SEG4630 2009-2010

Tutorial 2 – Frequent Pattern Mining

Frequent Patterns

- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
  - itemset: A set of one or more items
  - k-itemset: $X = \{x_1, \ldots, x_k\}$
  - Mining algorithms
    - Apriori
    - FP-growth

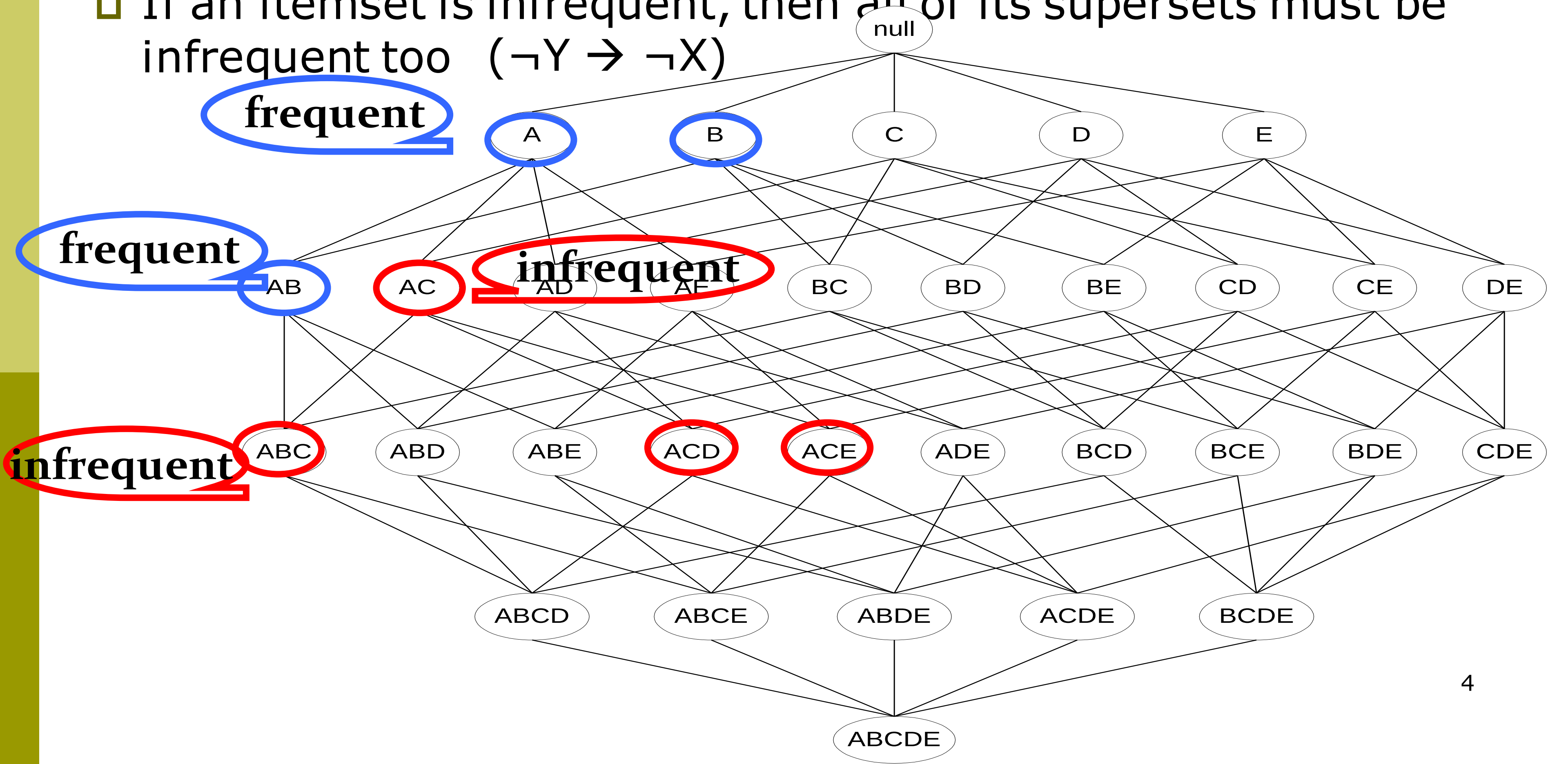
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Beer

Support & Confidence

- Support
  - (absolute) support, or, support count of X: Frequency or occurrence of an itemset X
  - (relative) support, s, is the fraction of transactions that contains X (i.e., the probability that a transaction contains X)
  - An itemset X is frequent if X's support is no less than a minsup threshold
- Confidence (association rule: $X \rightarrow Y$)
  - $sup(X \cup Y)/sup(x)$ (conditional prob.: $Pr(Y|X) = Pr(X \wedge Y)/Pr(X)$)
  - confidence, c, conditional probability that a transaction having X also contains Y
  - Find all the rules $X \rightarrow Y$ with minimum support and confidence
    - $sup(X \cup Y) \geq minsup$
    - $sup(X \cup Y)/sup(X) \geq minconf$

Apriori Principle

- If an itemset is frequent, then all of its subsets must also be frequent $(X \rightarrow Y)$
- If an itemset is infrequent, then all of its supersets must be infrequent too $(\neg Y \rightarrow \neg X)$



Apriori: A Candidate Generation & Test Approach

- Initially, scan DB once to get frequent 1-itemset
- Loop
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can be generated

Generate candidate itemsets

- **Example**

  Frequent 3-itemsets:

  {1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 4}, {1, 3, 5}, {2, 3, 4}, {2, 3, 5} and {3, 4, 5}

  - Candidate 4-itemset:

    {1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}

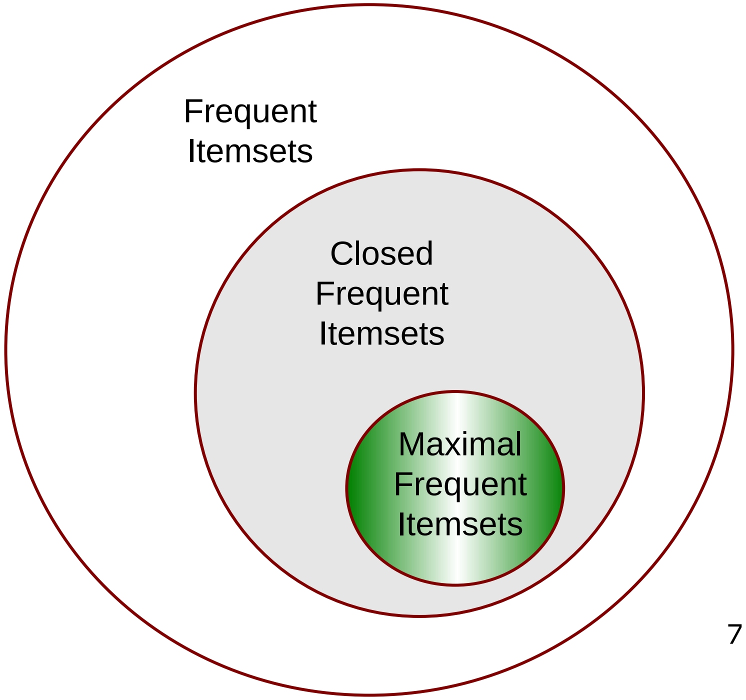
  - Which need not to be counted?

    {1, 2, 4, 5} & {1, 3, 4, 5} & {2, 3, 4, 5}

Maximal vs Closed Frequent Itemsets

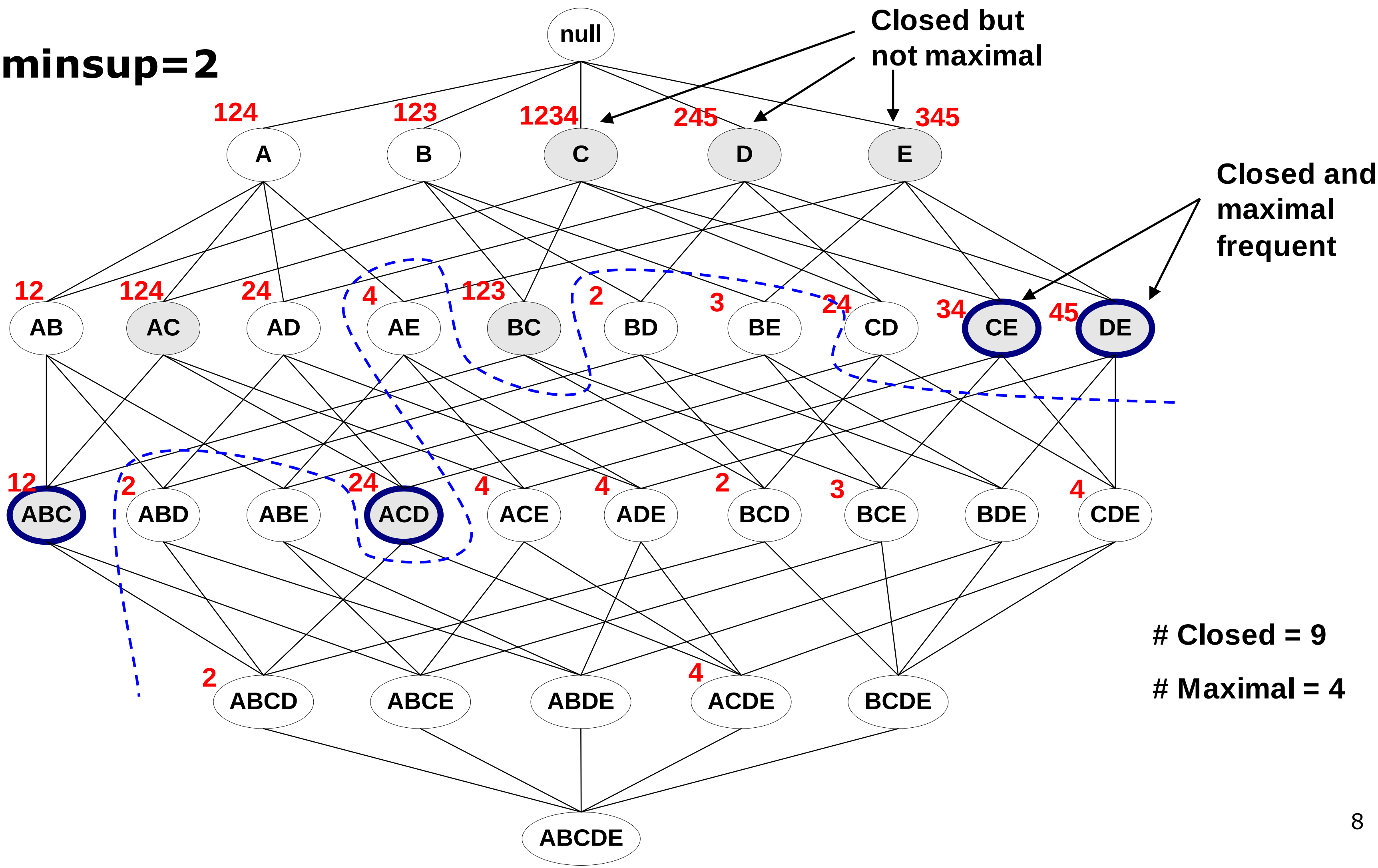
- An itemset X is a max-pattern if X is frequent and there exists no frequent super-pattern Y ɔ X
- An itemset X is closed if X is frequent and there exists no super-pattern Y ɔ X, with the same support as X

Closed Frequent Itemsets are Lossless: the support for any frequent itemset can be deduced from the closed frequent itemsets



Maximal vs Closed Frequent Itemsets

minsup=2



 # Closed = 9

 # Maximal = 4

Algorithms to find frequent pattern

- **Apriori**: uses a generate-and-test approach – generates candidate itemsets and tests if they are frequent
  - Generation of candidate itemsets is expensive (in both space and time)
  - Support counting is expensive
    - Subset checking (computationally expensive)
    - Multiple Database scans (I/O)
- **FP-Growth**: allows frequent itemset discovery without candidate generation. Two step:
  - 1.Build a compact data structure called the FP-tree
    - 2 passes over the database
  - 2.extracts frequent itemsets directly from the FP-tree
    - Traverse through FP-tree

Pattern-Growth Approach: Mining Frequent Patterns Without Candidate Generation

- The FP-Growth Approach
  - Depth-first search (Apriori: Breadth-first search)
  - Avoid explicit candidate generation

FP-Growth approach:

- For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
- Repeat the process on each newly created conditional FP-tree
- Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

Fp-tree construatioin:

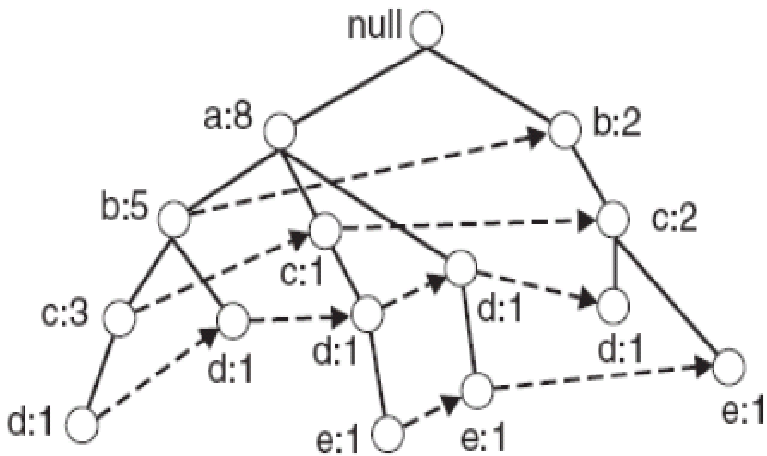
- Scan DB once, find frequent 1-itemset (single item pattern)
- Sort frequent items in frequency descending order, f-list
- Scan DB again, construct FP-tree

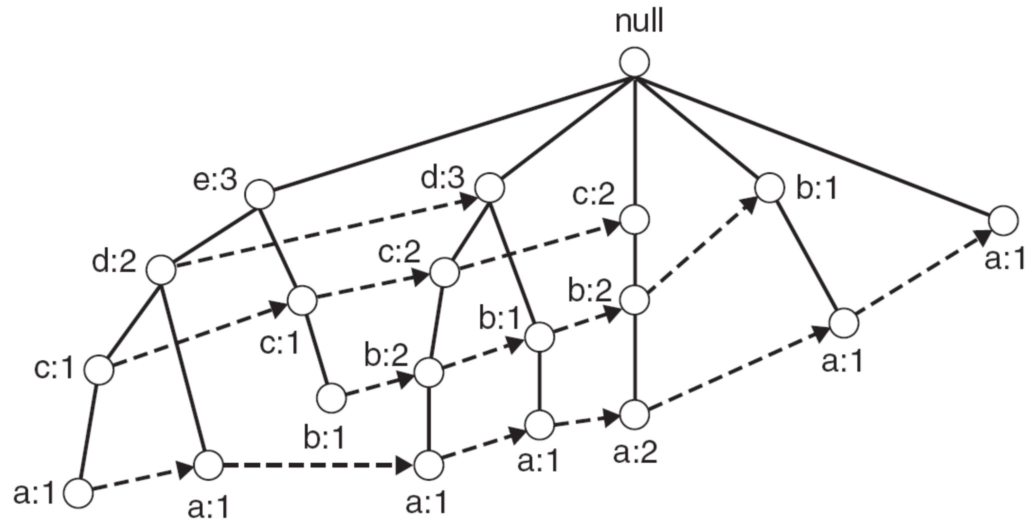
FP-tree Size

- The size of an FPtree is typically smaller than the size of the uncompressed data because many transactions often share a few items in common
  - **Bestcase** scenario: All transactions have the same set of items, and the FPtree contains only a single branch of nodes.
  - **Worstcase** scenario: Every transaction has a unique set of items. As none of the transactions have any items in common, the size of the FPtree is effectively the same as the size of the original data.
- The size of an FPtree also depends on how the items are ordered

Example

- FP-tree with item descending ordering
- FP-tree with item ascending ordering





Transaction Data Set

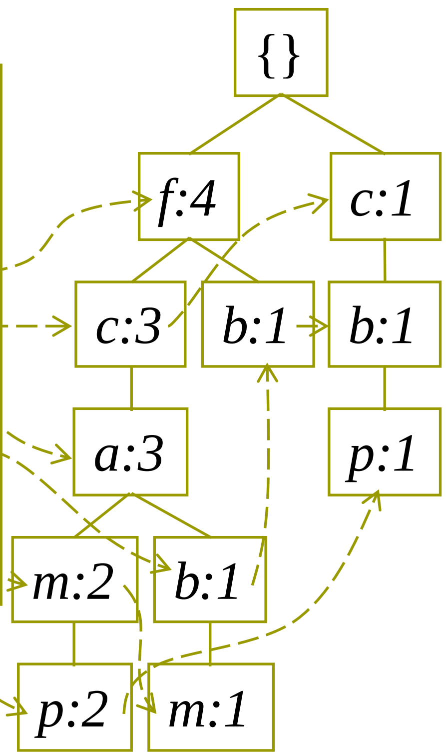
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Find Patterns Having p From P-conditional Database

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item *p*
- Accumulate all of *transformed prefix paths* of item *p* to form *p*'s conditional pattern base

Header Table

Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



***Conditional* pattern bases**

item	***cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

FP-Growth

1	***f, c, a, m, p***
2	***f, c, a, b, m***
3	***f, b***
4	***c, b, p***
5	***f, c, a, m, p***

1	***f, c, a, m***
4	***c, b***
5	***f, c, a, m***

+ p

1	***f, c, a, m***
2	***f, c, a, b, m***
3	***f, b***
4	***c, b***
5	***f, c, a, m***

1	***f, c, a***
2	***f, c, a, b***
5	***f, c, a***

+ m

1	***f, c, a***
2	***f, c, a, b***
3	***f, b***
4	***c, b***
5	***f, c, a***

2	***f, c, a***
3	***f***
4	***c***

+ b

1	***f, c, a***
2	***f, c, a***
3	***f***
4	***c***
5	***f, c, a***

1	***f, c***
2	***f, c***
5	***f, c***

+ a

1	***f, c***
2	***f, c***
3	***f***
4	***c***
5	***f, c***

FP-Growth

1	***f, c, a, m, p***
2	***f, c, a, b, m***
3	***f, b***
4	***c, b, p***
5	***f, c, a, m, p***

1	***f, c, a, m***	**+ *p***
4	***c, b***	
5	***f, c, a, m***	

(1)

1	***f, c, a***	**+ *m***
2	***f, c, a, b***	
5	***f, c, a***	

(2)

2	***f, c, a***	**+ *b***
3	***f***	
4	***c***	

(3)

1	***f, c***	**+ *a***
2	***f, c***	
5	***f, c***	

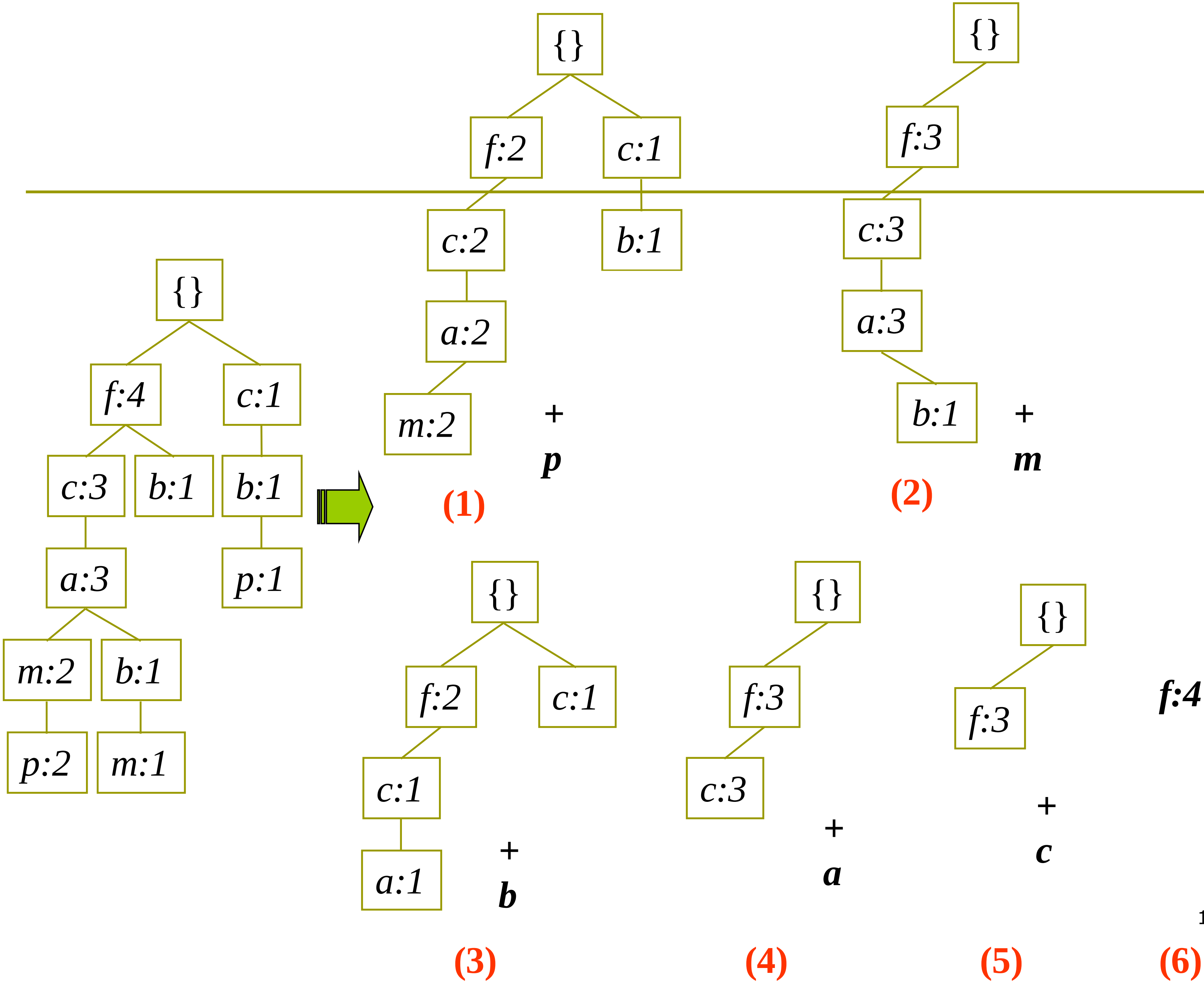
(4)

1	***f***	**+ *c***
2	***f***	
4		
5	***f***	

(5)

f: 1,2,3,5

(6)


{}
f:4
c:1
c:3
b:1
b:1
a:3
p:1
m:2
b:1
p:2
m:1
{}
f:2
c:1
c:2
b:1
a:2
m:2
+
p
(1)
{}
f:3
c:3
a:3
b:1
+
m
(2)
{}
f:2
c:1
c:1
a:1
+
b
(3)
{}
f:3
c:3
+
a
(4)
{}
f:3
+
c
(5)
f:4
(6)

min_sup = 3

1	***f, c, a, m, p***
2	***f, c, a, b, m***
3	***f, b***
4	***c, b, p***
5	***f, c, a, m, p***

1	***f, c, a, m***
4	***c, b***
5	***f, c, a, m***

+ *p*

1	***c***
4	***c***
5	***c***

+ *p*

p: 3
cp: 3

1	***f, c, a***
2	***f, c, a, b***
5	***f, c, a***

+ *m*

1	***f, c, a***
2	***f, c, a***
5	***f, c, a***

+ *m*

m: 3
fm: 3
cm: 3
am: 3
fcm: 3
fam: 3
cam: 3
fcam: 3

2	***f, c, a***
3	***f***
4	***c***

+ *b*

b: 3

1	***f, c***
2	***f, c***
5	***f, c***

+ *a*

a: 3
fa: 3
ca: 3
fca: 3

1	***f***
2	***f***
4	
5	***f***

+ *c*

c: 4
fc: 3

f: 1,2,3,5

f: 4