Mining Trees

CS 145

Fall 2014

Candidate generation

- Given an equivalence class of k-subtrees, how do we generate candidate (k+1)-subtrees?
- Main idea: consider each ordered pair of elements in the class for extension, including self extension
    - Sort elements by node label and position

Class extension

Let P be a prefix class with encoding P, and let (x, i) and (y, j) denote any two elements in the class. Let Px denote the class representing extensions of element (x, i). Define a join operator $\otimes$ on the two elements, denoted $(x, i)\otimes(y, j)$, as follows:

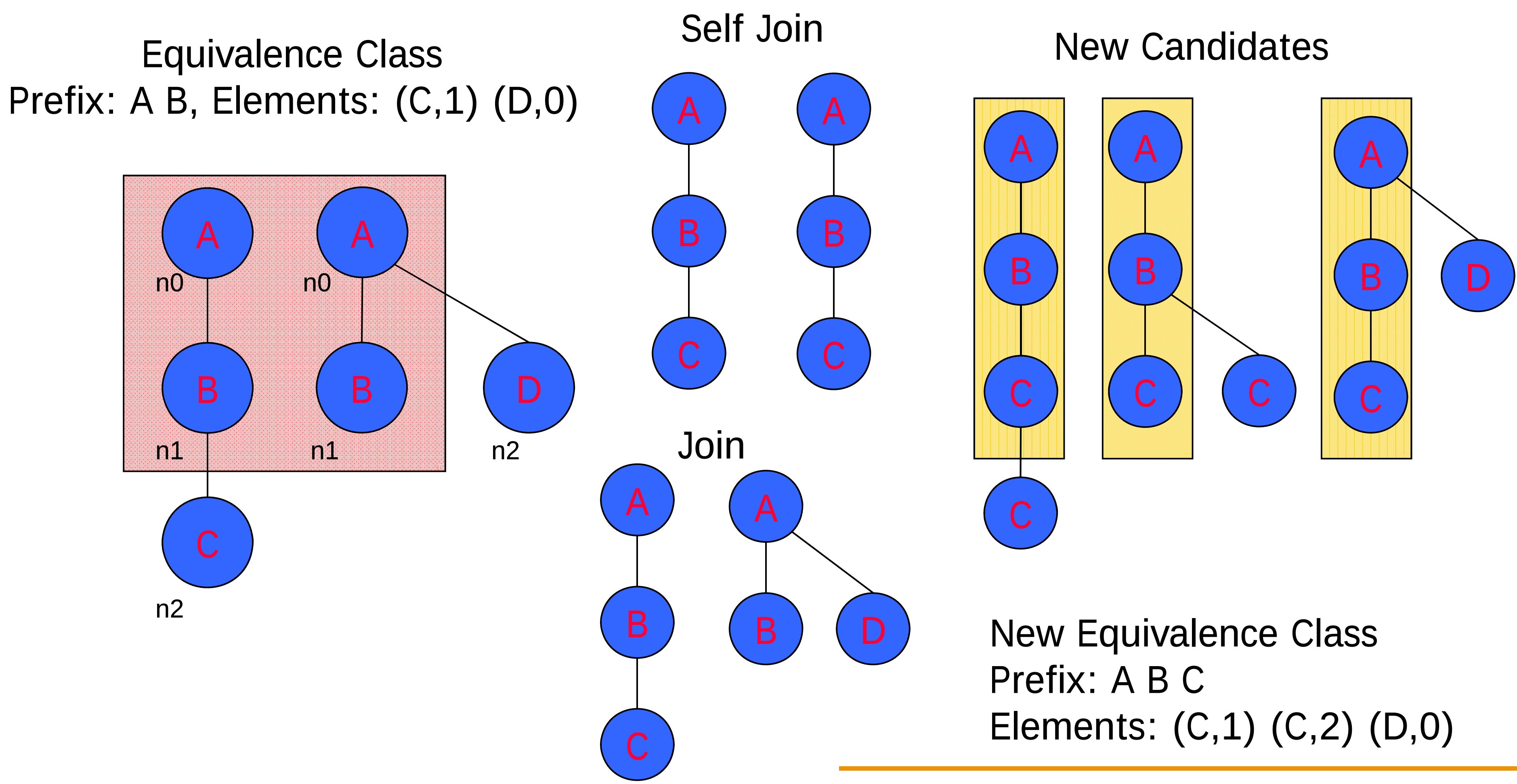
case I – $(i = j)$:

1. If $P \,!\!= \emptyset$, add (y, j) and $(y, \boldsymbol{ni})$ to class $[Px]$, where ni is the depth-first number for node (x, i) in tree Px.
2. If $P = \emptyset$, add $(y, j + 1)$ to $[Px]$.

case II – $(i > j)$: add (y, j) to class $[Px]$.

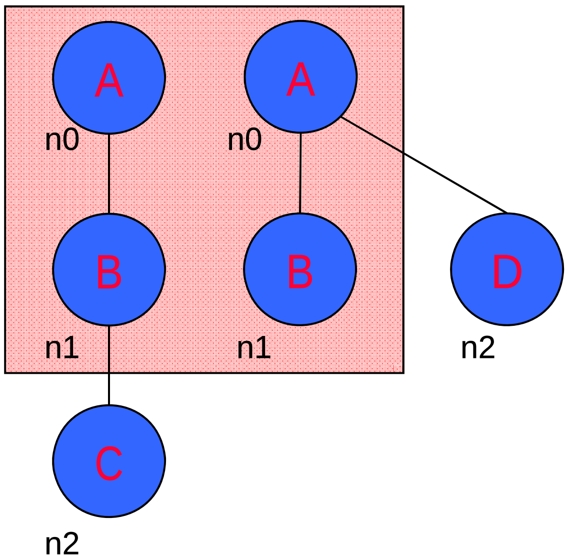
case III – $(i < j)$: no new candidate is possible in this case.

Candidate Generation (Join operator)

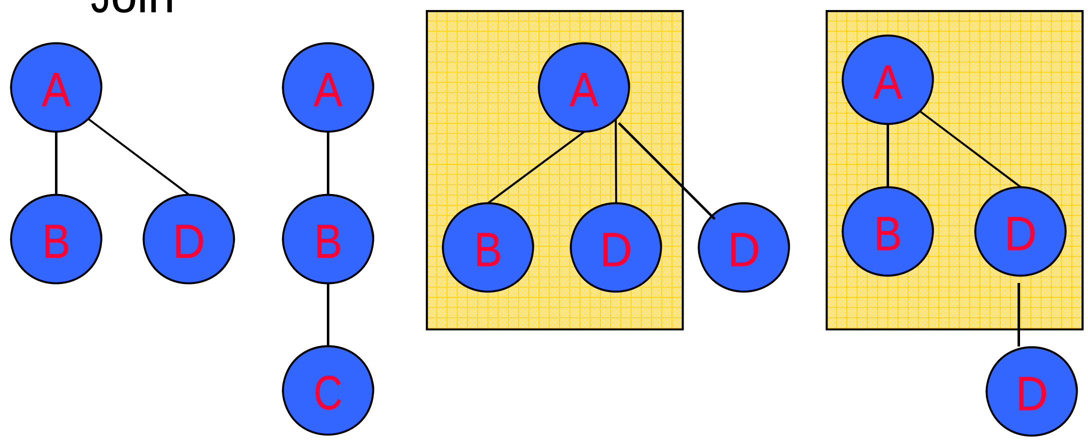


Candidate Generation (Join operator)

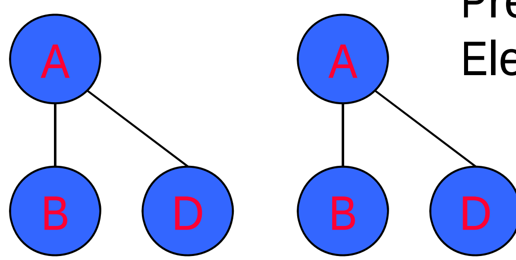
Equivalence Class
Prefix: A B, Elements: (C,1) (D,0)



Join

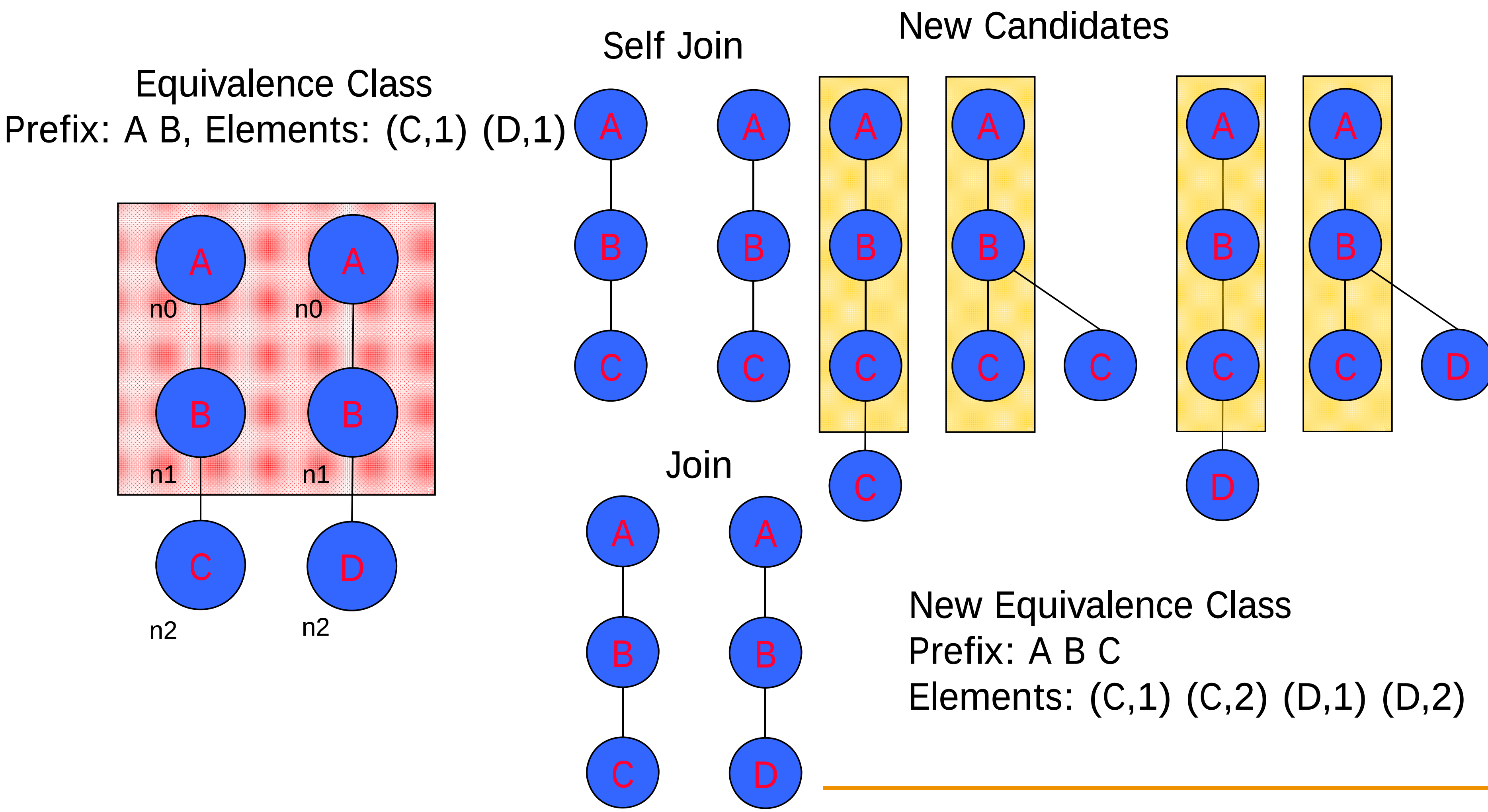


Self Join

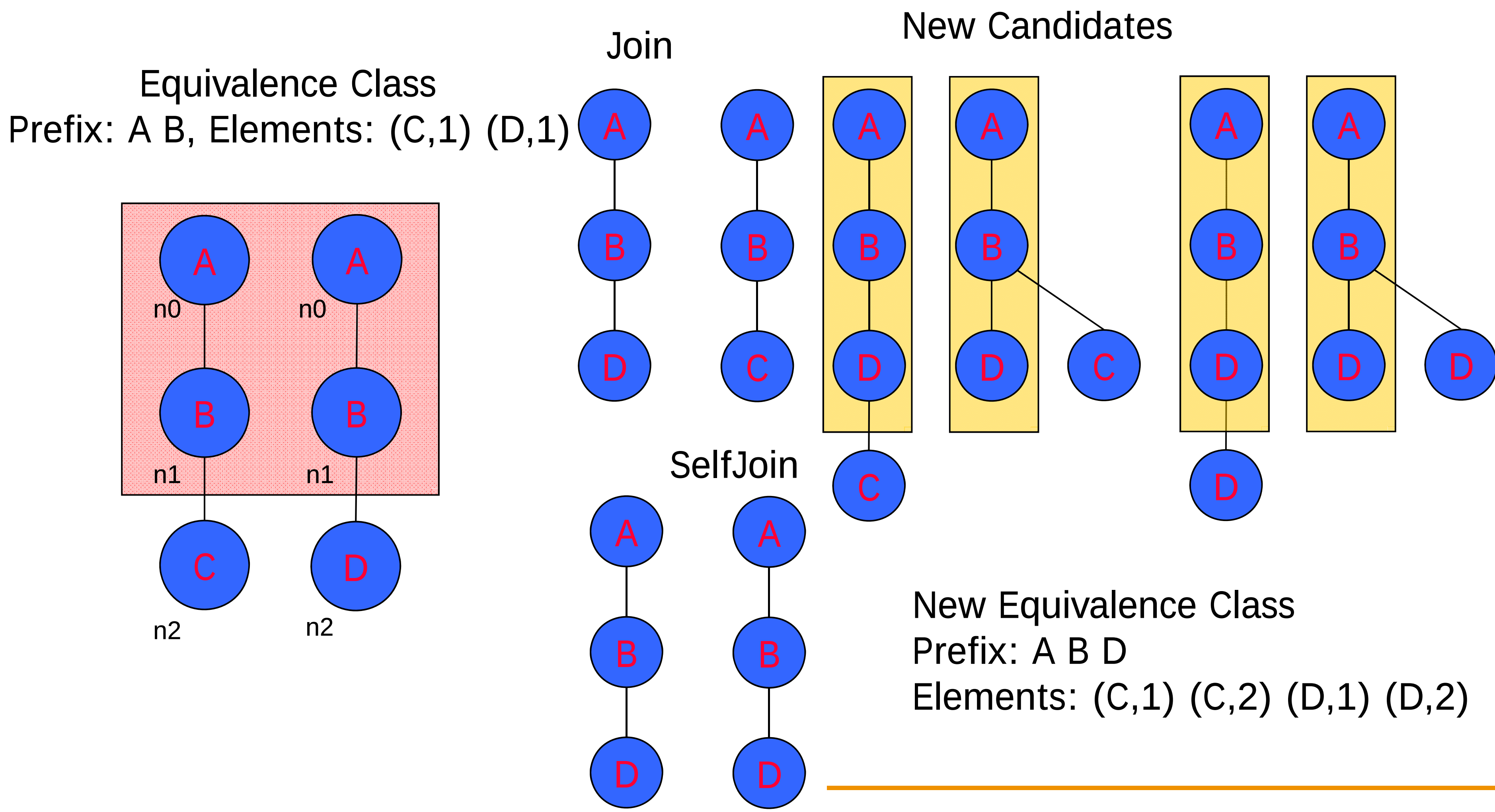


New Equivalence Class
Prefix: A B -1 D
Elements: (D,0) (D,2)

Candidate Generation (Join operator)



Candidate Generation (Join operator)



Apriori Style TreeMiner

TreeMiner (D, *minsup*):
    $F_1 = \{$ frequent 1-subtrees $\}$;
    $F_2 = \{$ classes $[P]_1$ of frequent 2-subtrees $\}$;
    for all $[P]_1 \in E$ **do** *Enumerate-Frequent-Subtrees*($[P]_1$);

Enumerate-Frequent-Subtrees($[P]$):
    for each element $(x, i) \in [P]$ **do**
        $[P_x] = \emptyset$;
        for each element $(y, j) \in [P]$ **do**
            $\mathbf{R} = \{(x, i) \otimes (y, j)\}$;
            $\mathcal{L}(\mathbf{R}) = \{\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)\}$;
            if for any $R \in \mathbf{R}$, R is frequent **then**
                $[P_x] = [P_x] \cup \{R\}$;
        Enumerate-Frequent-Subtrees($[P_x]$);

Depth first version of TreeMiner

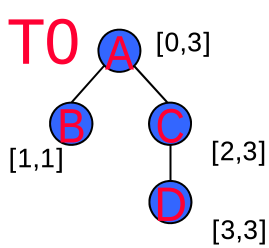
Scope-list for each element of a class

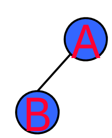
(t, m, s)

Tree id

Match label of the prefix

Scope of the last item





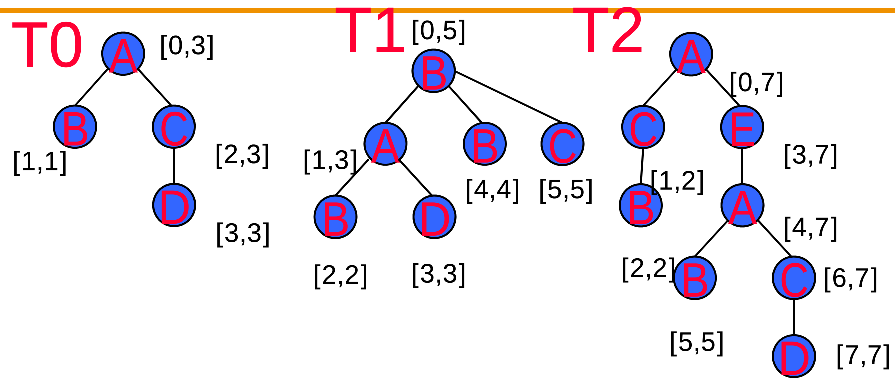
Prefix = A
Elements = (B, 0)

Tree Id,	Prefix Match Label,	Last Node Scope
0	0	[1,1]

Scope-List Joins

- Join elements (x, i) and (y, j)
  - In-scope test
    - Add (y, j+1) when i = j
      - Add y as a child of x
    - Check whether there exist (t_x, m_x, s_x) and (t_y, m_y, s_y) s.t.
      - $t_x = t_y$
      - $m_x = m_y$
      - $s_y \subset s_x$,

Frequency Computation: Scope List Joins – In Scope



Minsup = 3 (100%)

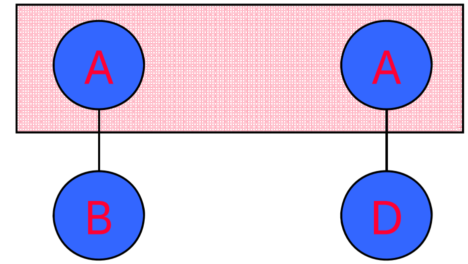
Equivalence Class: Prefix =∅

Elements: (A,-1) (B,-1) (C,-1) (D,-1)

A	B	C	D
0, [0,3]	0, [1,1]	0, [2,3]	0,[3,3]
1, [1,3]	1, [0,5]	1, [5,5]	1,[3,3]
2, [0,7]	1, [2,2]	2, [1,2]	2,[7,7]
2, [4,7]	1, [4,4]	2, [6,7]	
	2, [2,2]		
	2, [5,5]		

Prefix = A

Elements = (B, 0) (D,0)



B	D
0, 0, [1,1]	0, 0, [3,3]
1, 1, [2,2]	1, 1, [3,3]
2, 0, [2,2]	2, 0, [7,7]
2, 0, [5,5]	2, 4, [7,7]
2, 4, [5,5]	

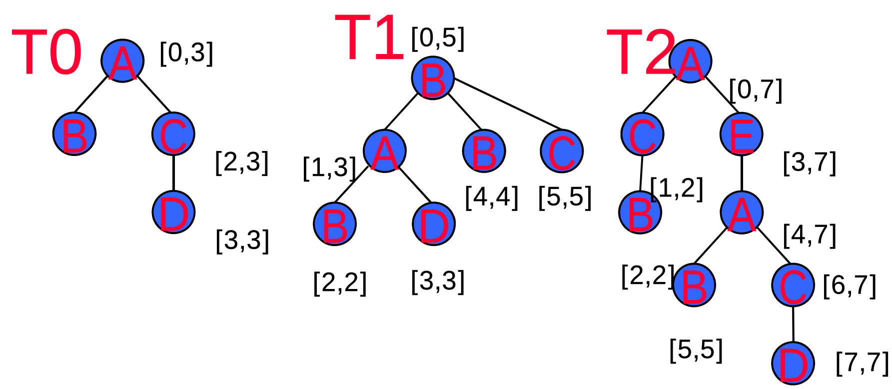
Count = 3

Tree Id,	Prefix Match Label,	Last Node Scope
0	0	[1,1]

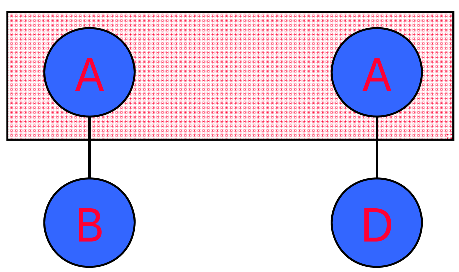
Scope-List Joins

- Join elements (x, i) and (y, j)
  - out-scope test
    - Add (y, j)
    - Add y as a sibling of x
    - Check whether there exist (t_x, m_x, s_x) and (t_y, m_y, s_y) s.t.
      - $t_x = t_y$
      - $m_x = m_y$
      - $s_y > s_x$,

Scope List Joins: Out Scope

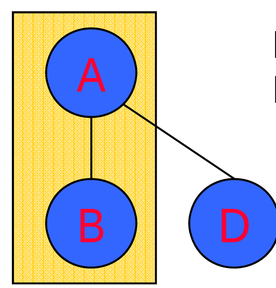


Prefix = A
Elements = (B, 0) (D,0)



0, 0, [1,1]	0, 0, [3,3]
1, 1, [2,2]	1, 1, [3,3]
2, 0, [2,2]	2, 0, [7,7]
2, 0, [5,5]	2, 4, [7,7]
2, 4, [5,5]	

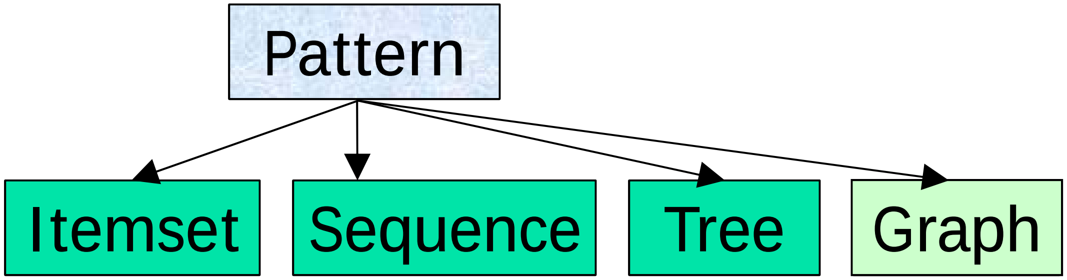
Prefix = AB
Elements = (D,0)



0, 01, [3,3]
1, 12, [3,3]
2, 02, [7,7]
2, 05, [7,7]
2, 45, [7,7]

Generic Pattern Class

- Types of patterns
  - Itemset
  - Sequence
  - Tree & Graphs
  - Define your own!



- Apriori property
  - Every sub-pattern of a frequent pattern is frequent
  - Every super-pattern of an infrequent pattern is infrequent
- Order to explore the solution space
  - Breadth-first level-wise
  - Depth-first projection based