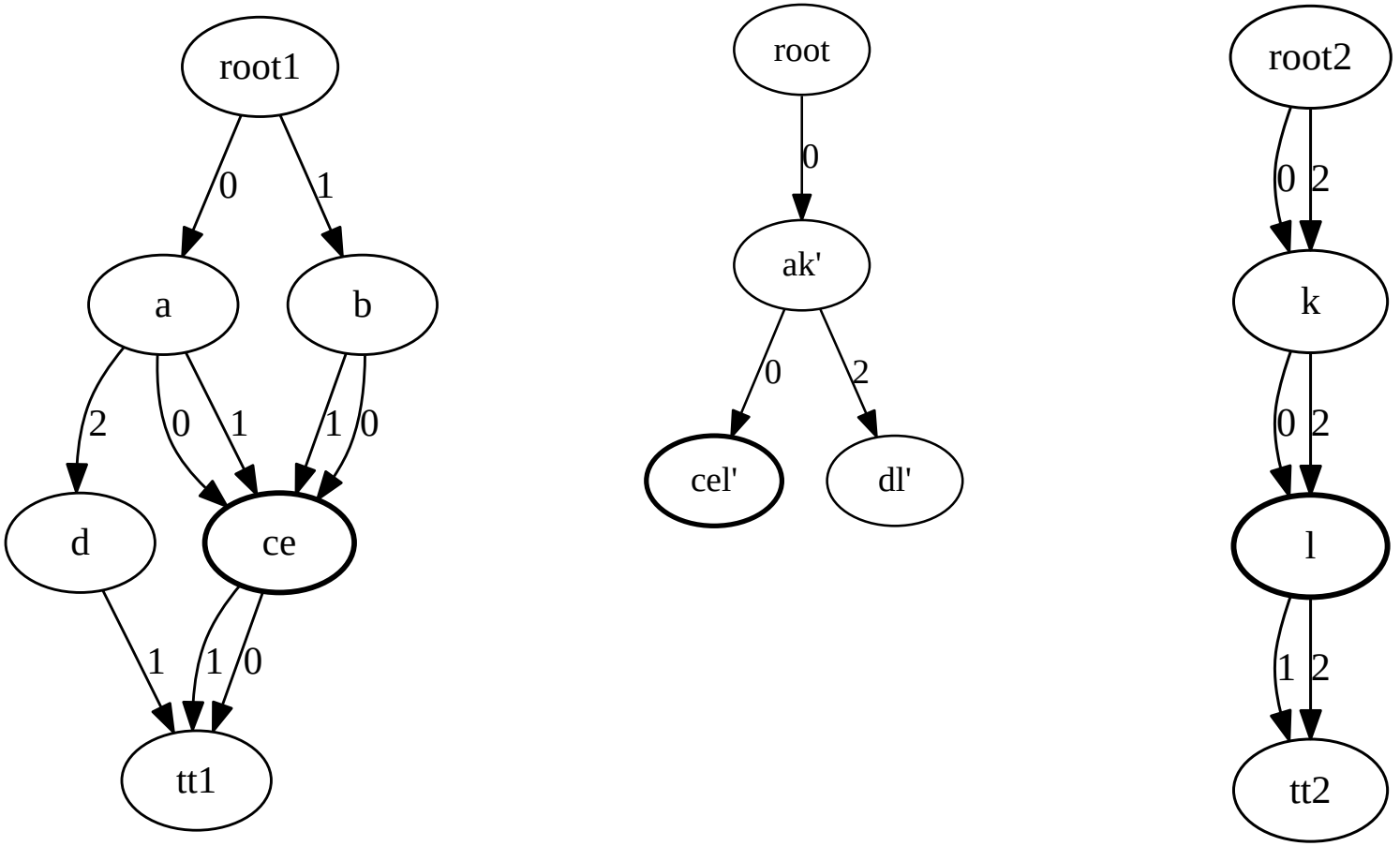
Intersection



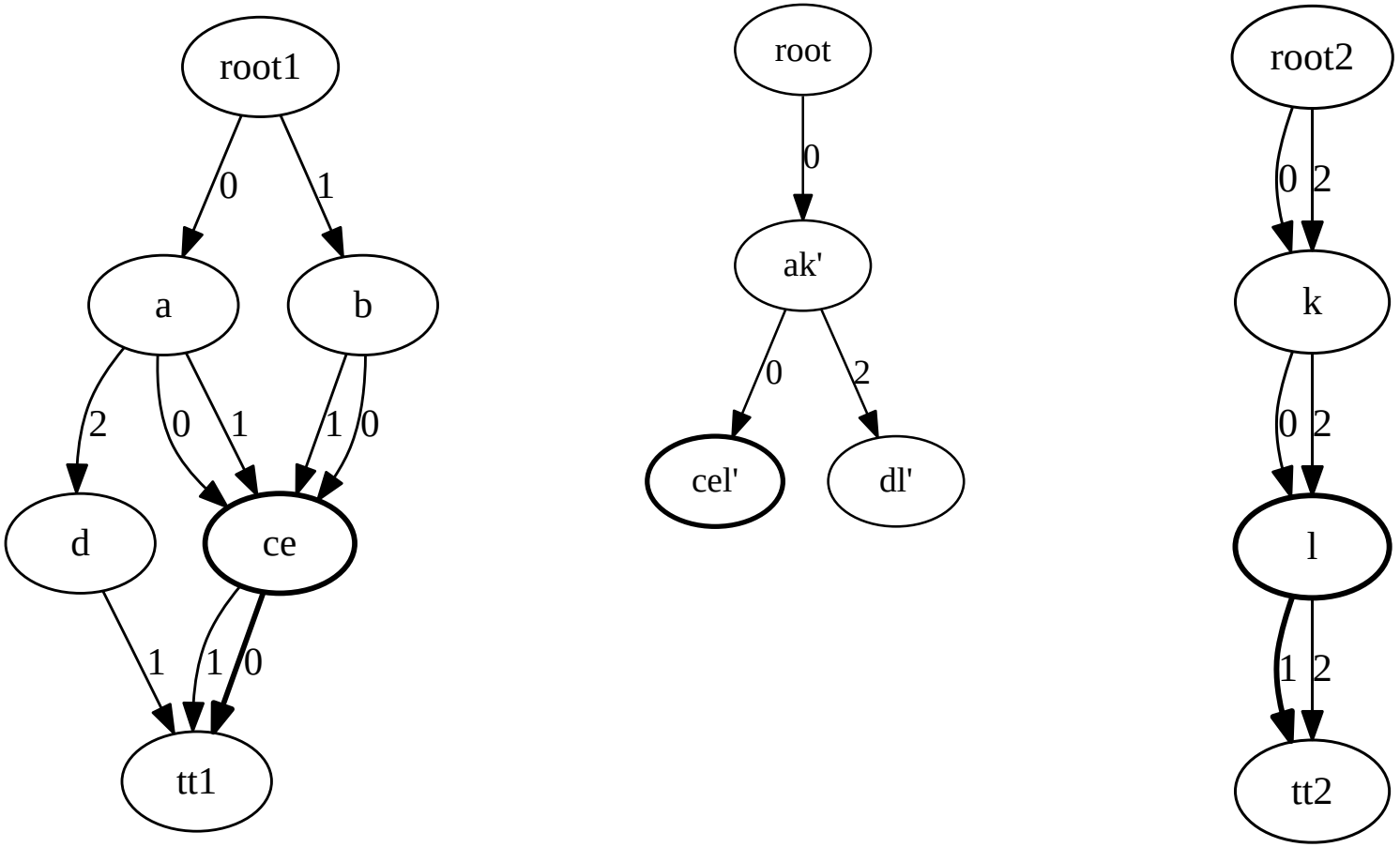
Intersection



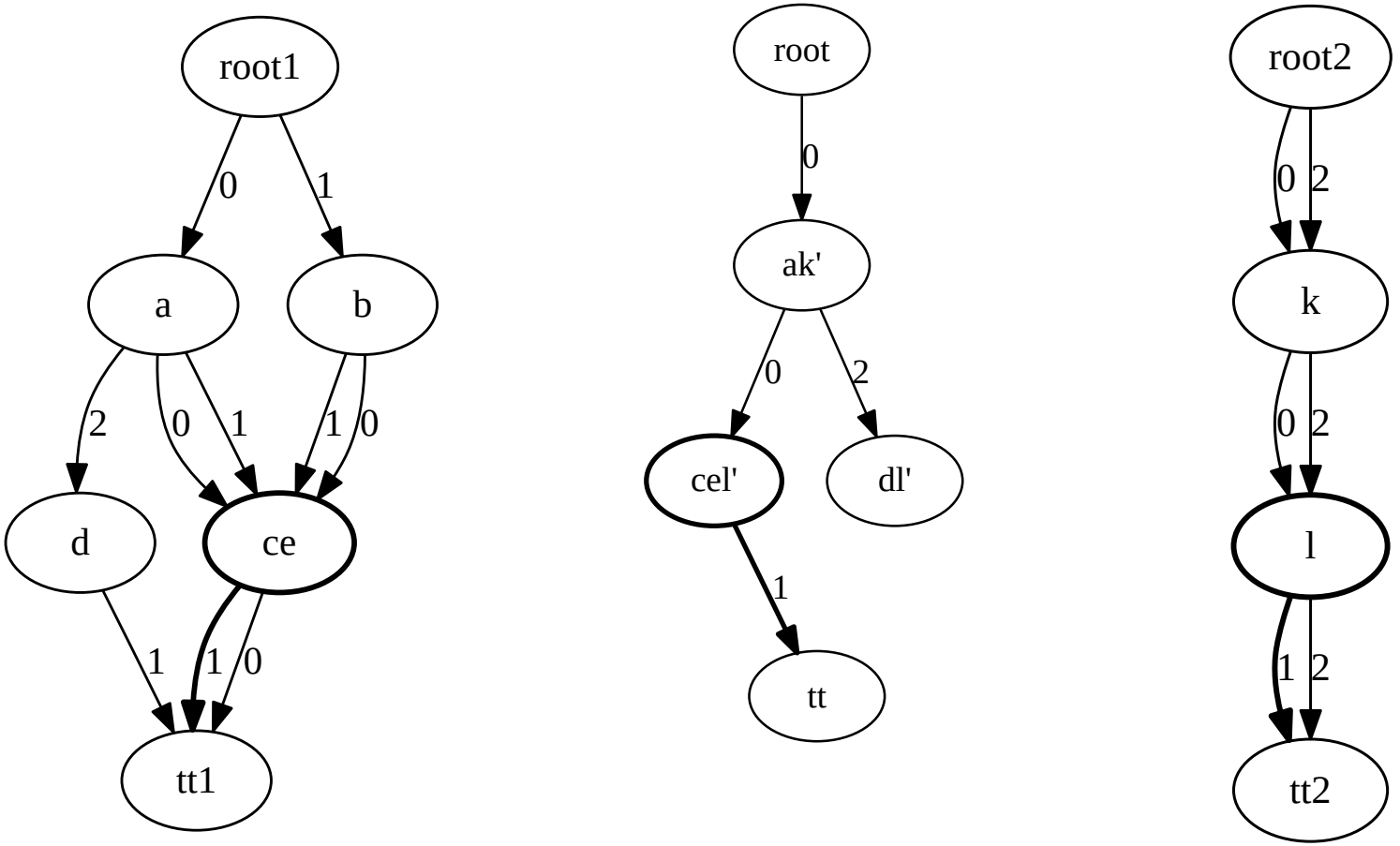
Intersection



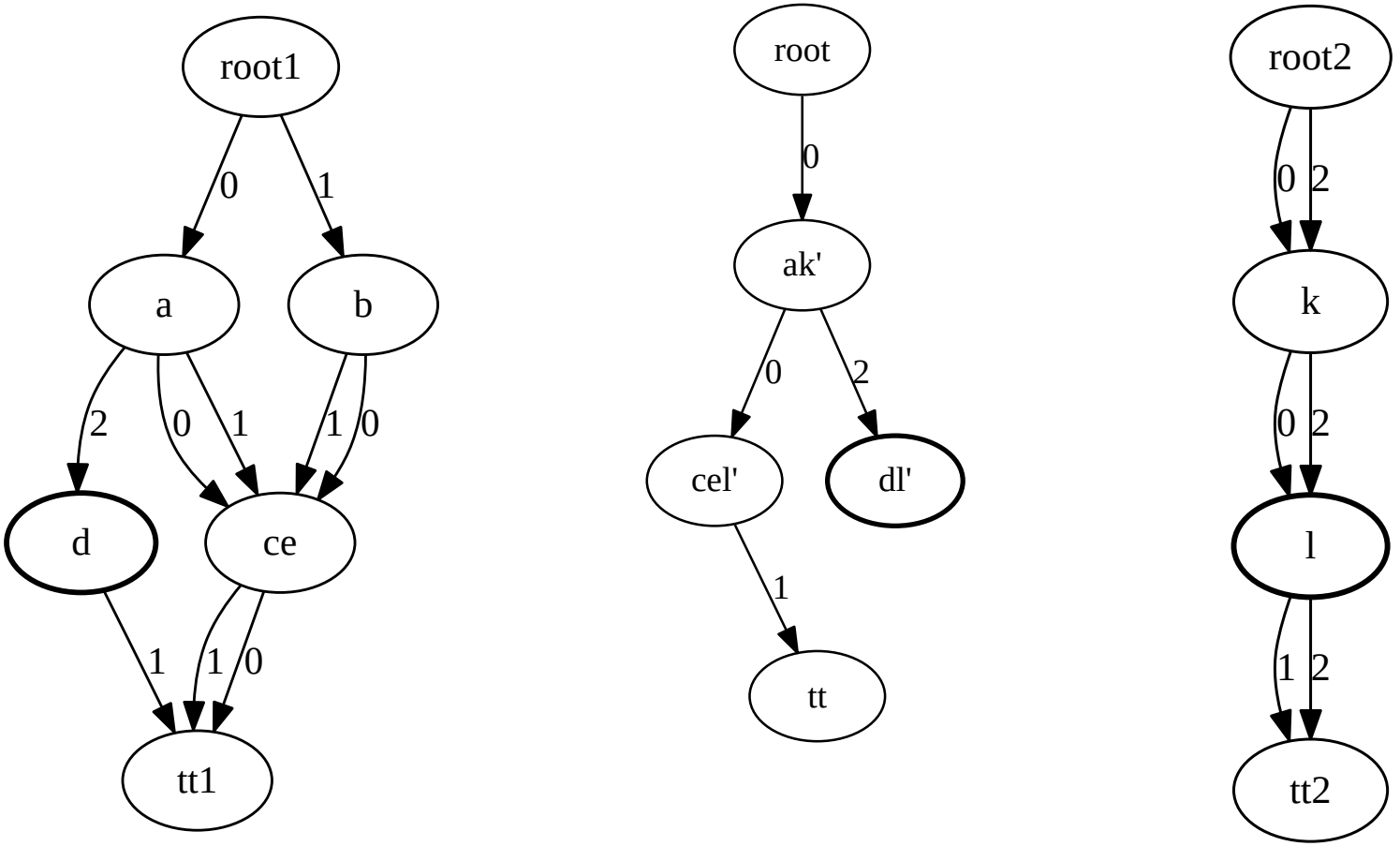
Intersection



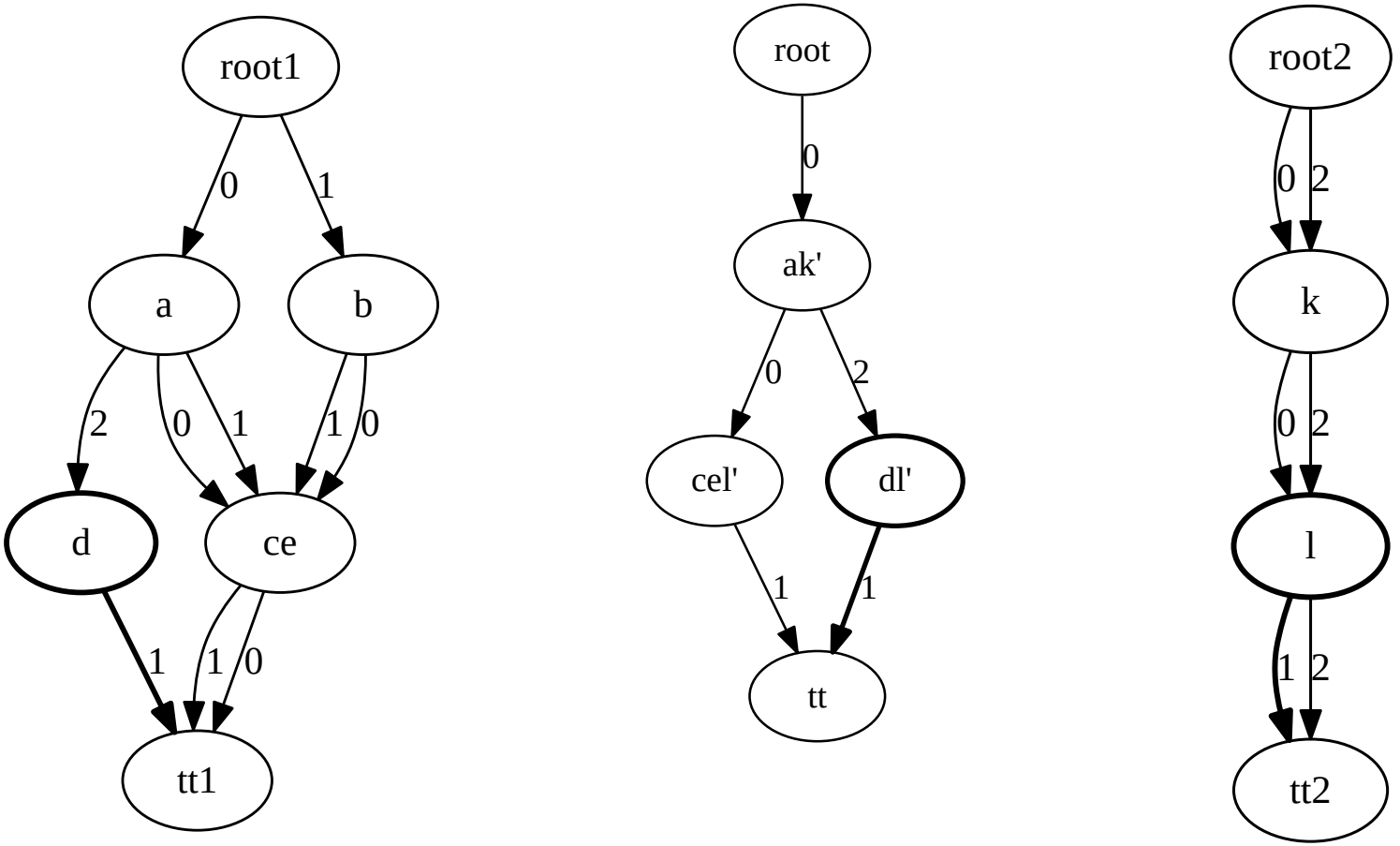
Intersection



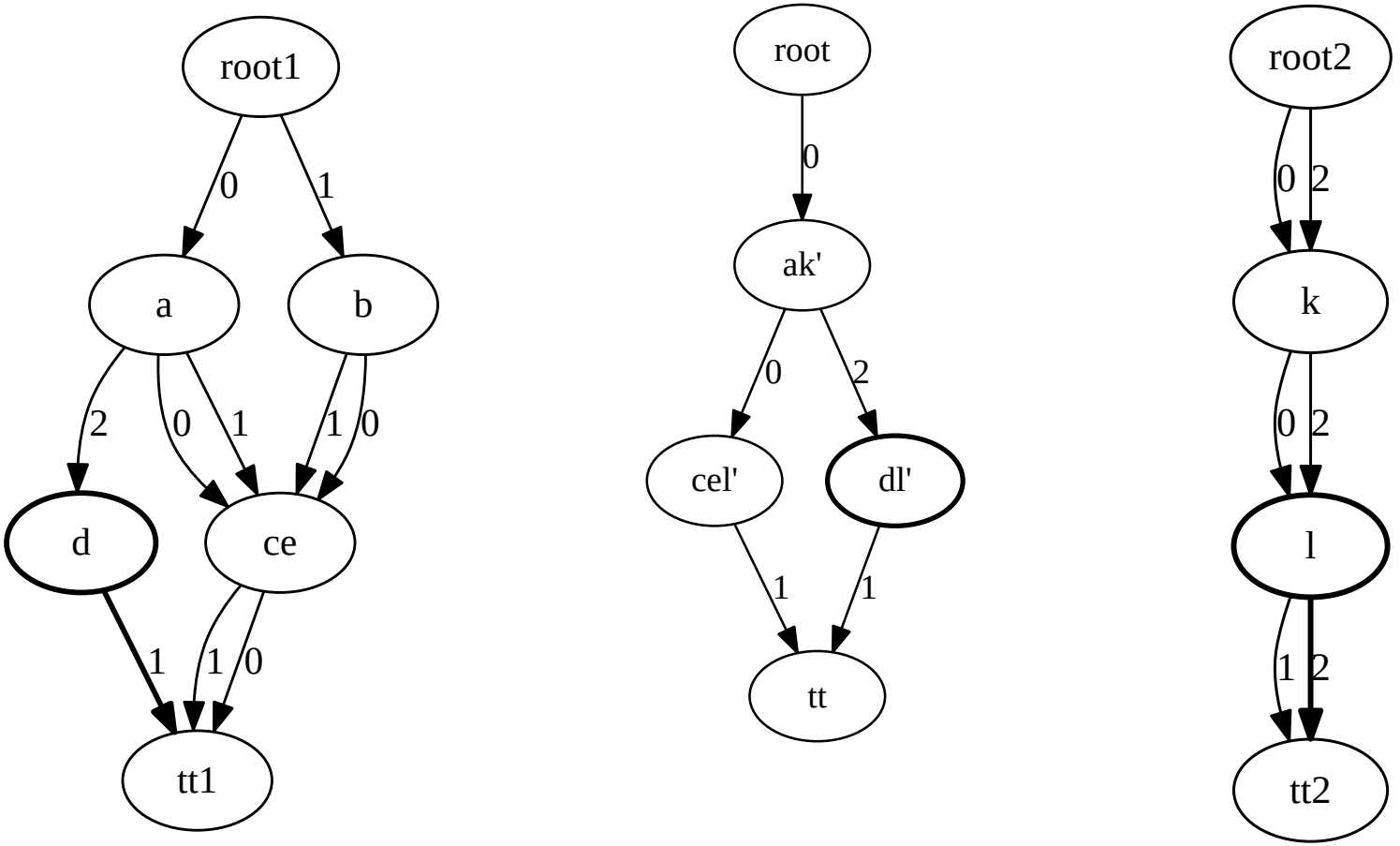
Intersection



Intersection



Intersection



Intersection

- Be careful, do not think that the number of nodes/edges of the intersection will be reduced
- The resulting MDD can be exponentially larger! Because it can be locally decompressed (TODO)

Apply

- Apply:
  - $mdd_r \leftarrow mdd_1 \oplus mdd_2$
    - **Function** of the two MDDs operands
  - Each node $x \in mdd_r$ is a combination of:
    - $x_1 \in mdd_1$
    - $x_2 \in mdd_2$
- For each node of mdd_r
  - Create outgoing edges of x (x_1, x_2) ***function of***:
    - $\omega^+(x_1)$
    - $\omega^+(x_2)$
    - For each edge (x, v, y), 4 possible states
- The **function** define the **operation**

Apply

- Apply(mdd$_1$, mdd$_2$, $\oplus$)
- Need to distinguish the level
  - Level i$\in$1… r-1
  - Level i = r
- Example of functions (T: create an edge, F: $\neg$T)

	op[0] $\neg a1 \wedge \neg a2$		op[1] $\neg a1 \wedge a2$		op[2] $a1 \wedge \neg a2$		op[3] $a1 \wedge a2$	
layer	[1..r-1]	r	[1..r-1]	r	[1..r-1]	r	[1..r-1]	r
$A \cap B$	F	F	F	F	F	F	T	T
$A \cup B$	F	F	T	T	T	T	T	T
$A - B$	F	F	F	F	T	T	T	F
$A \Delta B$	F	F	T	T	T	T	T	F
$\overline{A \cup B}$	T	T	T	F	T	F	T	F
$\overline{A \cap B}$	T	T	T	T	T	T	T	F

MDD Intersection

- Apply the algorithm on the two previous MDDs

MDD Union

- Draw (from scratch) the MDD equals to the union of the 2 previous MDDs
- In this MDD
  - There is at most one a
  - OR there is at least one b

MDD Union

- Use the apply operator for computing the union of the 2 previous MDDs

MDD Union: another example

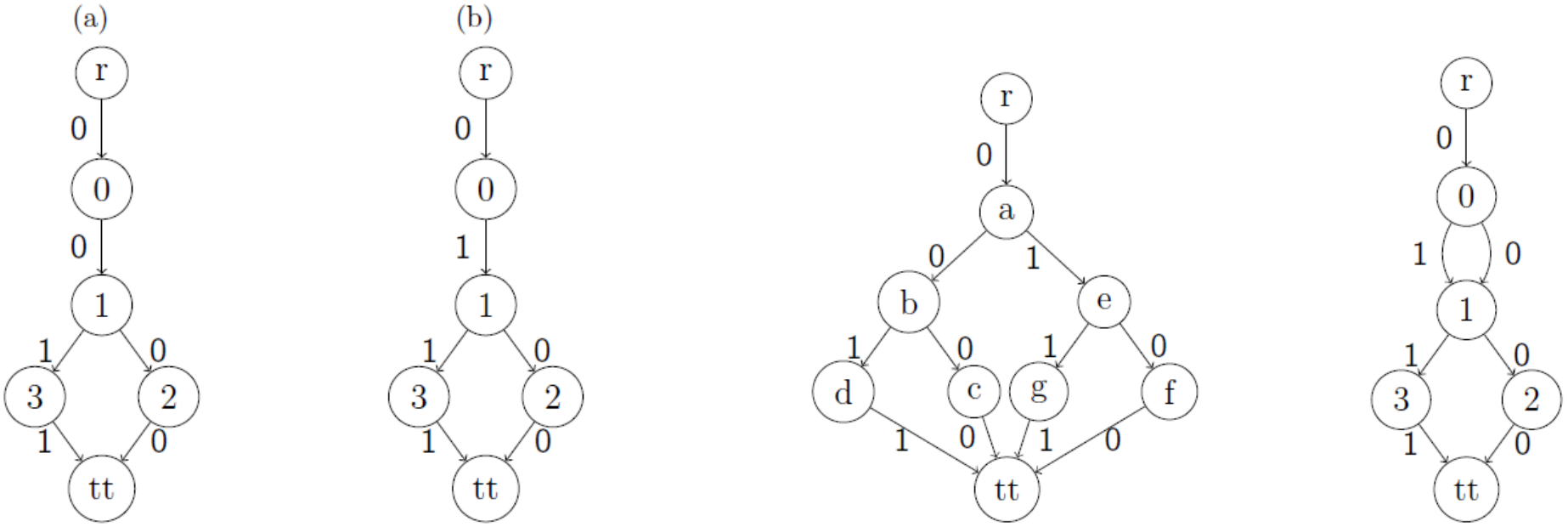


FIGURE 5.5 – The left MDD represents the application of the union of the two MDDs on the left of Figure 5.4. The right MDD is the left MDD after the application of the reduction operator.

MDD Negation

- We can use the difference operator for computing the negation of an MDD.
  - We perform Umdd – MDD where Umdd is the universal MDD (every combination is acceptable)
    - This defines the negation of the MDD, because we accept everything excepted what is in MDD

MDD Negation

- For the 2 previous MDDs
  - Compute the negation of each MDD from scratch
  - Compute the negation of each MDD by using the difference operator

Plan

- Multi-valued Decision Diagram
- Construction
- Operations (intersection, union, difference, negation)
- **Reduction algorithm**
- In-place Operations

- Modeling tools

Reduction Algorithms

- *mddify* - **Cheng & Yap** (*Constraints* 2010)
  - ***DFS***
  - Keeps a **dictionary** of the **visited nodes**
  - **Equivalent nodes** (i.e. already in the dictionary) are merged.
  - The Dictionary is an important part of the algorithm
  - Time complexity of O(***nd***), but this implies some memory issues…
    - Memory issues can be solved by losing some time
      - Dictionary using Hash table
      - Change the complexity

Reduction

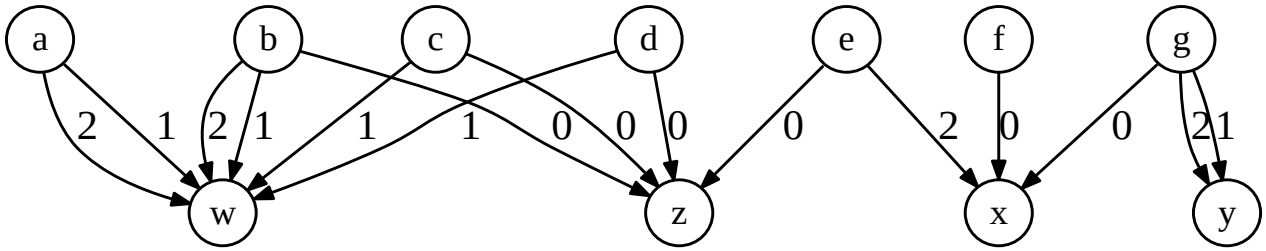
- A better idea is to use a variation of the radix sort algorithm

Reduction

- ***pReduce*:**
  - Process all the ***nodes of a layer*** instead of one by one
  - Represent any outgoing edge (n1,val,n2) by the **couple (val,n2)**
  - Represent a node n1 by it's **sorted** list of couple (value, n2)
  - **Property: nodes are merged iff they have the same sorted list**
  - Algorithm:
    - Go through the sorted list and split nodes into pack according the first difference

Reduction - How To

- Layer by Layer
  - **Bottom – up**



⇔

a	b	c	d	e	f	g
1w2w	0z1w2w	0z1w	0z1w	0z2x	0x	0x1y2y

- First Value

a	b	c	d	e	f	g
1w2w	**0**z1w2w	**0**z1w	**0**z1w	**0**z2x	**0**x	**0**x1y2y

- Second value

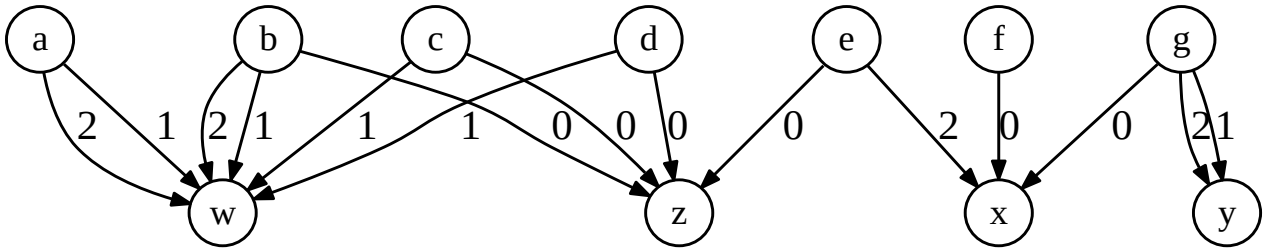
a	b	c	d	e	f	g
1w2w	0**z**1w2w	0**z**1w	0**z**1w	0**z**2x	0**x**	0**x**1y2y

- Third Value

a	b	c	d	e	f	g
1w2w	0**z**1w2w	0**z**1w	0**z**1w	0**z**2x	0x	0**x**1y2y

Reduction - How To

- Layer by Layer
  - **Bottom – up**



a	b	c	d	e	f	g
1w2w	0z**1**w2w	0z**1**w	0z**1**w	0z**2**x	0x	0x**1**y2y

  - Third Value

a	b	c	d	e	f	g
1w2w	0z1w2w	0z1w	0z1w	0z2x	0x	0x1y2y

  - Fourth Value

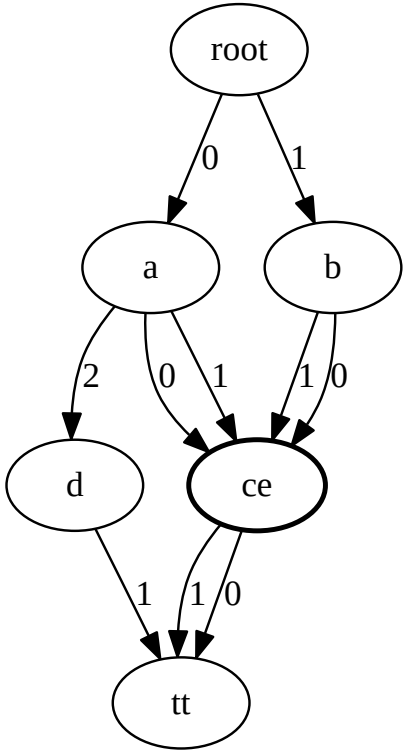
a	b	c	d	e	f	g
1w2w	0z1w**2**w	0z1w	0z1w	0z2x	0x	0x1y2y

Merge

  - Fifth Value

Reduction

- ***pReduce*:**
  - Time complexity is
    - **O(n + m + d)**
  - Space complexity is
    - **O(n + d)**



Reduction

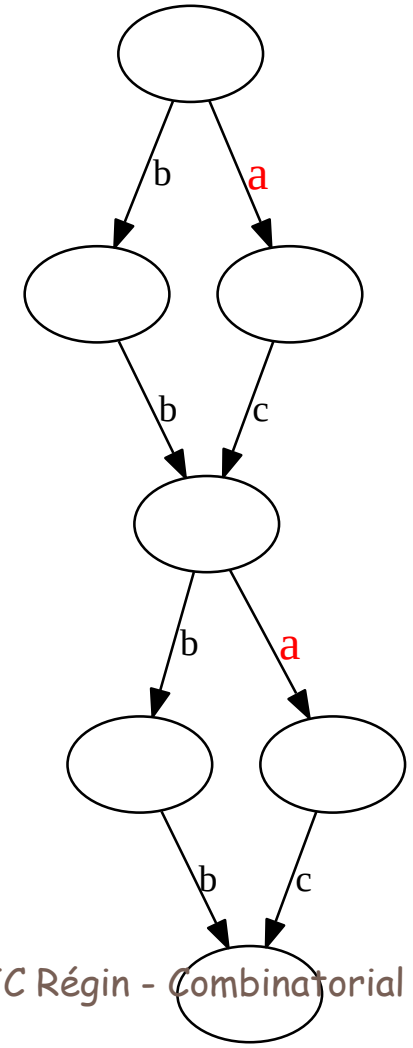
- Based on radix algorithm
- Here algorithm simpler than the radix (we just need to identify nodes having the same signature)
  - Complexity : for each letter : number of nodes of the layer or number of labels
  - Can be a lot ☹
  - We can reduce these numbers ☺ by using an indirection
    - The complexity is needed only to determine the non null entries
    - We can associate a second set with the non null entries because we do not need to have any specific ordering. This set points to the non null entries, so we can clear the non null entries efficiently

Plan

- Multi-valued Decision Diagram
- Construction
- Operations (intersection, union, difference, negation)
- Reduction algorithm
- **In-place Operations**

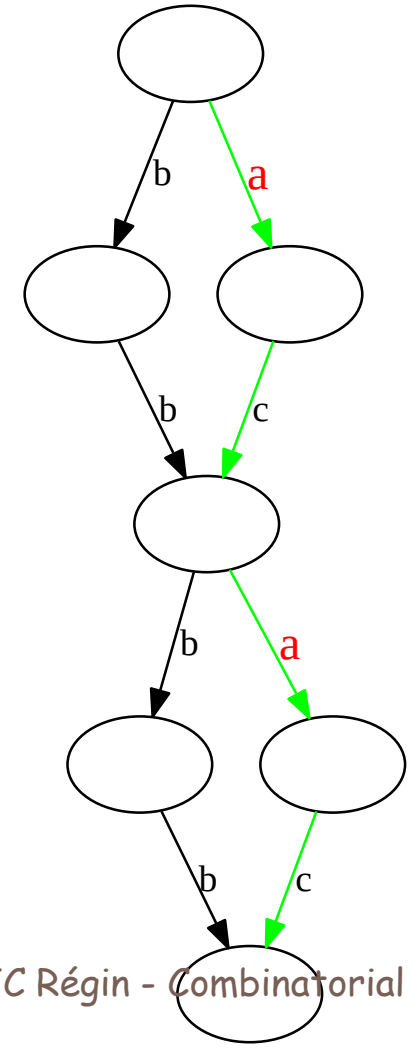
- Modeling tools

In place operations



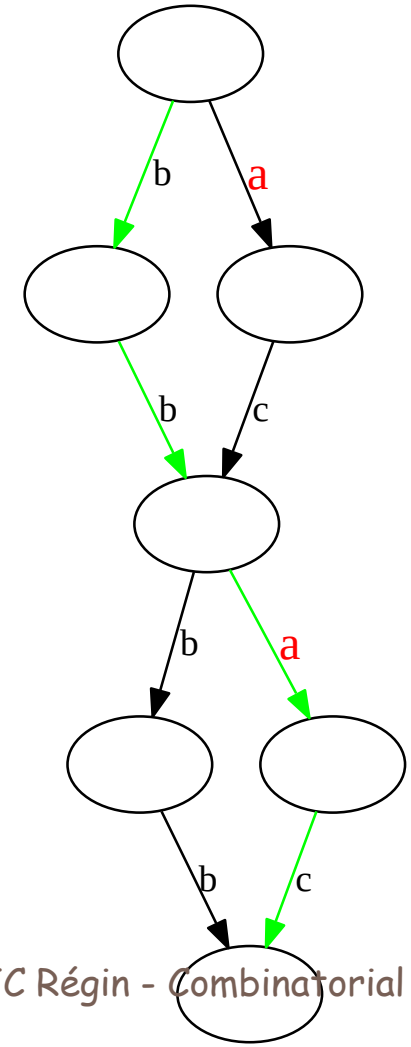
- Constraint :
  - Lighter version for example
  - Value “a” become mandatory during the search
  - Each path must contain the label “a”

In place operations



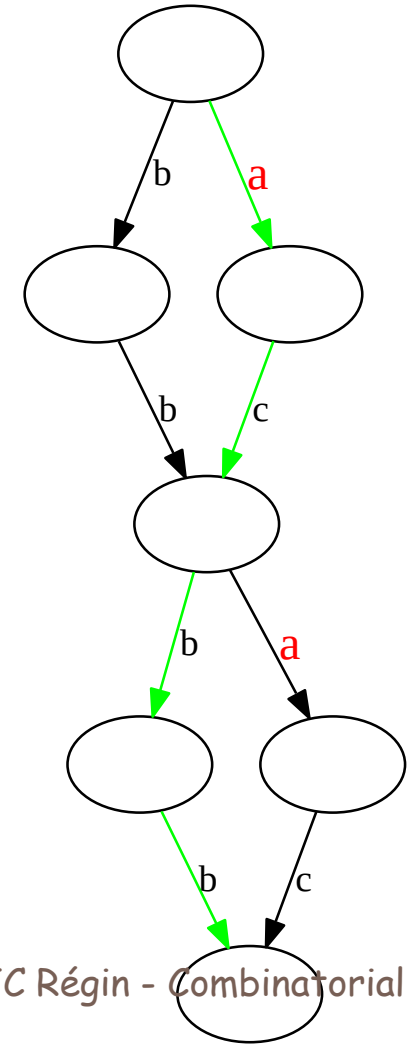
- Constraint:
  - Each path must contain the label "a"
- Valid path

In place operations



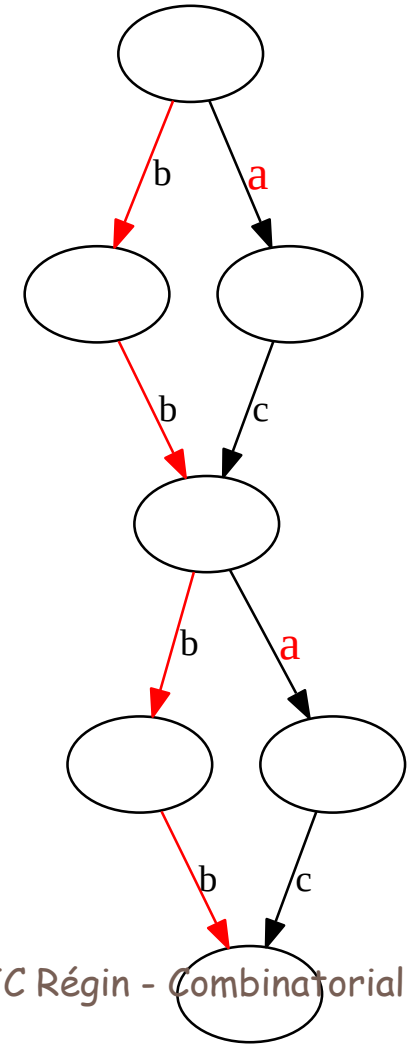
- Constraint:
  - Each path must contain the label "a"
- Valid path

In place operations



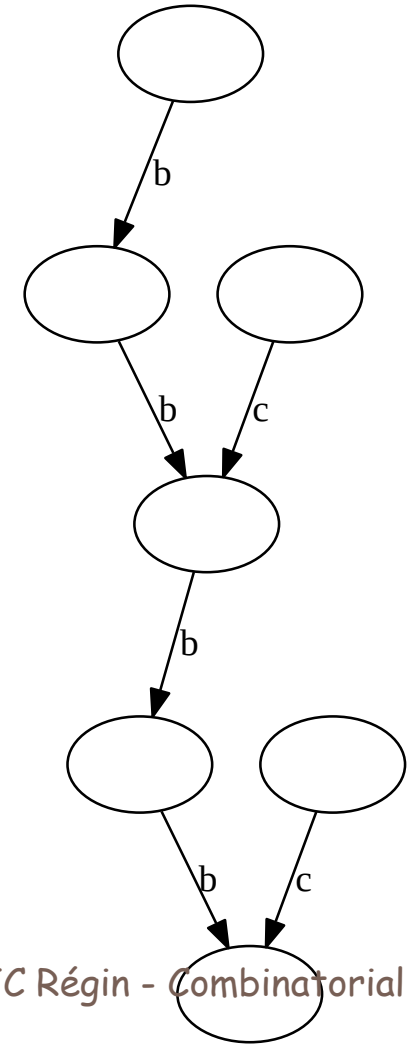
- Constraint:
  - Each path must contain the label "a"
- Valid path

In place operations



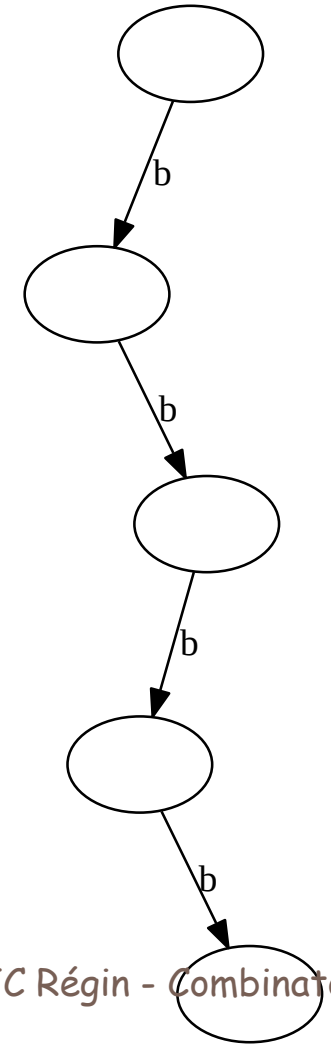
- Constraint:
  - Each path must contain the label "a"
- **NOT VALID**
  - But all this edges belong to a valid path!
  - How can I remove only the invalid paths
    - The **full valid** MDD is **not a sub-graph** of the original!
    - But the **full invalid** MDD is a **sub-graph**

In place operations



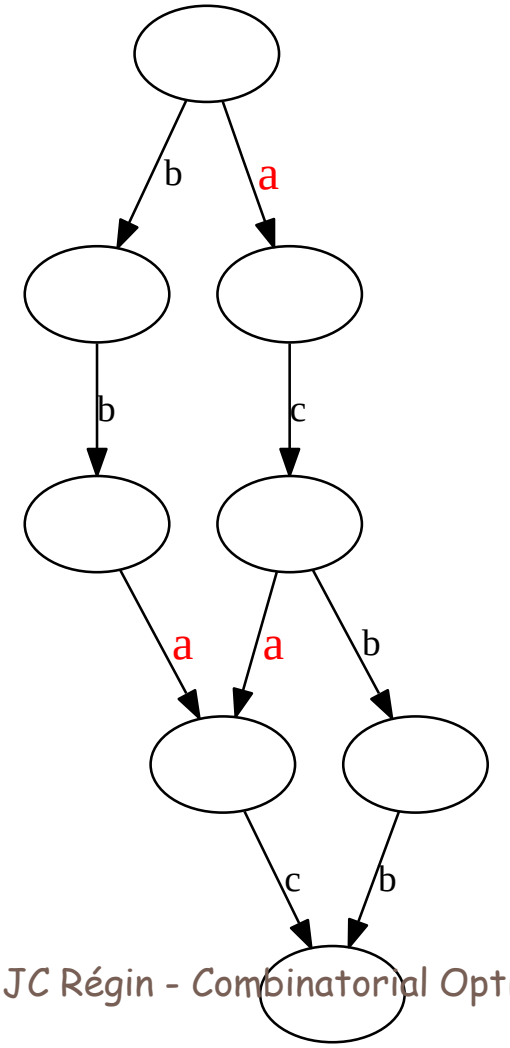
- Constraint:
  - Each path must contain the label "a"
- How can I remove only the invalid paths
  - **Delete** the mandatory edges ("a")

In place operations



- Constraint:
  - Each path must contain the label "a"
- How can I remove only the invalid paths
  - Delete the mandatory edges ("a")
  - **Propagate** the deletion

In place operations



- Constraint:
  - Each path must contain the label "a"
- How can I remove only the invalid paths
  - Delete the mandatory edges ("a")
  - Propagate the deletion
  - **Delete** the resulting MDD from the original MDD
  - **Need of a fast deletion operator**

In-Place Deletion

- ***Deletion*** of an MDD from another MDD (in-place)
  - Operator :

$$\mathbf{mdd_1 \leftarrow mdd_1 - mdd_2}$$

- Local **decompression $\mathbf{mdd_2}$** from $\mathbf{mdd_1}$
- Deletion of the common edges between $\mathbf{mdd_1}$ **&** $\mathbf{mdd_2}$
  - Only for the last layer
- Incremental reduction (local re-compression)

Path isolation

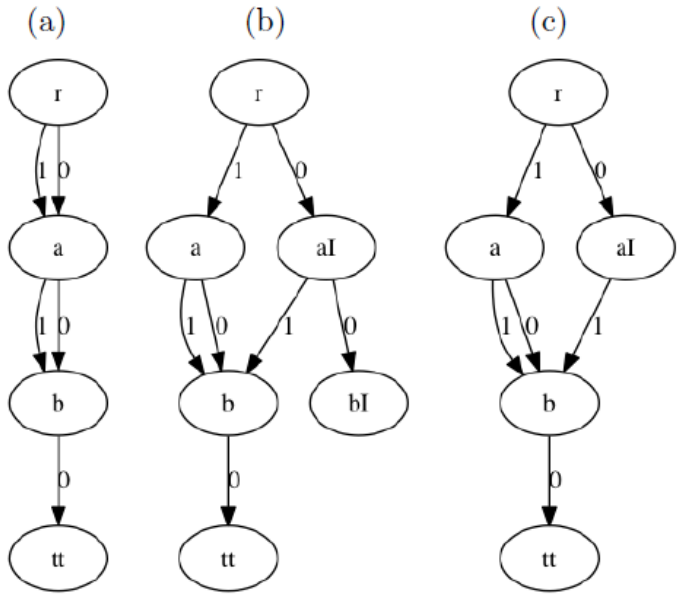
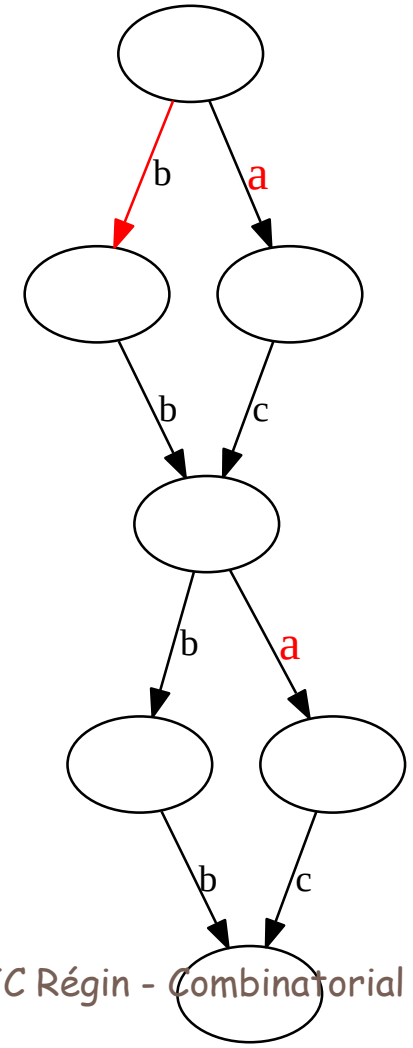


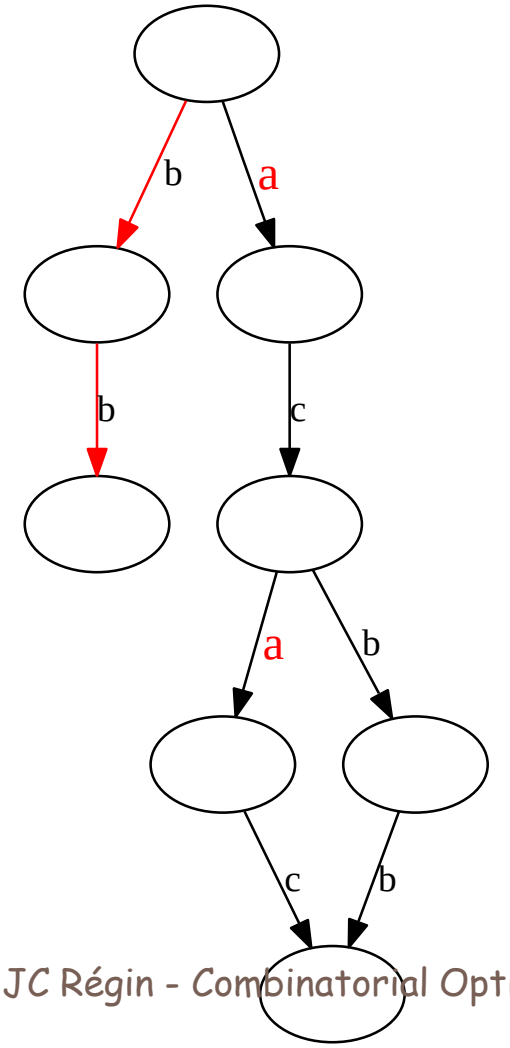
FIGURE 5.10 – Tuple {0,0,0} is removed from the MDD (a). The isolation of the path corresponding to the tuple is performed (MDD (b)) and then the reduction is applied (MDD (c)). Nodes aI and bI are created from nodes a and b during the path isolation.

In-Place Deletion



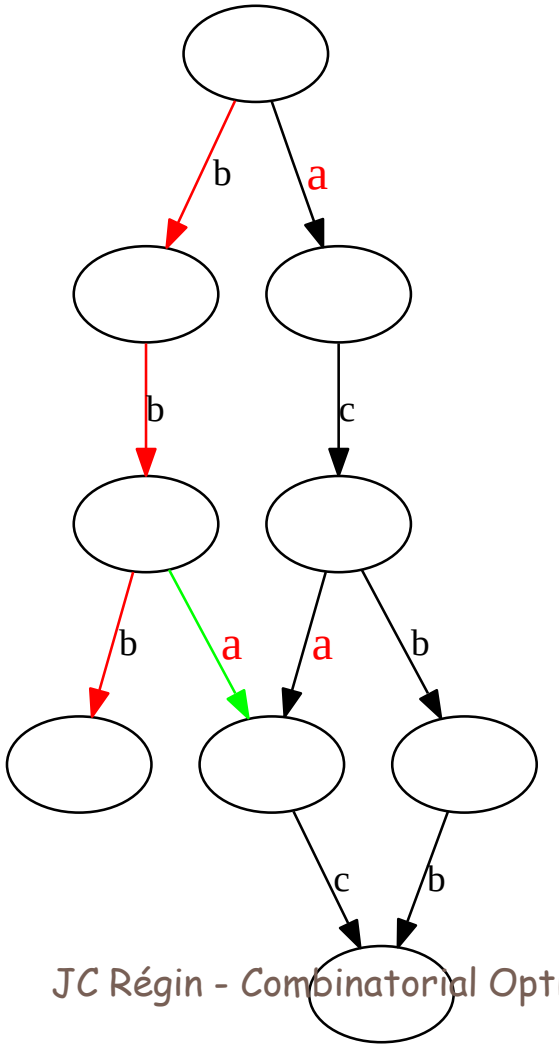
- Deletion of the tuple
  - **T = (b, b, b, b)**
- **Decompress T** in the first layer
  - Nothing to do

In-Place Deletion



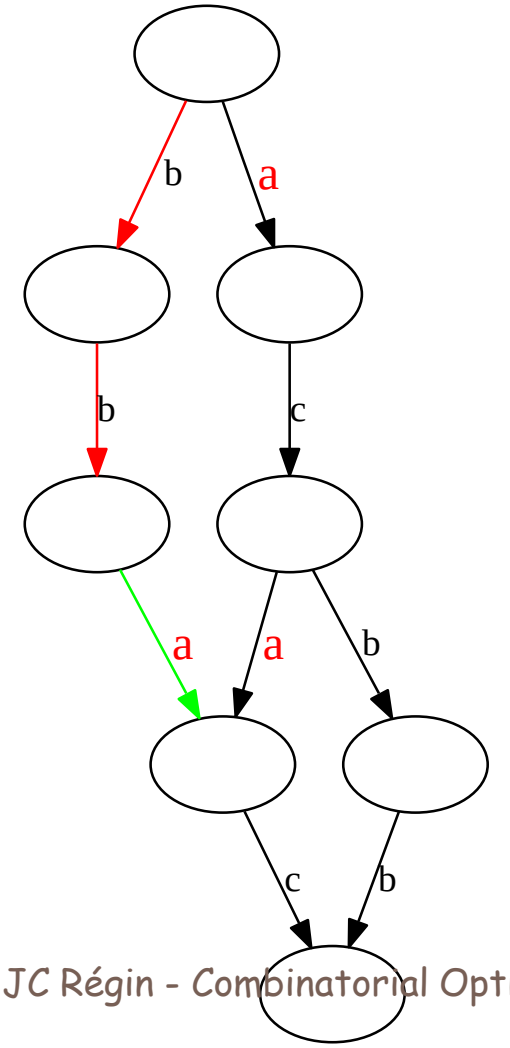
- Deletion of the tuple
  - **T = (b, b, b, b)**
- **Decompress T** in the second layer
  - Duplicate the node

In-Place Deletion



- Deletion of the tuple
  - **T = (b, b, b, b)**
- **Decompress T** in the third layer
  - The edge label by "a" isn't in **T**
    - It returns to the compressed MDD
- **Last Layer**
  - **Nothing** to do because **b ∈ T**

In-Place Deletion



- Deletion of the tuple
  - **T = (b, b, b, b)**
- **Remove** the invalid nodes
  - Nodes with no outgoing edges
- **Reduce**
  - Nothing to do here

Deletion

- Constraint example
  - 6 variables
  - 230 000 tuples
  - Deletion of 100 000 tuples
    - Table : 100 000 * 6 = 600 000 modifications
    - MDD : 135 000 modifications (average)
      - Edges and node modified, deleted or created

Addition

- Addition of a MDD (in-place)
  - $\mathbf{mdd_1} \leftarrow \mathbf{mdd_1} + \mathbf{mdd_2}$
- Locally uncompress $\mathbf{mdd_2}$ from $\mathbf{mdd_1}$ while it is possible
- Addition of the new edges from $\mathbf{mdd_2}$ into $\mathbf{mdd_1}$
- Incremental reduction (local re-compression)
- **Duality** addition/deletion
  - Adding a set T to mdd_1 is equivalent to delete T from the complementary of mdd_1

Incremental reduce

- Reduction is costly (traverse the whole graph)
  - Even when a small part of the MDD is modified
- Stay in the complexity of the operation
- Solution :
  - Reduction compare the ***outgoing edges***
    - Only the modified nodes and their neighbors can be merged
    - Use them to drive the reduction
  - Mark them during the deletion/addition
    - Or any incremental algorithm

MDD: creation

- MDD **can be created without enumerating the solution set**

Compression gain

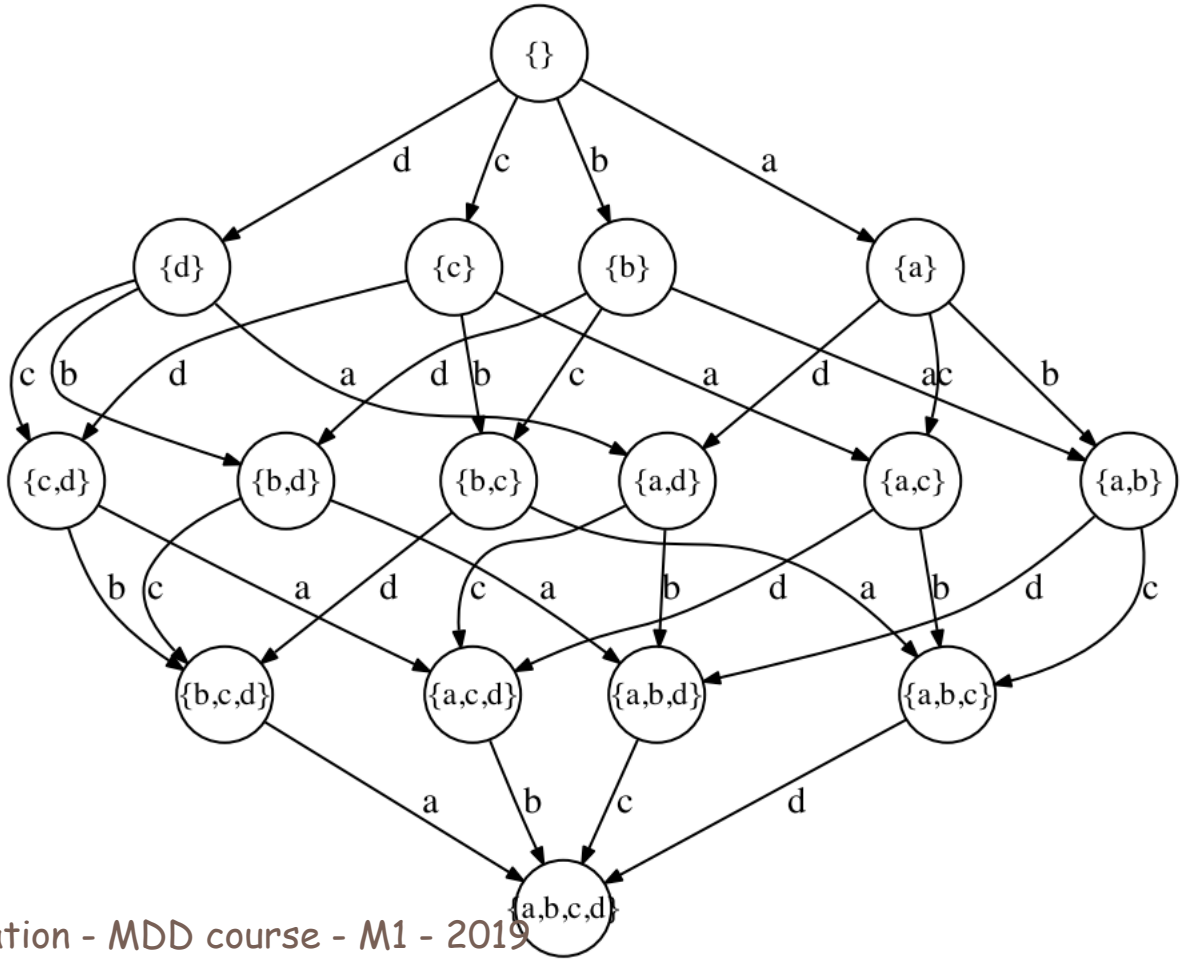
- Compression may gain an exponential factor
- It often does!

- Example (see later)
  - MDD requiring 600,000 edges for representing 10^90 solutions, that is a compression factor of 10^86.

- Sometimes it can be subtle

Alldiff constraint

- #node = 2^n
- #solutions = n!
- n!/2^n ???
- is exponential



Plan

- Multi-valued Decision Diagram
- Construction
- Operations (intersection, union, difference, negation)
- Reduction algorithm
- In-place Operations

- **Modeling tools**

MDD as a new modeling object

- Can be built from a table
  - Corresponds to an enumeration of the solutions, then compress
- Can represent complex constraints
  - Dynamic programming/Automata based constraints
    - Knapsack (**Trick** ,2003), Regular (Pesant 2004)…
  - Avoid the enumeration of the solutions. Compress in both ways
- Can be defined by combining MDDs (high level object)
  - Does not correspond to a known function
- Can be viewed as a low-level object:
  - Store some information inside the nodes
    - Like the current sum for knapsack