THE CENTER FOR AUTOMATA PROCESSING

Frequent Subtree Mining on the Automata Processor: Challenges and Opportunities

Elaheh Sadredini, Reza Rahimi, Ke Wang, Kevin Skadron
Department of Computer Science
University of Virginia

Presenting in:
International Conference on Supercomputing, June 13-16, 2017

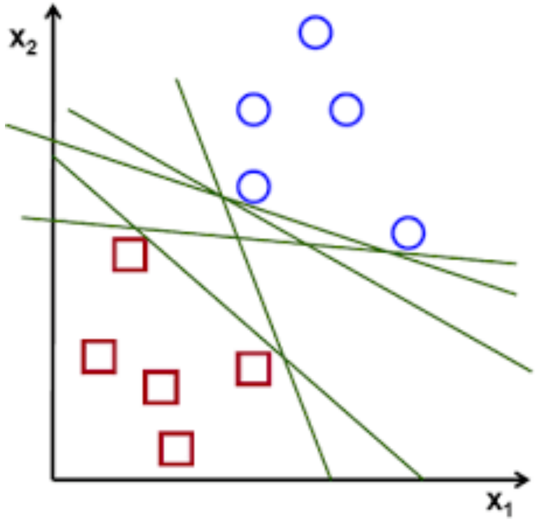
Motivation

Motivation

Motivation

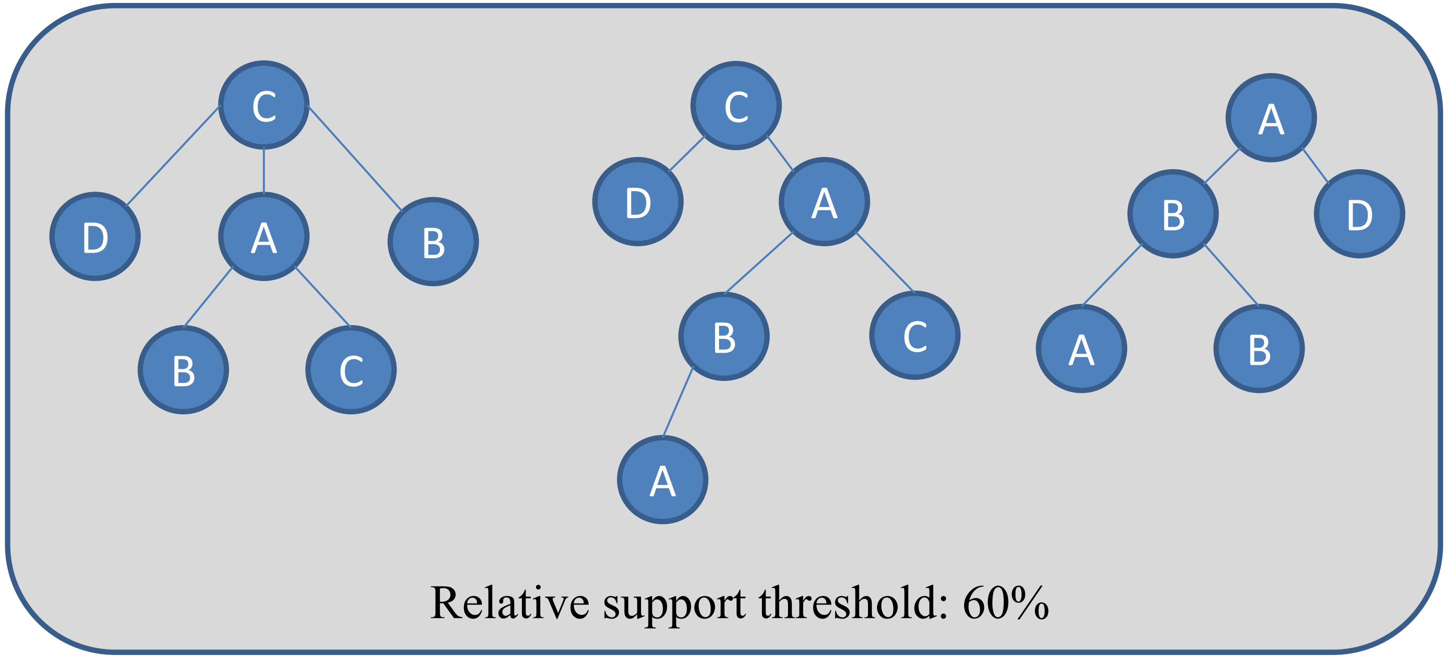
Google amazon

Walmart



What Is the Frequent Subtree Mining (FTM)?

- To efficiently enumerate all frequent subtrees in a forest (database of trees) according to a given minimum support
- The support of a subtree is the number of subtrees in D that contains one occurrence of S
- A subtree S is frequent if its support is more than or equal to a user specified minimum support value

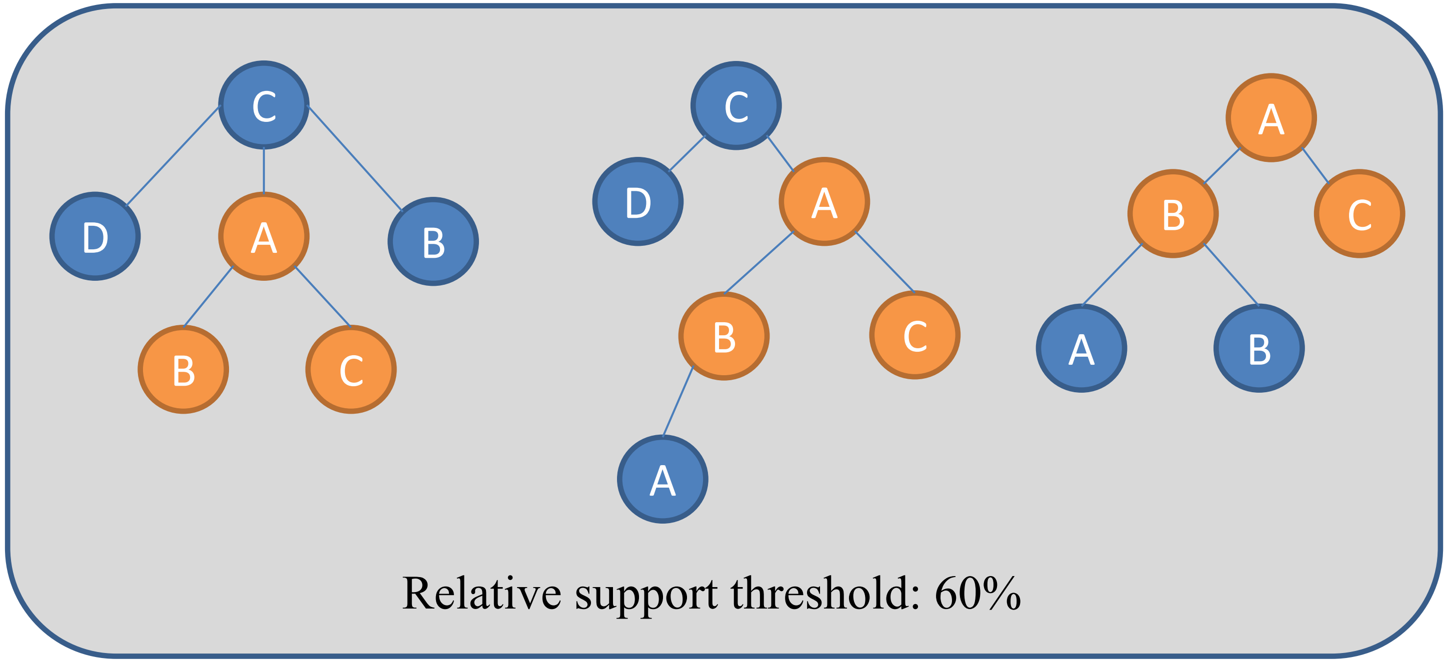


What Is the Frequent Subtree Mining (FTM)?

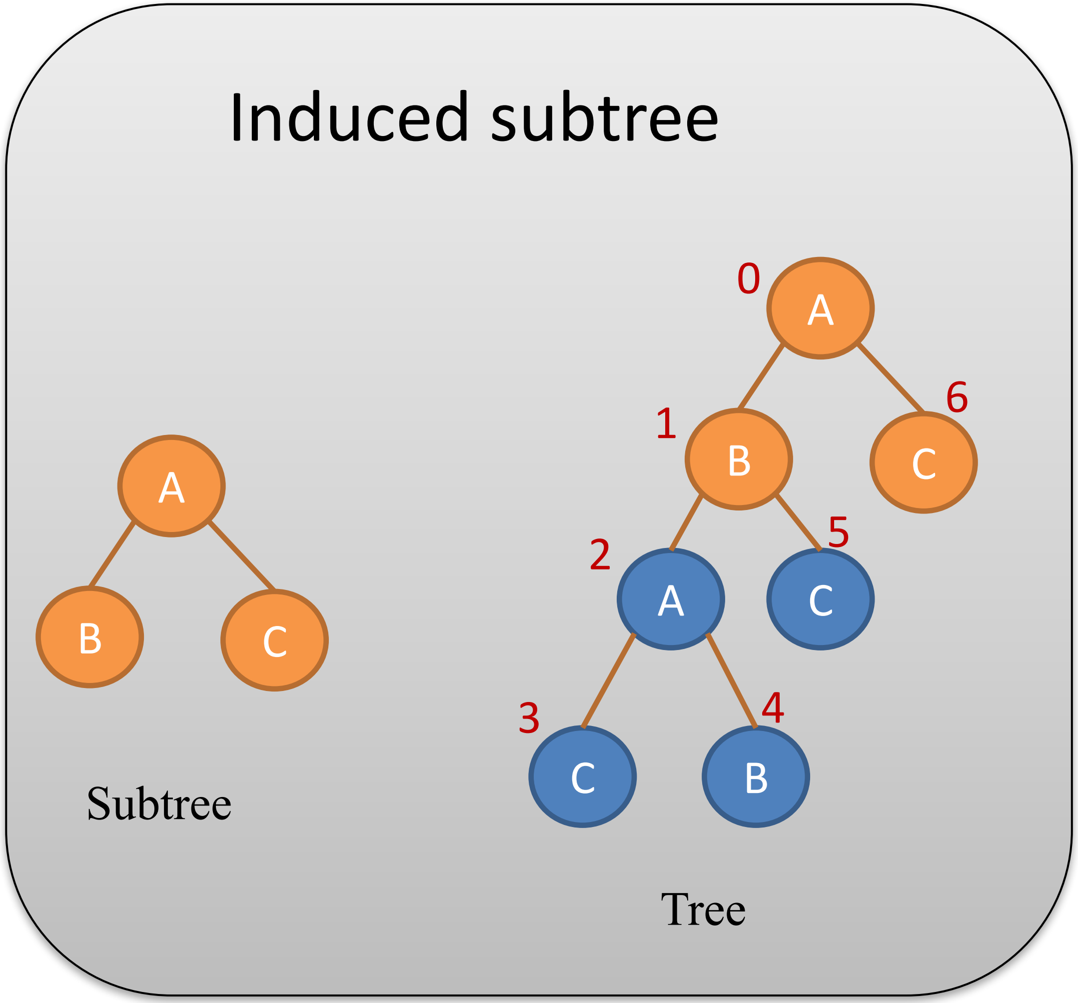
- To efficiently enumerate all frequent subtrees in a forest (database of trees) according to a given minimum support
- The support of a subtree is the number of subtrees in D that contains one occurrence of S
- A subtree S is frequent if its support is more than or equal to a user specified minimum support value

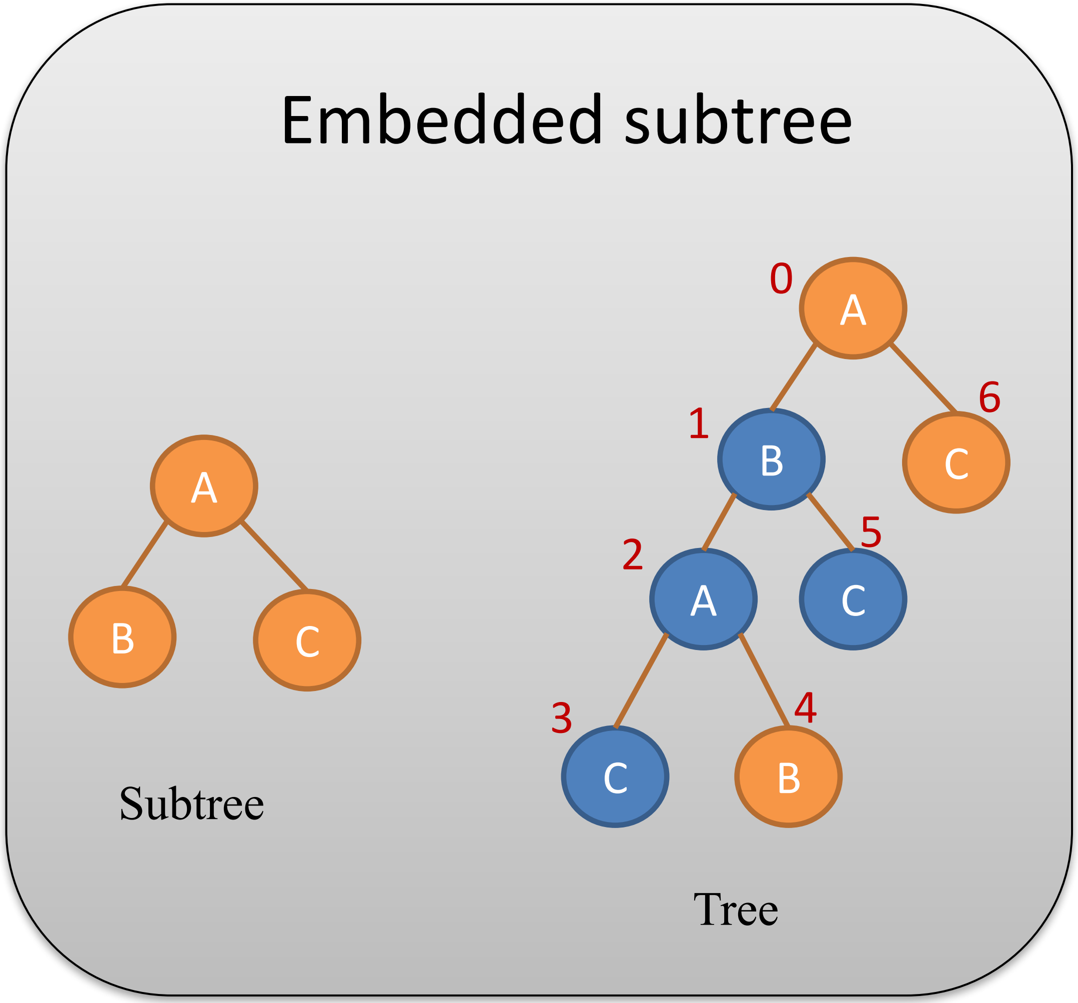
Support = 3

Is frequent: ✓



Preliminaries



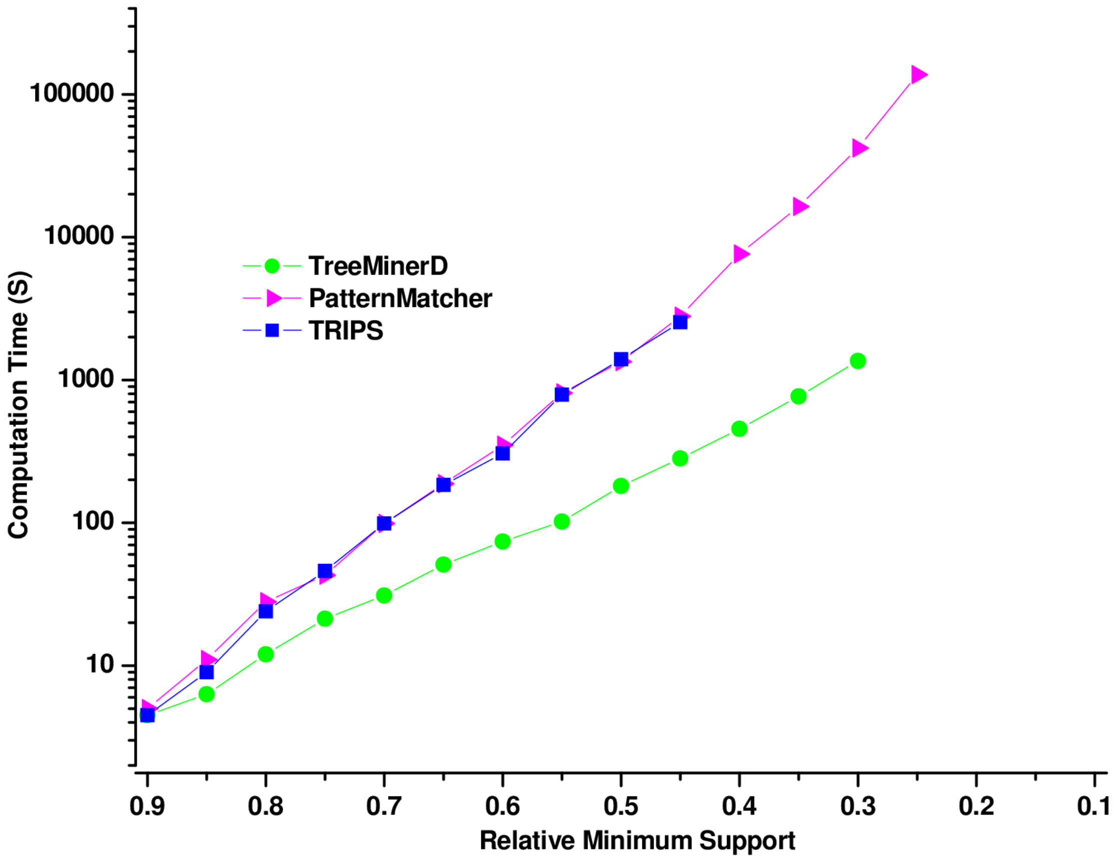


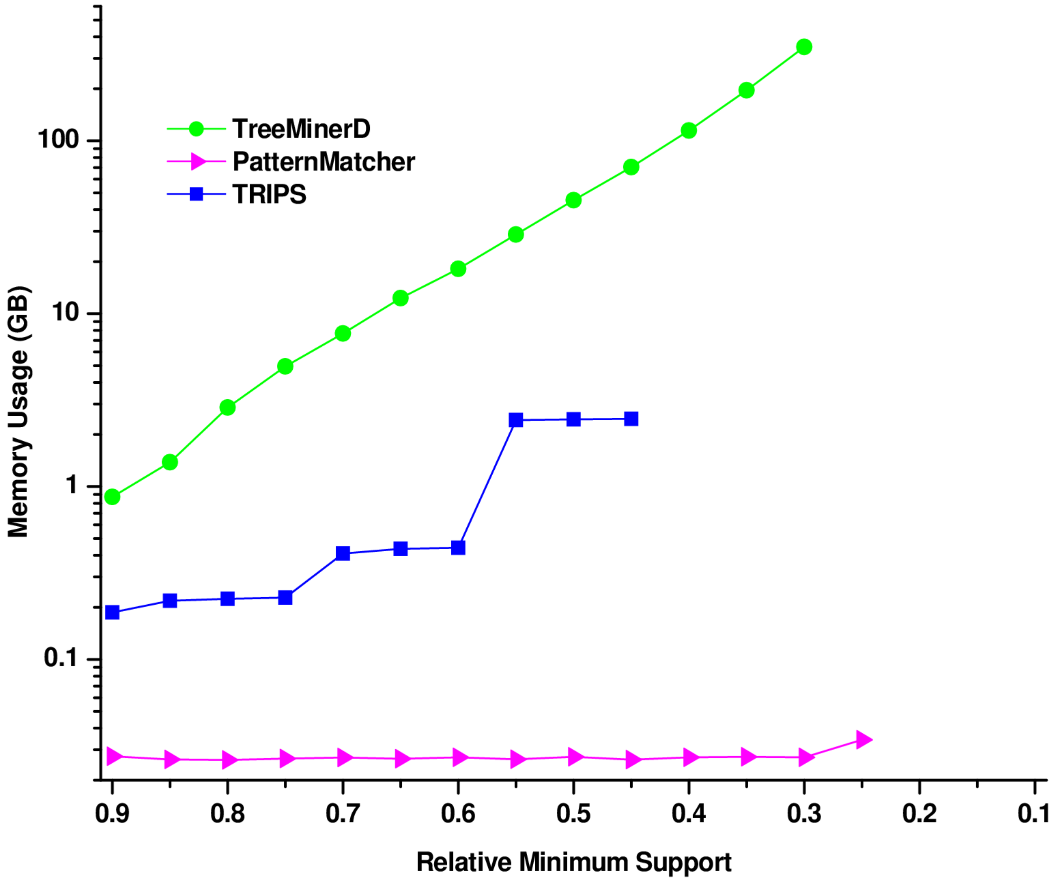
Issues with the Current FTM Solutions (1)

	Pros	Cons
BFS	Massive pruning Memory efficient	Multi-pass of dataset slow
DFS	Fast	Little pruning opportunity Memory-hungry

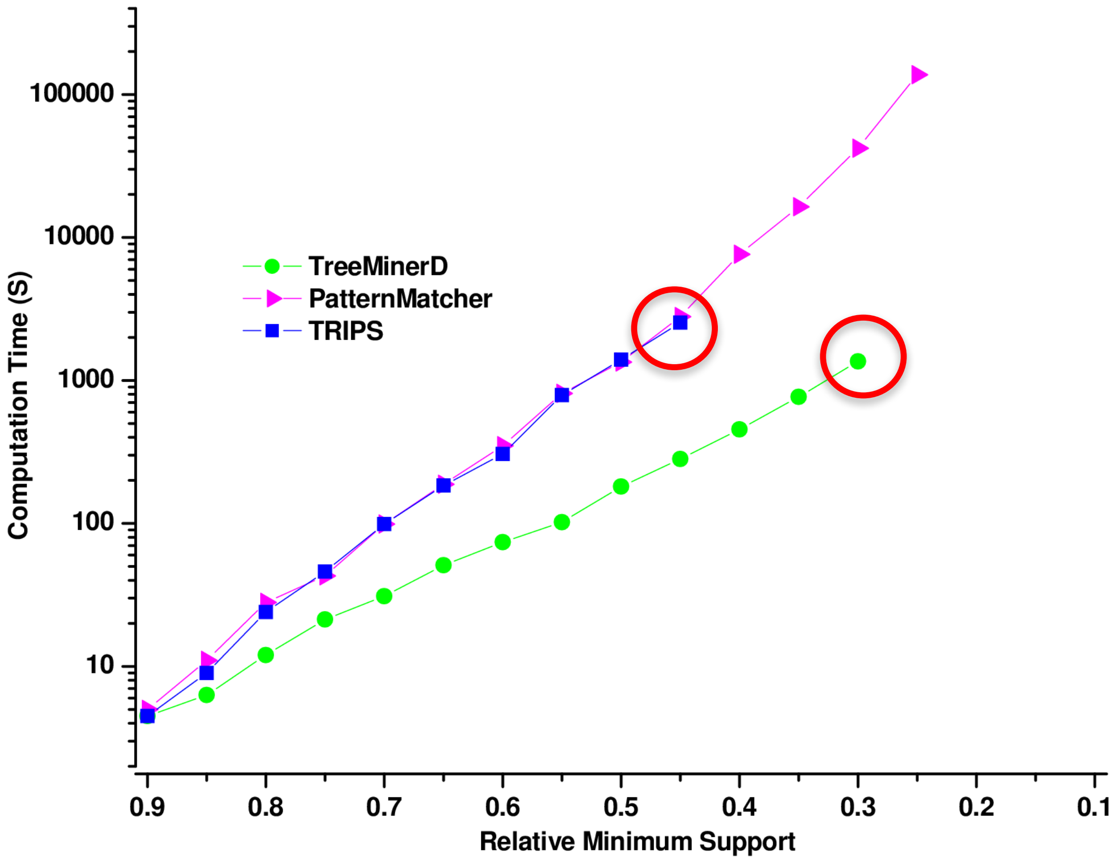
BFS and DFS refer to candidate generation approach, not tree traversal ☺

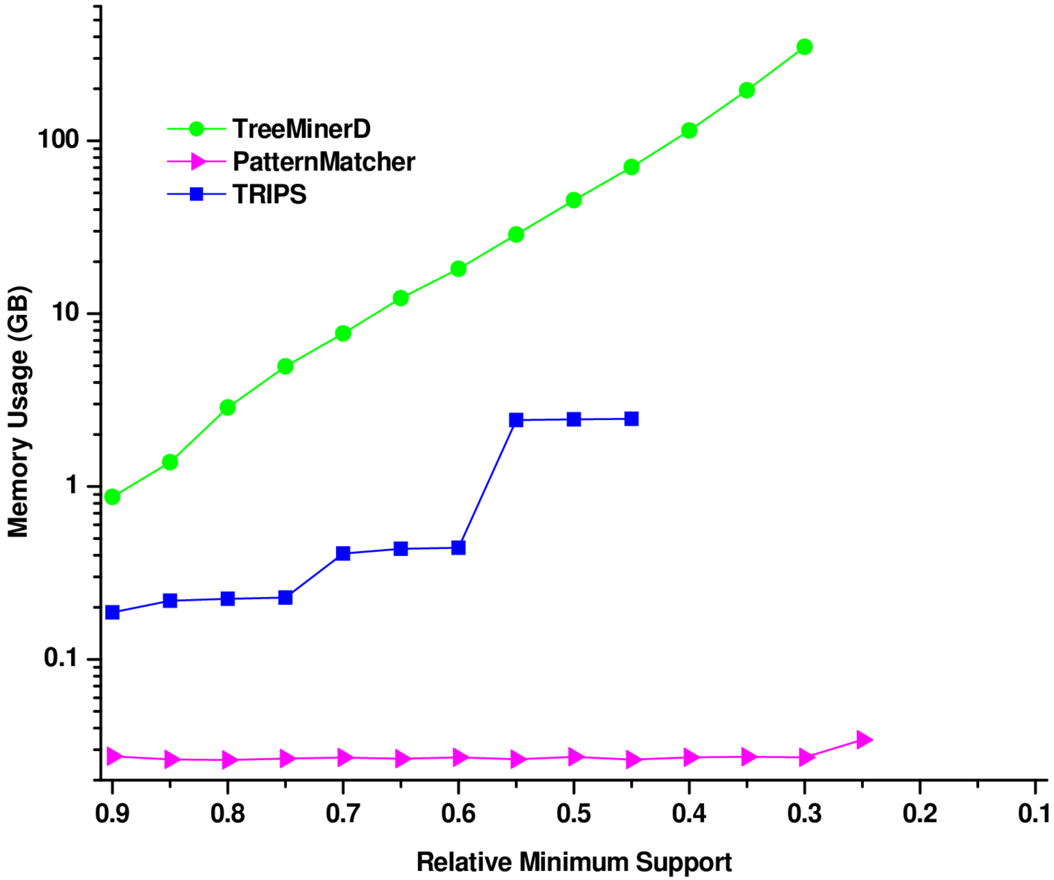
Issues with the Current FTM Solutions (2)





Issues with the Current FTM Solutions (2)





Would that be acceptable to achieve hundreds-X speedup at the expense of loosing a couple of percent accuracy?

Contribution of this research:

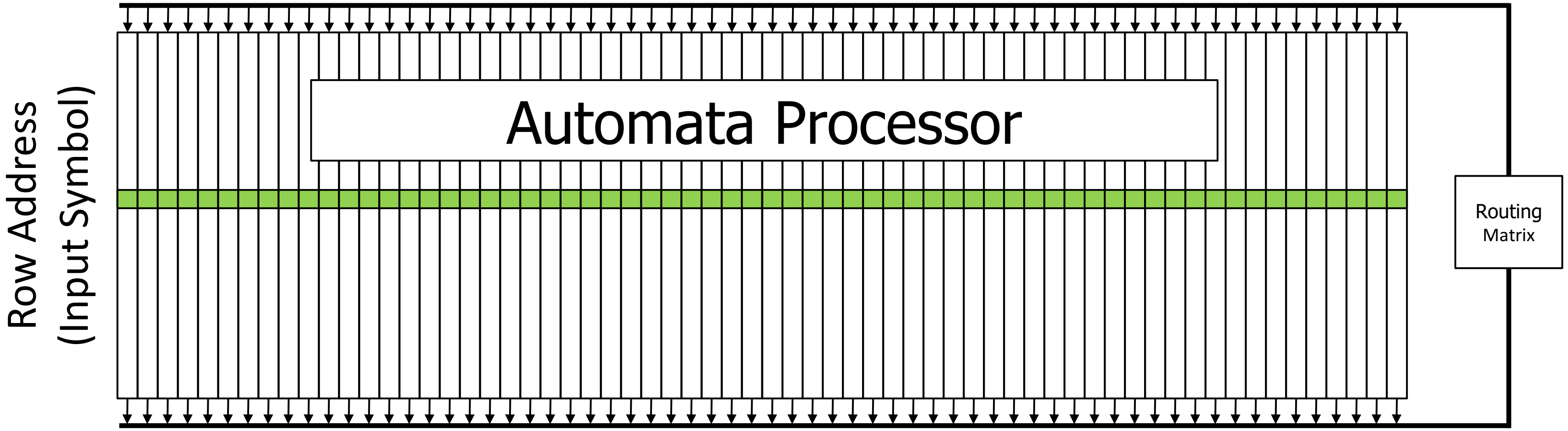
- Proposing a memory efficient and fast solution to the frequent subtree mining problem on the Automata Processor
- Achieving 350X and more speed up, when allowing 7.5% false positive subtrees

The Rest of This Talk

- Automata Processor
- FTM Challenges and Opportunities on the AP
- Experimental Evaluation
- Takeaways

The Automata Processor (1)

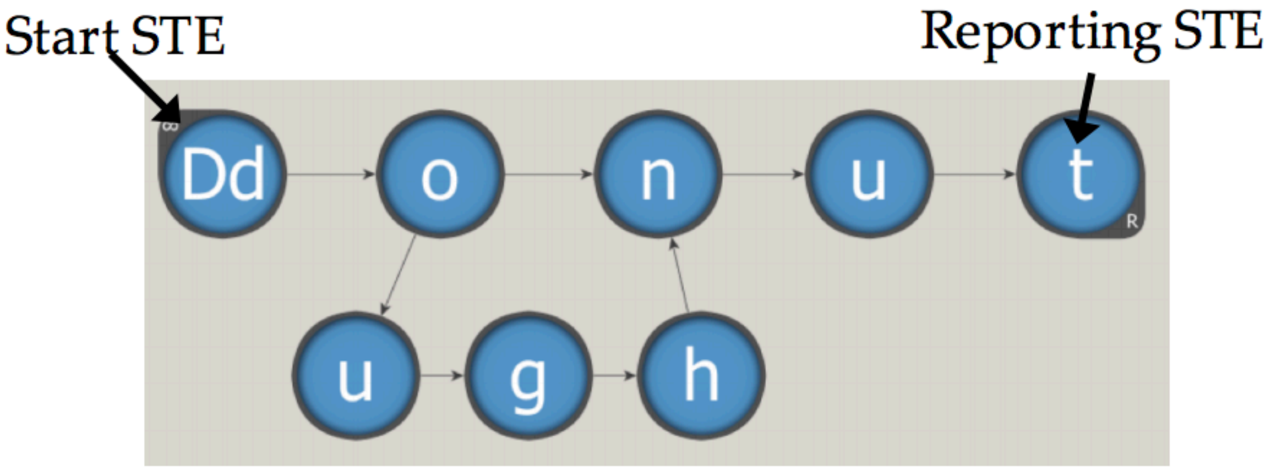
- The Micron Automata Processor (AP) is a reconfigurable non-von Neumann architecture, which implements non-deterministic finite automata (NFA) with Boolean logic gates and counters in hardware based on DRAM technology.



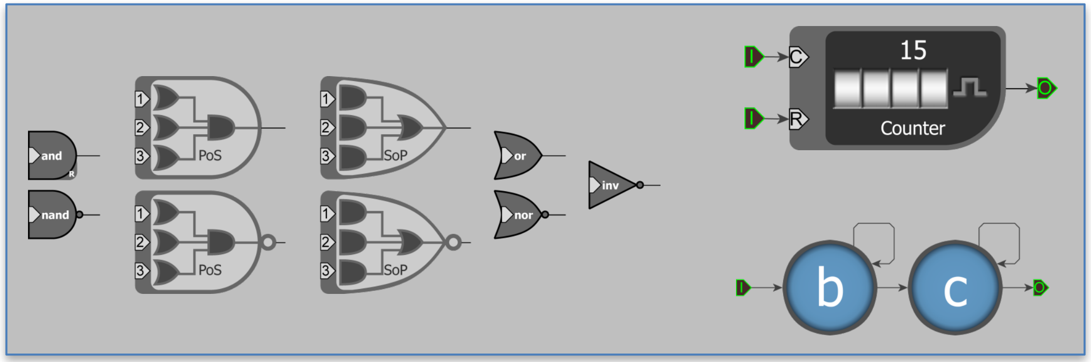
Row Access results in **49,152** match & route operations
(then Boolean AND with "active" bit-vector)

The Automata Processor (2)

- A massively parallel 'MISD' architecture
- 1 Gbps data processing
- Hardware resources on development board
  - State Transition Elements (STE): 1.5M
  - Reporting STEs: 200K
  - Counter Elements: 25K
  - Boolean Elements: 74K

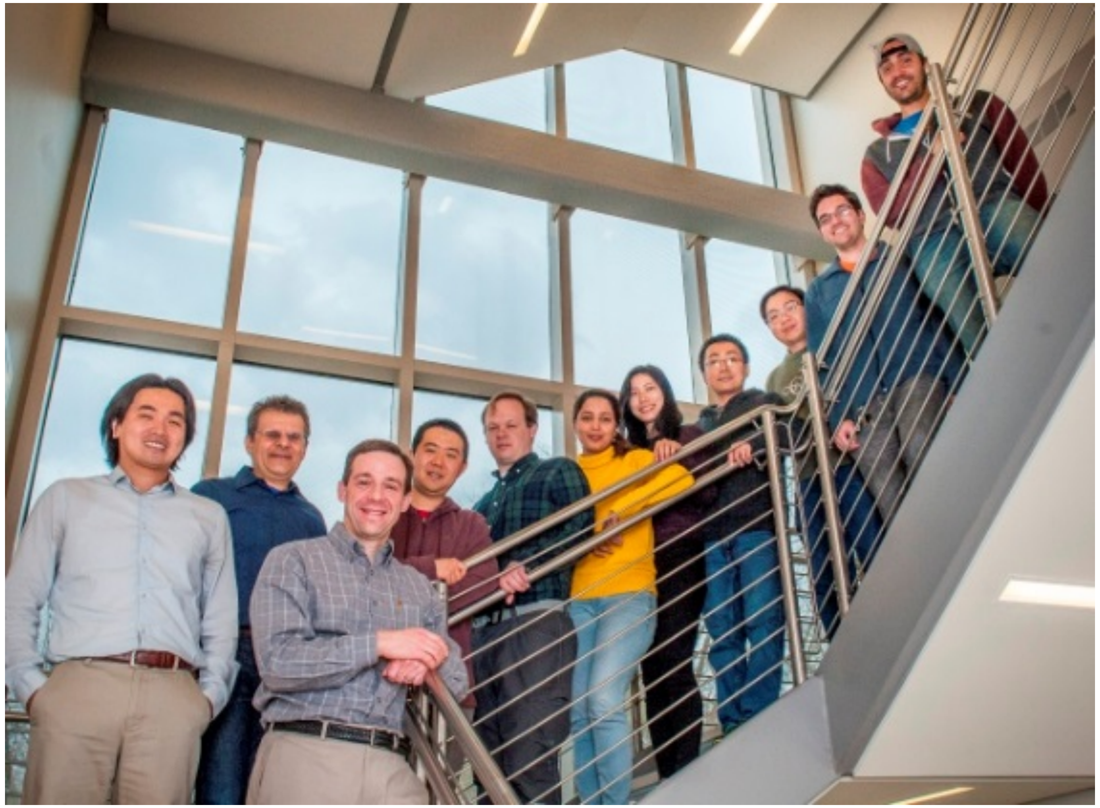


.*[Dd](o|ough)nut



Applications on the AP

- Data mining
  - Frequent itemset mining
  - Sequential pattern mining
- Machine learning
  - Random forest
  - Entity resolution
  - String/tree kernel
- Bioinformatics
  - Motif discovery
  - DNA alignment
- …



THE CENTER FOR AUTOMATA PROCESSING

Challenges: Exact FTM on the AP

The AP supports regular languages

Tree can be represented using context-free-grammar [Ivn07]

Challenges: Exact FTM on the AP

The AP supports regular languages

Tree can be represented using context-free-grammar [Ivn07]

The AP can not efficiently implement exact FTM

Challenges: Exact FTM on the AP

The AP supports regular languages

Tree can be represented using context-free-grammar [Ivn07]

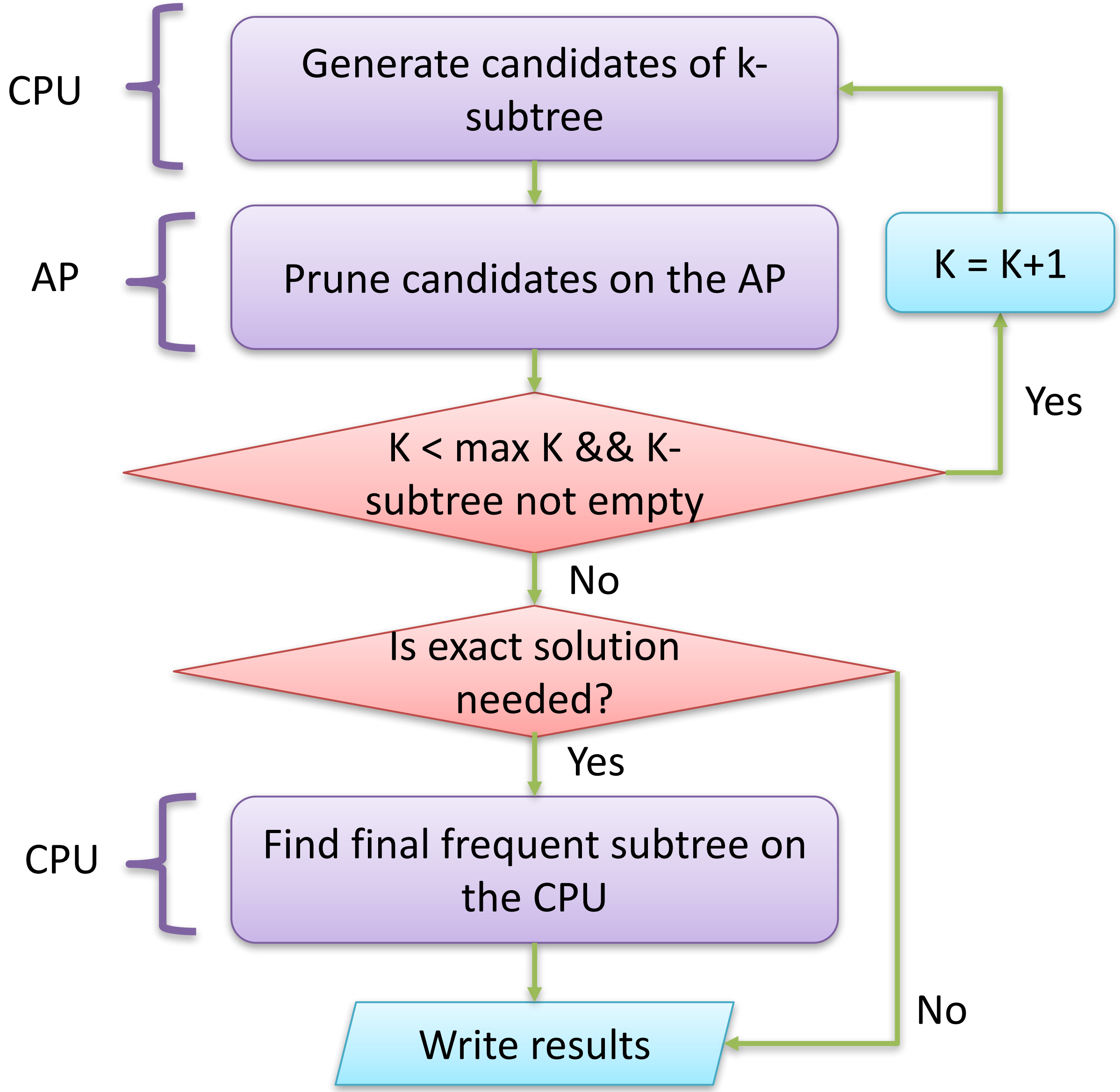
The AP can not efficiently implement exact FTM

Exact solutions (e.g., stack implementation on the AP) ⇒

- Inefficient
- Database dependent
- Impractical

Opportunities: Pruning

- Four-stage pruning strategy
  - Subset pruning
  - Intersection pruning
  - Downward pruning
  - Connectivity pruning
- Kernel properties
  - Complementary pruning
  - Avoiding false negatives



Background

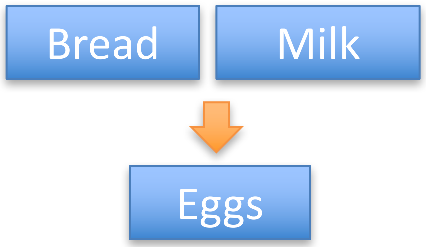
- Frequent itemset mining (FIM)

Trans.	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk,Diaper, Beer, Coke
4	Bread,Milk, Diaper, Beer, Coke
5	Bread, Milk, Coke, Diaper

sup({Diaper, Milk})= 3

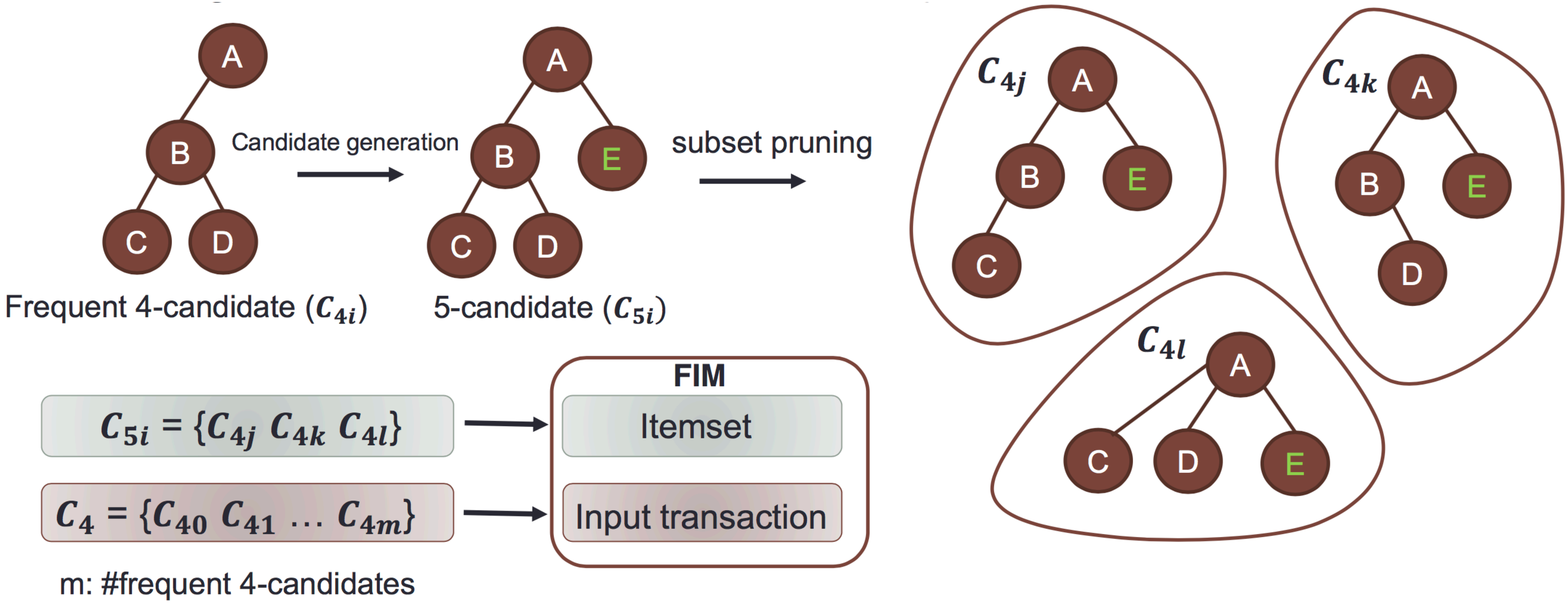
- Sequential pattern mining (SPM)

Trans.	Items
1	<{Bread, Milk}, {Coke}>
2	<{Bread, Milk, Diaper}{Beer, Eggs}{Diaper}>
3	<{Milk} {Diaper} {Beer, Coke}>
4	<{Bread, Milk, Diaper}{Beer, Diaper}{Beer, Coke, Eggs}>
5	<{Bread, Milk}{Coke}{Diaper}{Eggs}>



Kernel 1: Subset Pruning

- **Main goal**: checks downward closure property



* Wang, Ke, et al. "Association rule mining with the Micron Automata Processor." *Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2015.

Kernel 2: Intersection Pruning

- **Main goal**: checks if all the subsets of a candidate happens in the same tree

CPU implementation

	T1	T2	T3	T4
C_{4i}	1	1	0	1
C_{4j}	1	0	0	1
C_{4k}	1	1	1	0
C_{4l}	1	0	1	1

Minimum support = 50%

C_{5i} = 25%

AP implementation

T1	C_{4i}	C_{4j}	C_{4k}	C_{4l}
T2	C_{4i}	C_{4k}	…	…
T3	C_{4k}	C_{4l}	…	…
T4	C_{4i}	C_{4j}	C_{4l}	…

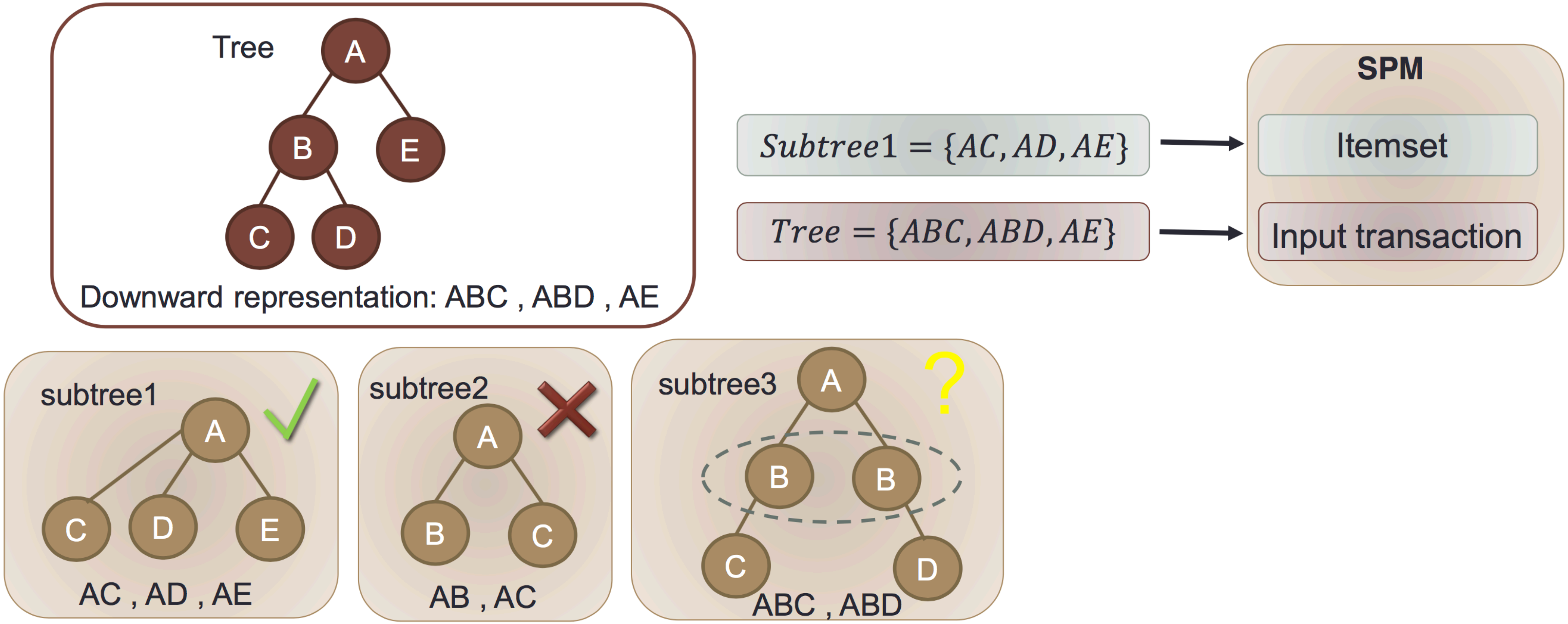
FIM

$itemset = C_{5i} = \{C_{4i}\ C_{4j}\ C_{4k}\ C_{4l}\}$

$$Input = \{C_{4i}\ C_{4j}\ C_{4k}\ C_{4l}\ \ldots,\ C_{4i}\ C_{4k}\ \ldots,\ C_{4k}\ C_{4l}\ \ldots,\ C_{4i}\ C_{4j} C_{4l}\ \ldots\}$$

Kernel 3: Downward Pruning

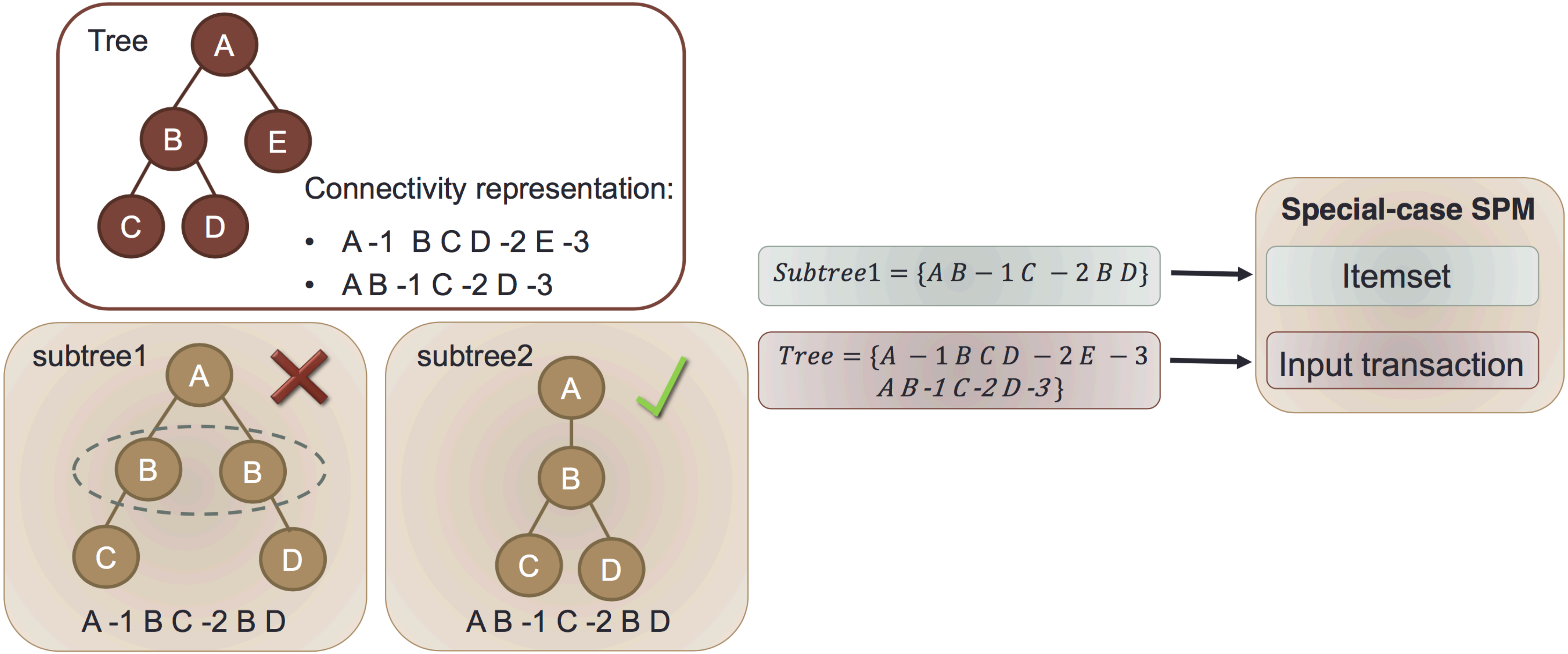
- **Main goal**: checks if ancestor descendant relationship is met



* Wang, Ke, **Elaheh Sadredini**, and Kevin Skadron. "Sequential pattern mining with the Micron automata processor." *Proceedings of the ACM International Conference on Computing Frontiers*. ACM, 2016.

Kernel 4: Connectivity Pruning

- **Main goal**: checks if sibling relationship is met



Framework

1. Making ARM and SPM automata for each kernel
2. Creating appropriate input stream
3. Configuring the automata for each kernel on the AP and streaming the corresponding input stream
4. Getting the potential frequent subtrees fro the AP output after applying the kernels

Performance Evaluation

- Platform
  - CPU: Intel(R) Core™ i7-5820k CPU @ 3.30GHz, Memory: 32GB
  - GPU: Tesla K80, Memory 24GB

- Dataset

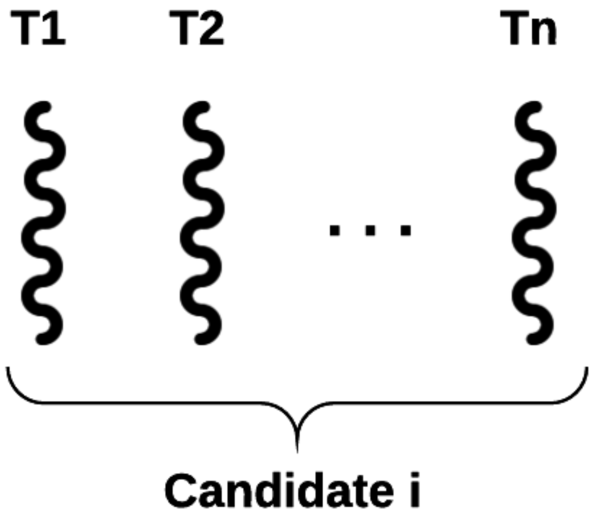
Name	#Trees	Ave_Node	SD_Node	#Items	Size (MB)
T1M	1,000,000	5.5	6.2	500	49.3
T2M	2,000,000	2.95	3.3	100	60.1
CSLOGS	59691	12.94	22.47	13361	6.3
TREEBANK	52581	68.03	32.46	1387266	27.3

Ave_Node = Average number of nodes per tree.
SD_Node = Standard deviation of number of nodes per tree.

- Apples-to-apples comparison

GPU Implementation

- BFS Approach, because:
  - Not be bound by the finite GPU global memory
  - Exposes many ready-to-process candidates and provide parallelism
- FTM-GPU
  - Candidate generation on the CPU
  - Subset pruning on the CPU
  - Enumeration on the GPU
    - Trees in shared memory
    - Candidate in constant memory
- Sorting the input trees
  - Decrease divergence



Algorithmic & Architectural Contributions

AP kernel over CPU kernel speedup:

Subset = up to163X

Intersection: up to 19X

Downward: up to 3144X

Connectivity: up to 2635X

Algorithmic & Architectural Contributions

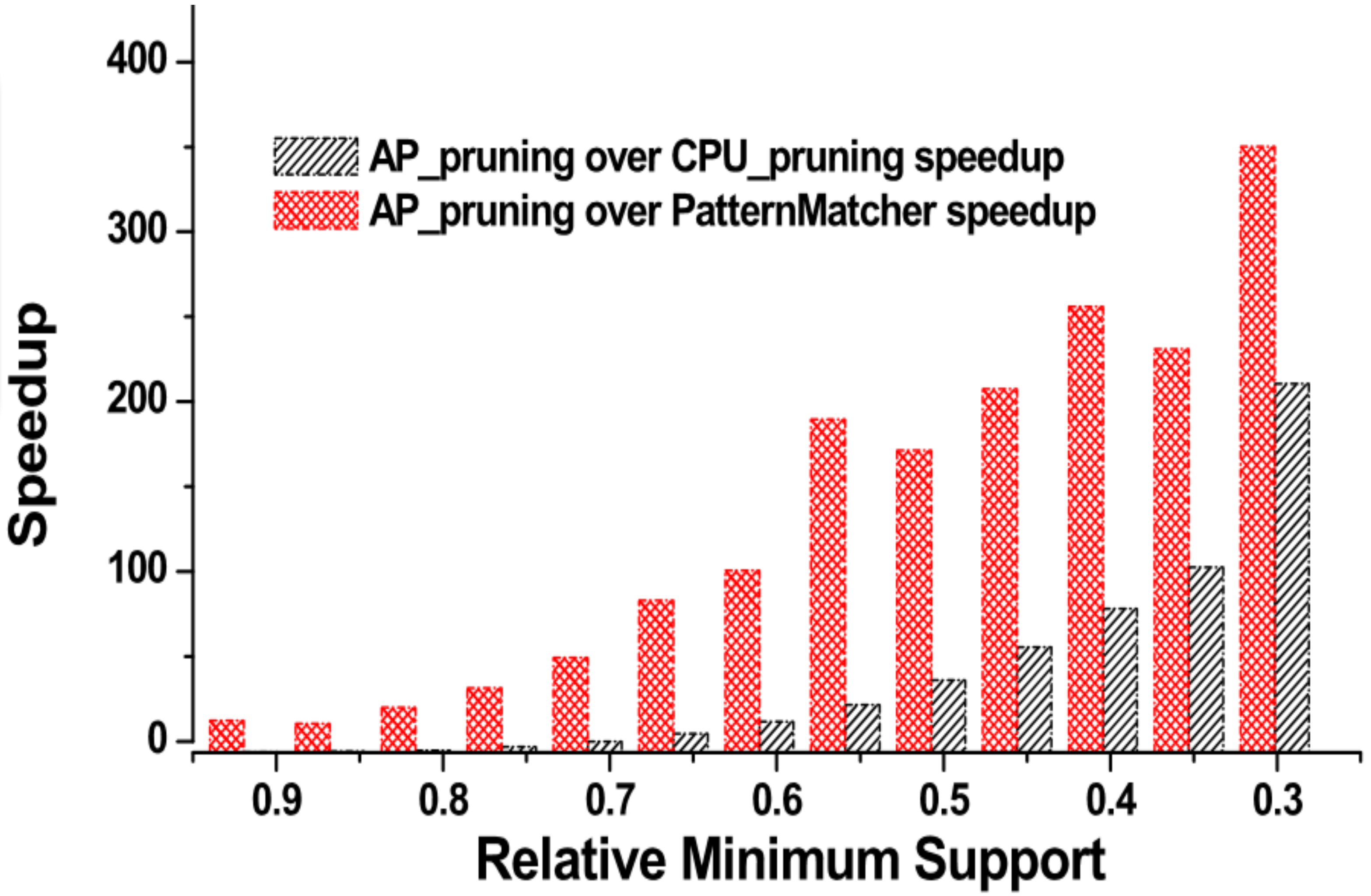
AP kernel over CPU kernel speedup:

Subset = up to163X

Intersection: up to 19X

Downward: up to 3144X

Connectivity: up to 2635X



Algorithmic & Architectural Contributions

AP kernel over CPU kernel speedup:

Subset = up to163X

Intersection: up to 19X

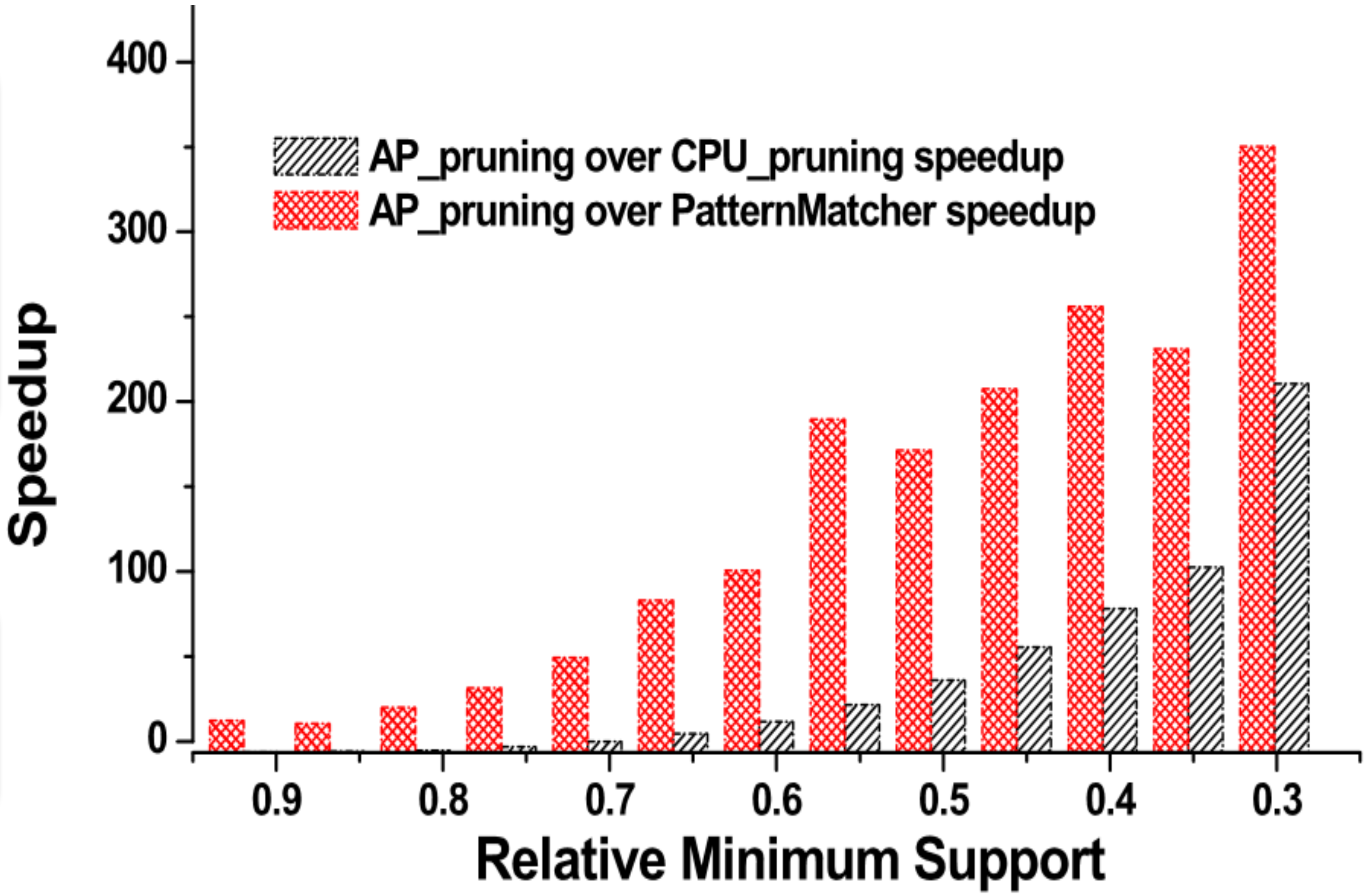
Downward: up to 3144X

Connectivity: up to 2635X

Red bar over black bar: up to 1.6X

Black bars: up to 215X

Red bars: up to 353X



Pruning Efficiency

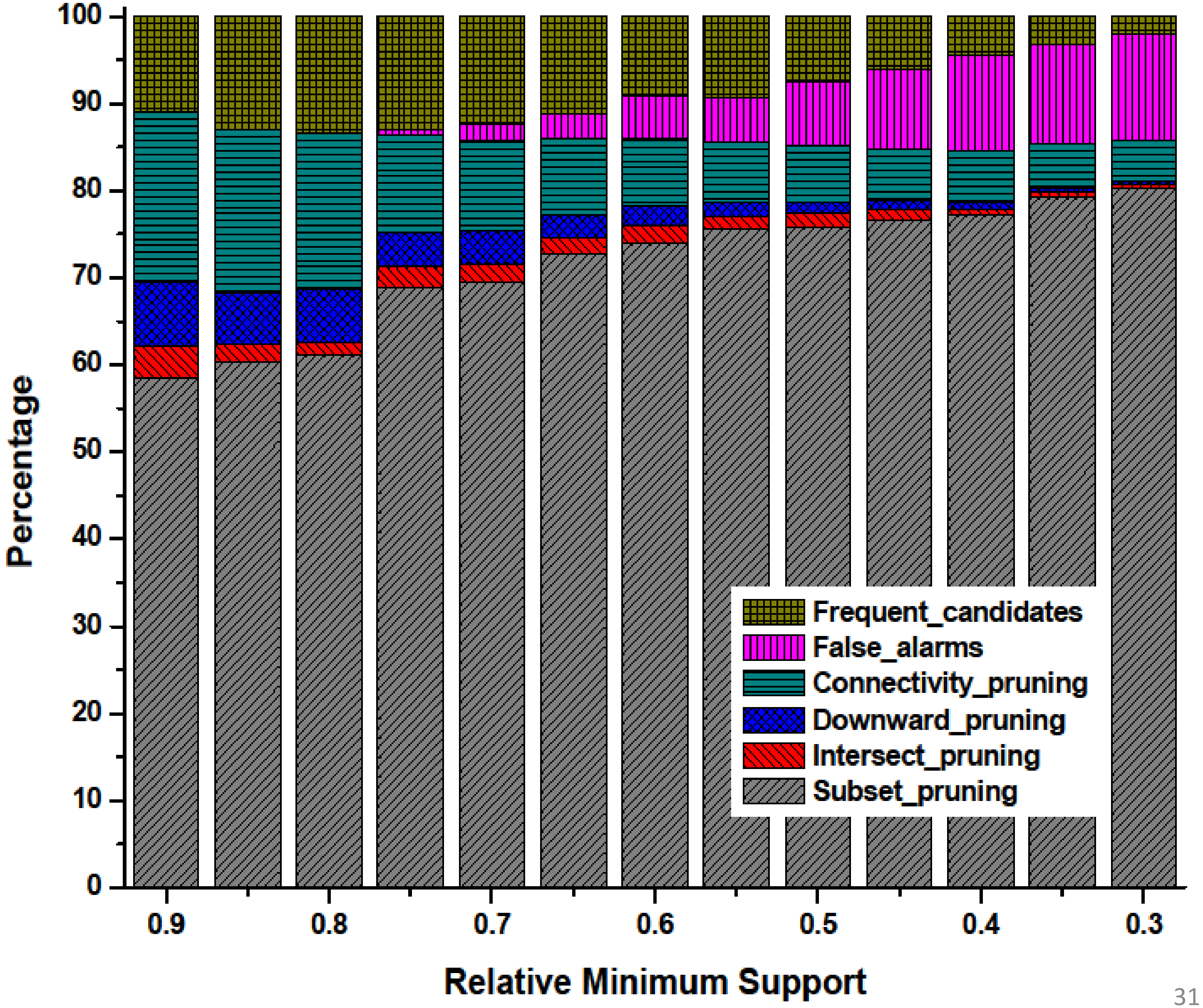
Kernel effectiveness:

Subset = 80%

Intersection: 0.5%

Downward: 3.5%

Connectivity: 4.8%



Pruning Efficiency

Kernel effectiveness:

Subset = 80%

Intersection: 0.5%

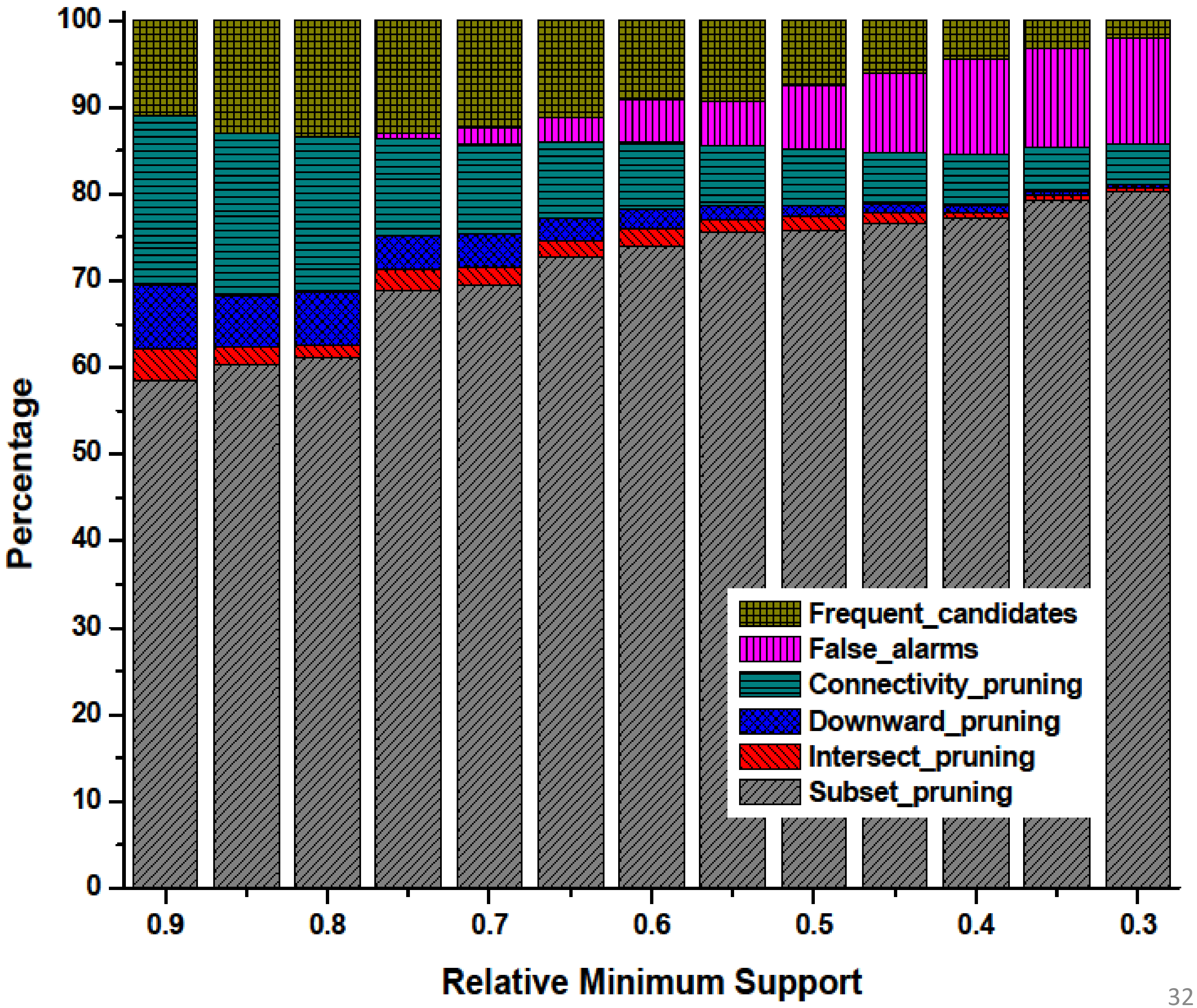
Downward: 3.5%

Connectivity: 4.8%

Removing intersection kernel:

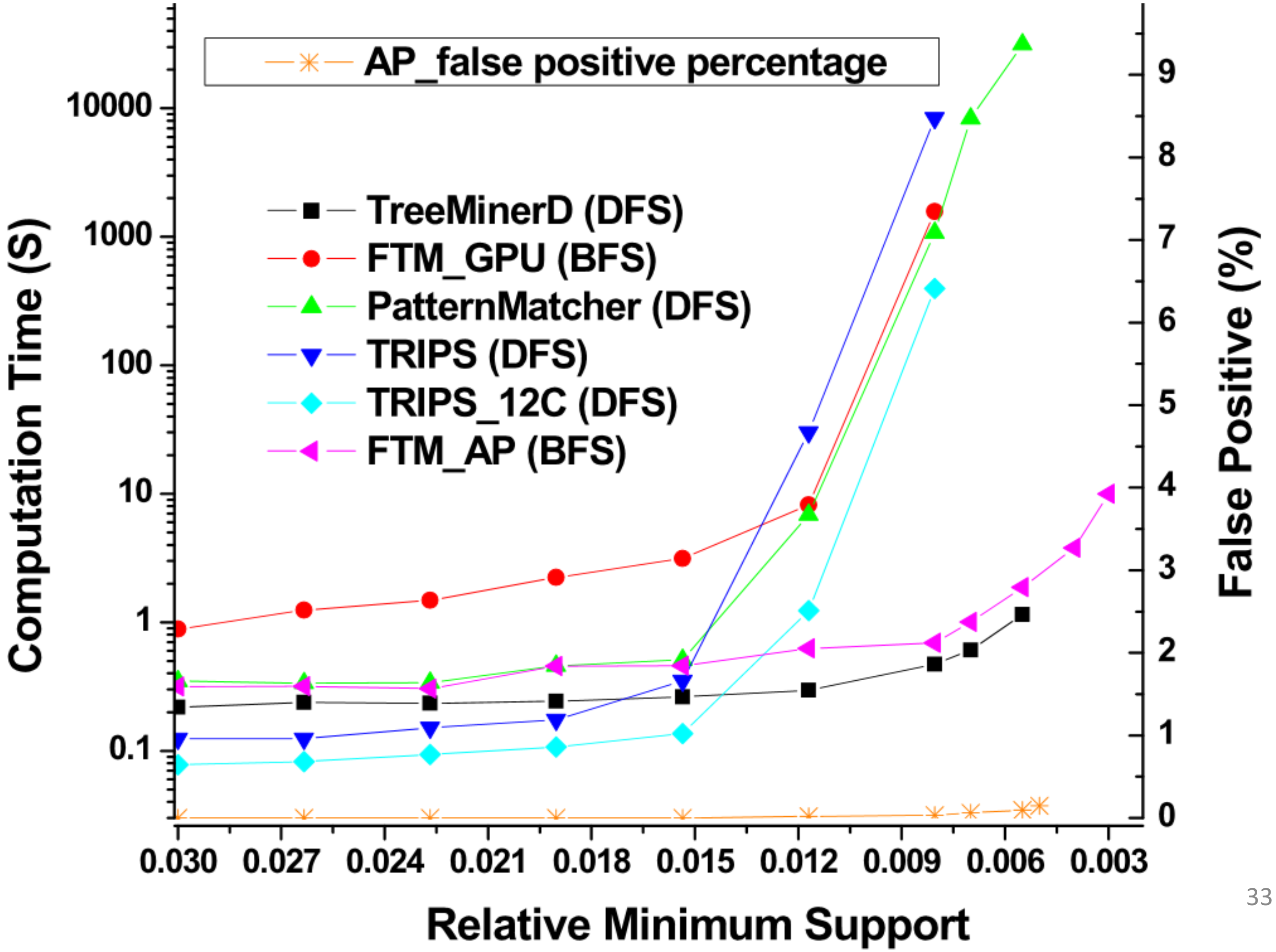
AP over PatternMatcher: 353X → 2190X

Accuracy: 86% → 83%



FTM-AP vs Other FTM Algorithms

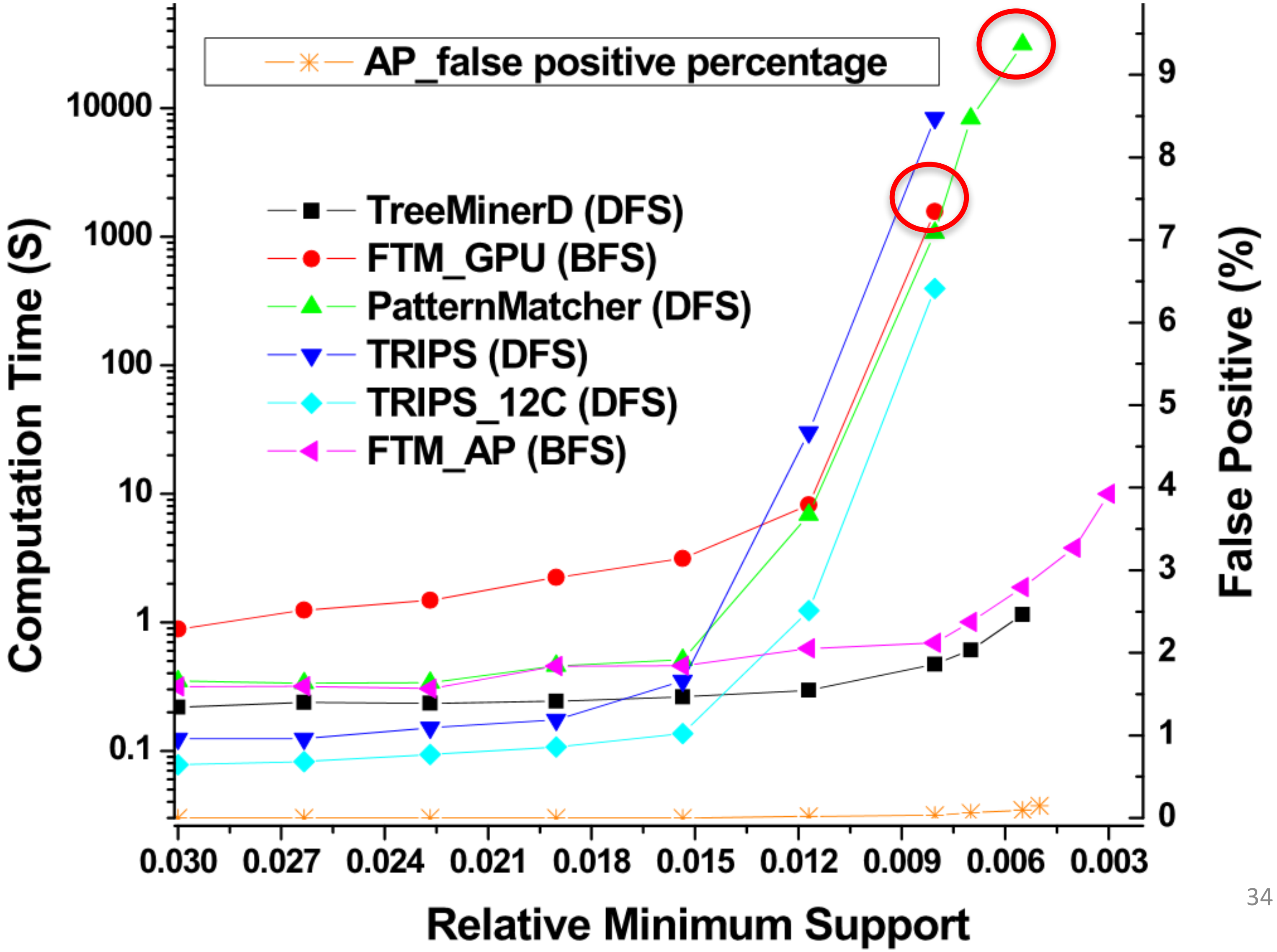
Trade-off between *speed* and *accuracy* of the AP solution vs the existing FTM implementation



Dataset: CSLOGS

FTM-AP vs Other FTM Algorithms

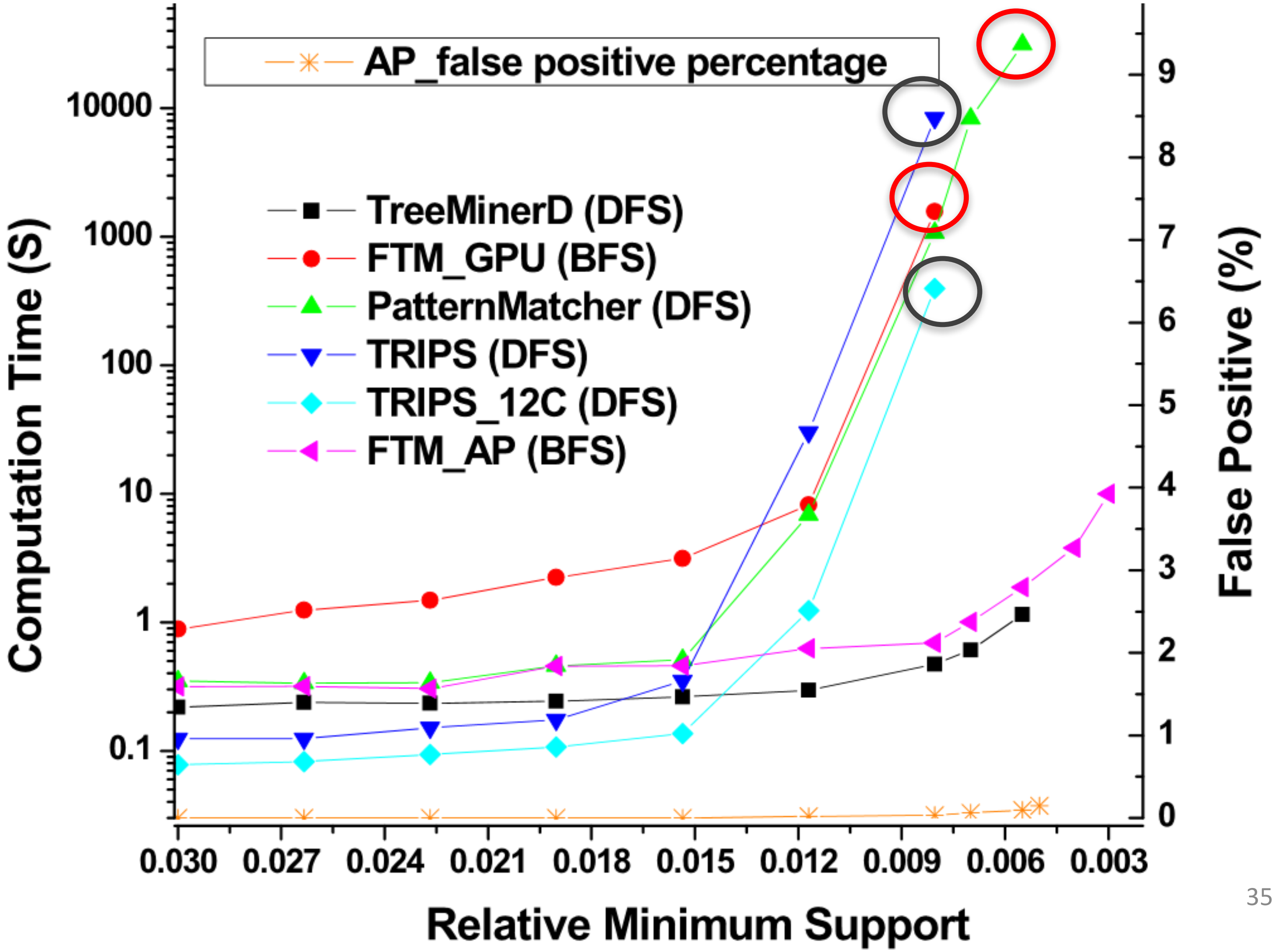
Trade-off between *speed* and *accuracy* of the AP solution vs the existing FTM implementation



Dataset: CSLOGS

FTM-AP vs Other FTM Algorithms

Trade-off between *speed* and *accuracy* of the AP solution vs the existing FTM implementation



Dataset: CSLOGS

FTM-AP vs Other FTM Algorithms

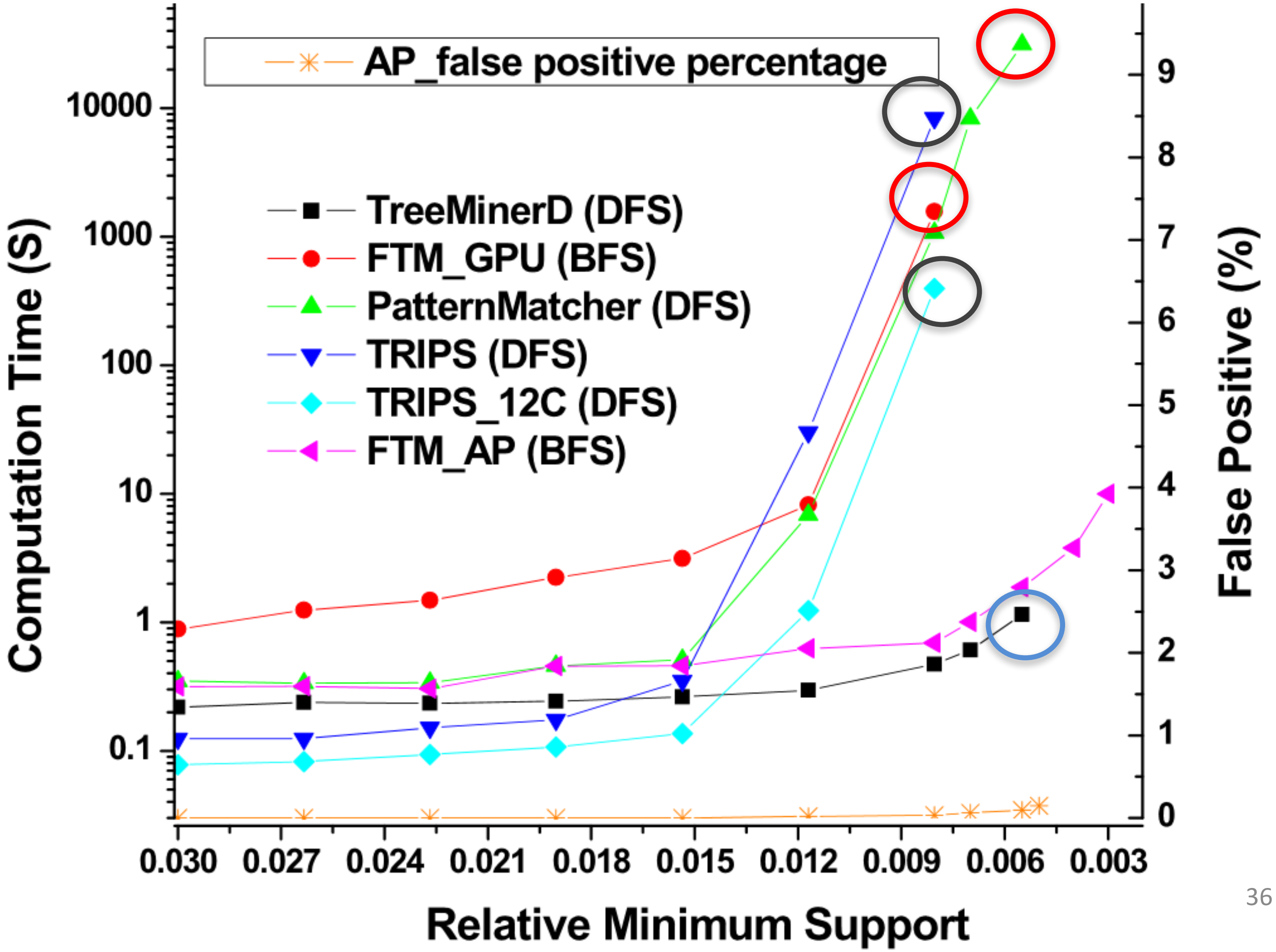
Trade-off between *speed* and *accuracy* of the AP solution vs the existing FTM implementation

Speedup

$$\frac{FTM\_AP}{PatternMatcher} = \text{up to 353X}$$

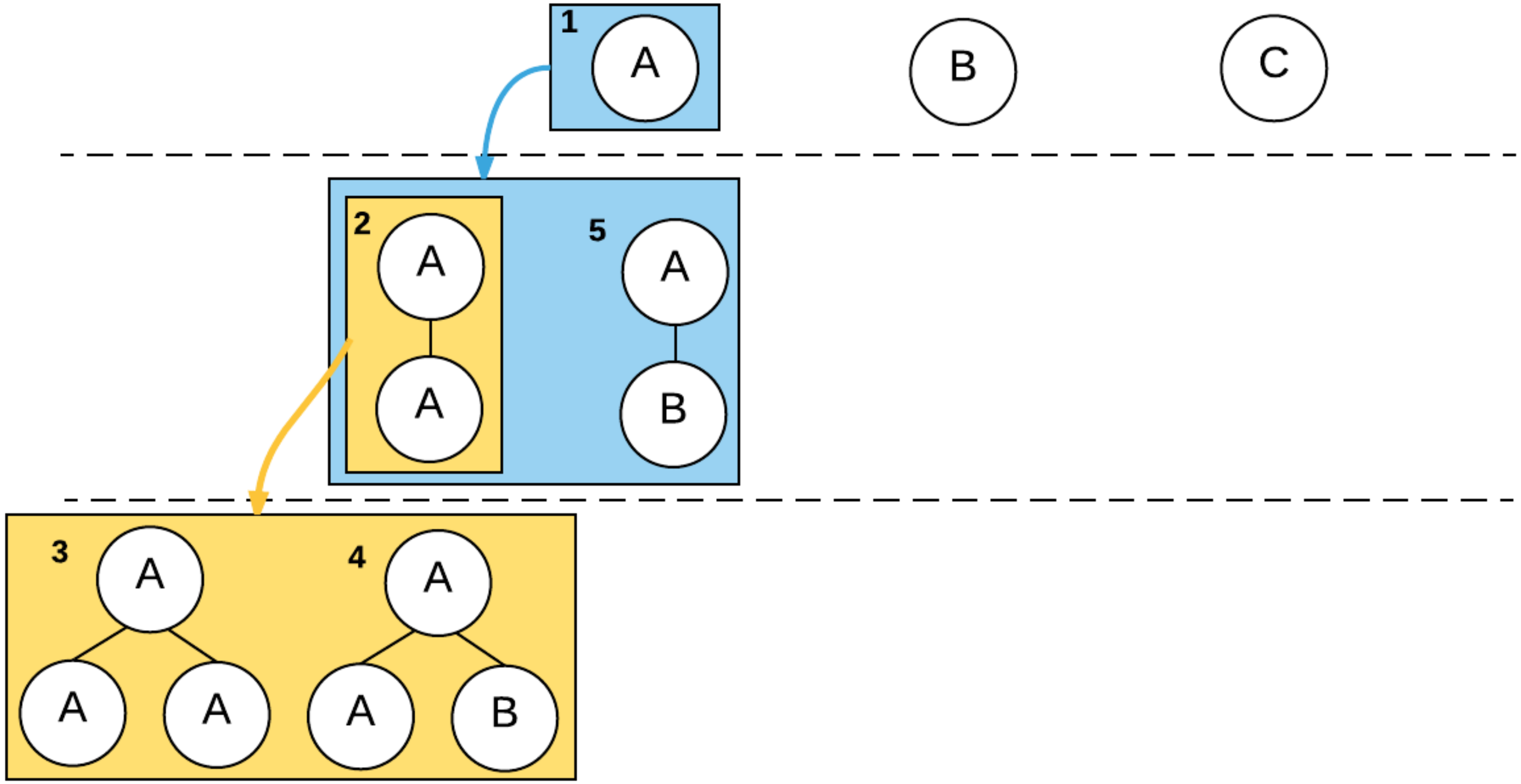
Memory usage

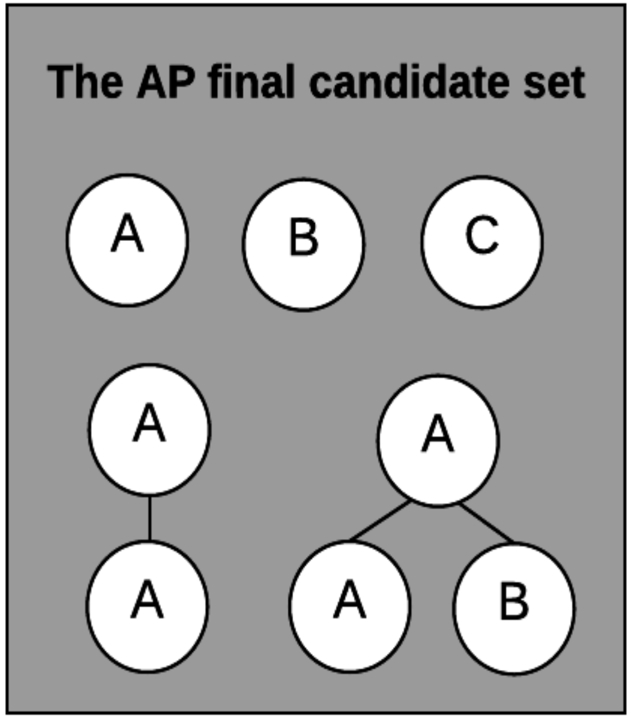
$$\frac{TreeMinerD}{FTM\_AP} = \text{up to 5000X}$$



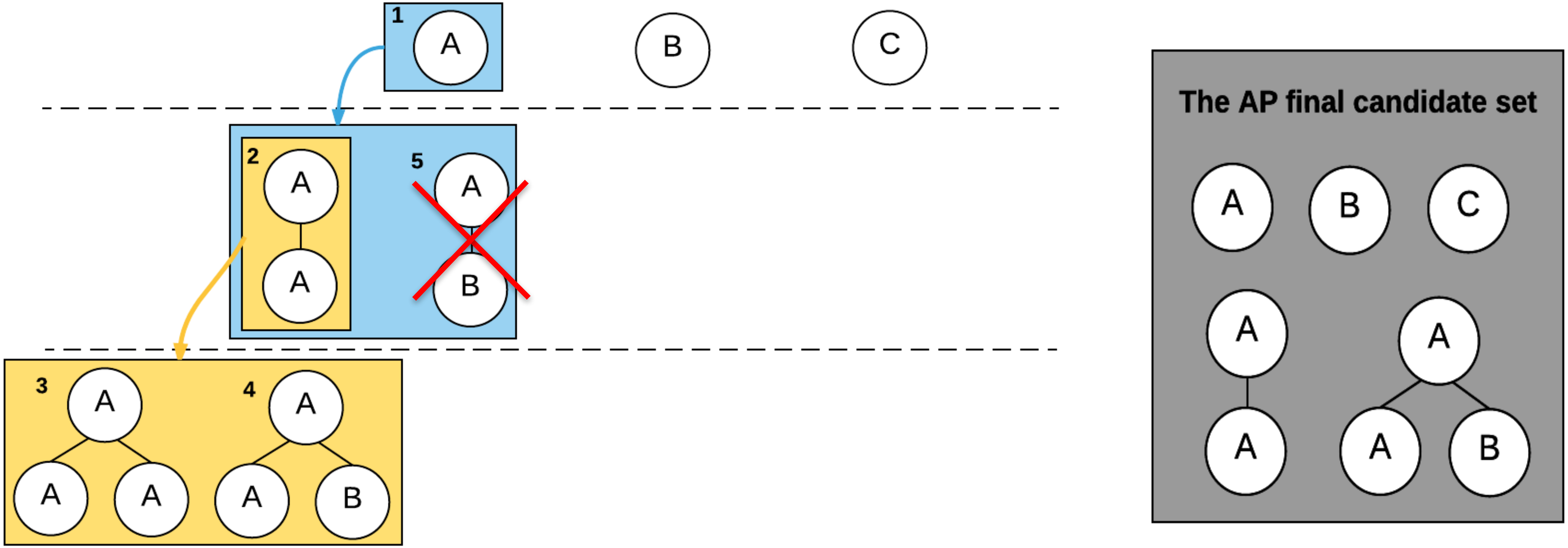
Dataset: CSLOGS

Exact Solution: AP + TreeMinerD

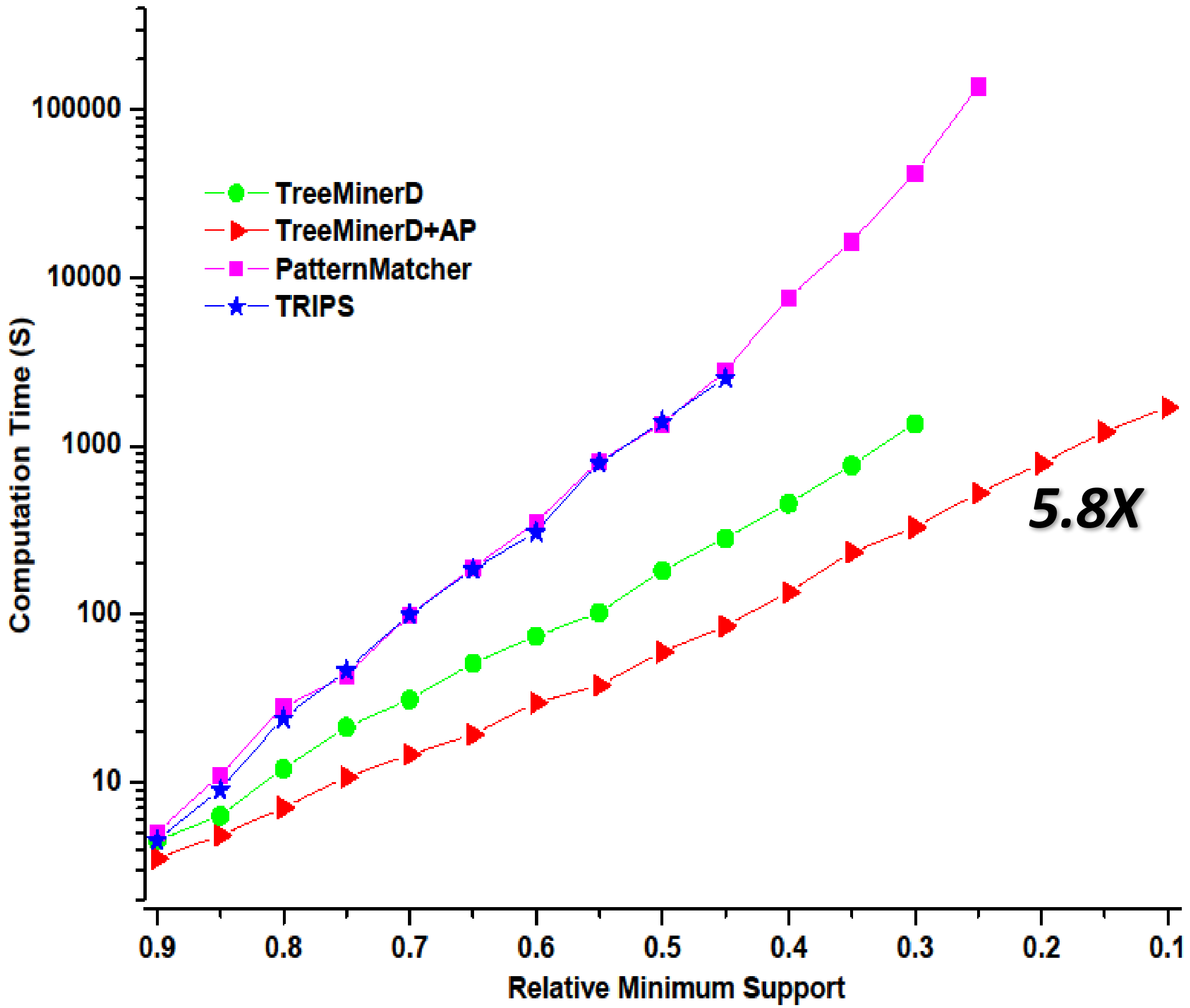


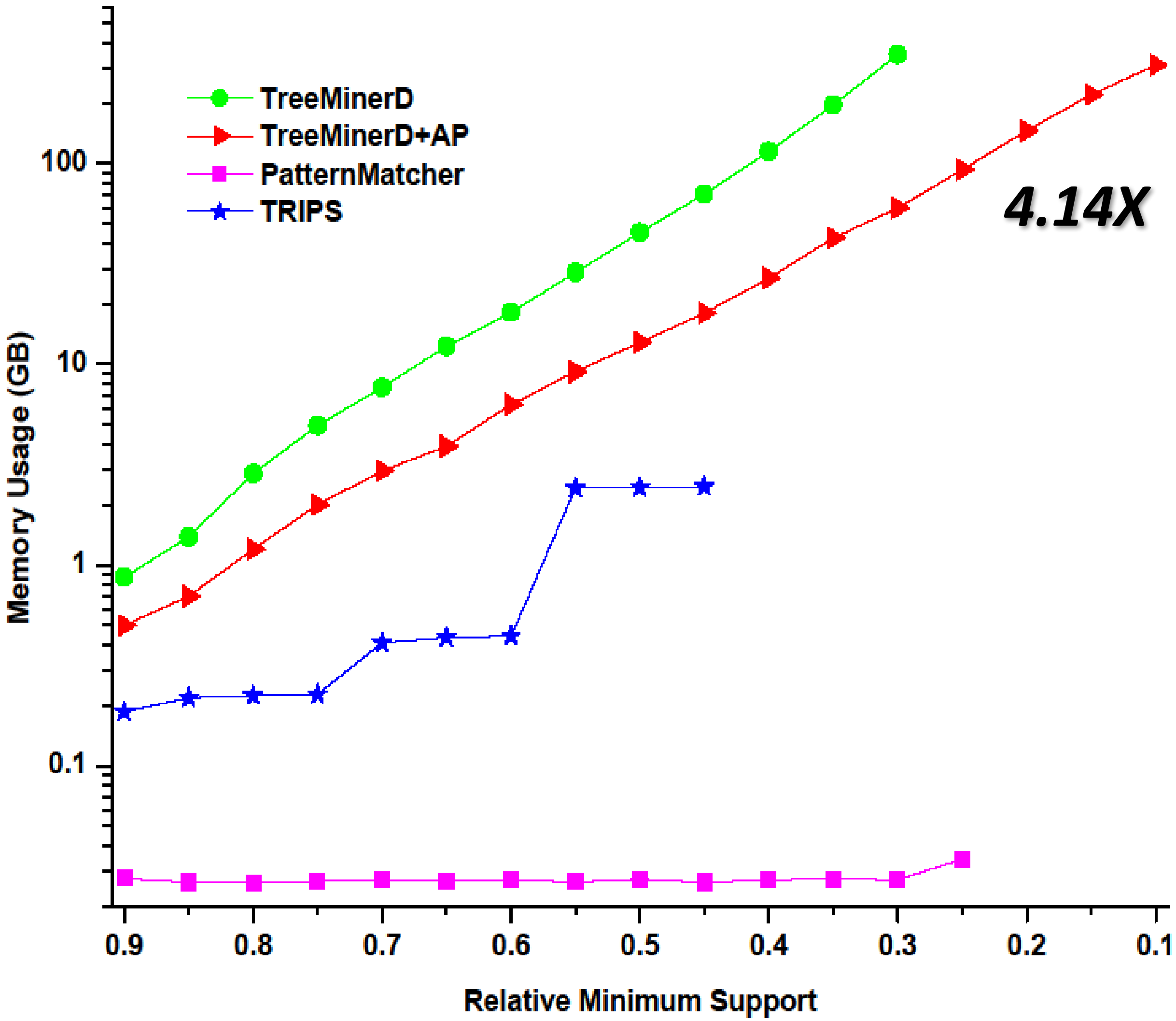


Exact Solution: AP + TreeMinerD



Performance Evaluation: Exact Solution: AP + TreeMinerD





Intel Xeon CPU, 2.30GHz, 512 GB memory, 2.133GHz

Dataset: TREEBANK

Summary

- Propose a multi-stage pruning framework on the AP
  - The first work to use the AP as a pruning media
  - Novel pruning kernels
  - A better scalability and stable behavior
  - Propose an exact solution

Takeaways

- Rethinking the algorithm when having a new hardware architecture
- Applies to spatial automata computing architecture such as FPGAs
- This approach can be adopted for other complex pattern mining problems
- Provides some insight for the architectural changes

References

- Ke Wang, Elaheh Sadredini, and Kevin Skadron. "Hierarchical Pattern Mining with the Micron Automata Processor.“ International Journal of Parallel Programming (IJPP), 2017.
- Ke Wang, Elaheh Sadredini, and Kevin Skadron. "Sequential Pattern Mining with the Micron Automata Processor." ACM International Conference on Computing Frontiers, May 2016
- J. Wadden, V. Dang, N. Brunelle, T. Tracy II, D. Guo, E. Sadredini, K. Wang, C. Bo, G. Robins, M. Stan, K. Skadron. "ANMLZoo: A Benchmark Suite for Exploring Bottlenecks in Automata Processing Engines and Architectures." IEEE International Symposium on Workload Characterization (IISWC), October 2016.
- Elaheh Sadredini, Reza Rahimi, Ke Wang, and Kevin Skadron. "Frequent Subtree Mining on the Automata Processor: Opportunities and Challenges." ACM International Conference on Supercomputing (ICS), Chicago, June 2017
- Shirish Tatikonda, Srinivasan Parthasarathy, and Tahsin Kurc. ”TRIPS and TIDES: new algorithms for tree mining.” Proceedings of the 15th ACM International Conference on Information and Knowledge Management. ACM, 2006.
- Ke Wang, Elaheh Sadredini, and Kevin Skadron. ”Sequential pattern mining with the Micron automata processor.” Proceedings of the ACM International Conference on Computing Frontiers. ACM, 2016.
- Mohammed Javeed Zaki. ”Efficiently mining frequent trees in a forest: Algorithms and applications.” IEEE Transactions on Knowledge and Data Engineering 17.8 (2005): 1021-1035.
- Yun Chi, et al. ”Frequent subtree mining an overview.” Fundamenta Informaticae 21: 1001-1038, 2011. Paul Dlugosch, et al. ”An efficient and scalable semiconductor architecture for parallel automata processing.” IEEE Transactions on Parallel and Distributed Systems, 25.12:3088-3098, 2014.
- Renta Ivncsy, and Istvn Vajk. ”Automata Theory Approach for Solving Frequent Pattern Discovery Problems.” World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering 1(8): 2556-2561, 2007.
- Fedja Hadzic, Henry Tan, and Tharam S. Dillon. Mining of data with complex structures. Vol. 333. Springer-Verlag, 2011.
- John L. Hennessy, and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.
- Ke Wang, et al. ”Association rule mining with the Micron Automata Processor.” Parallel and Distributed Processing Symposium (IPDPS), IEEE International, 2015.
- Wang, Ke, Kevin Angstadt, Chunkun Bo, Nathan Brunelle, Elaheh Sadredini, Tommy Tracy II, Jack Wadden, Mircea Stan, and Kevin Skadron. "An overview of micron's automata processor." In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, p. 14. ACM, 2016.

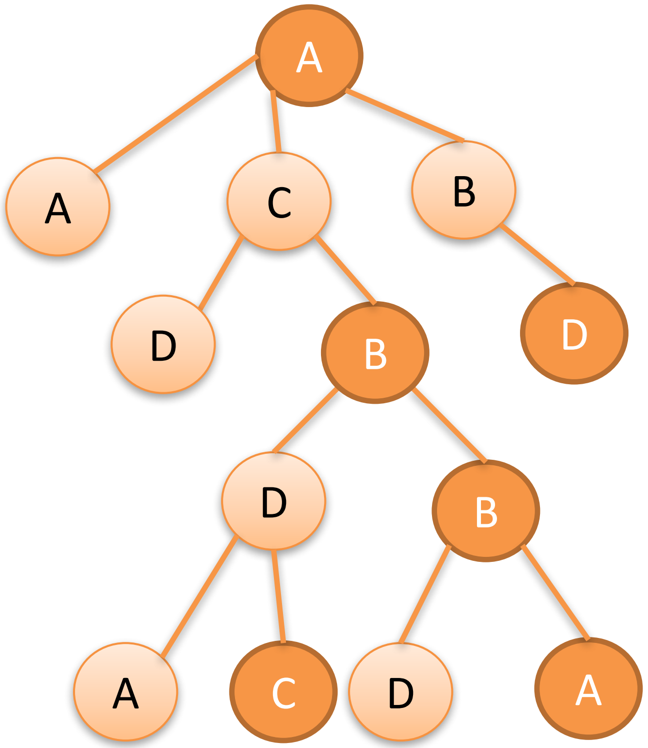
Thank you ☺

Questions?

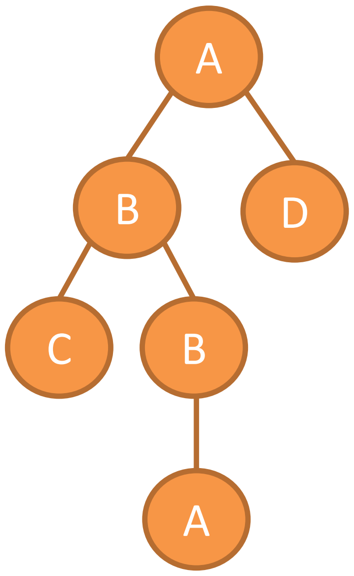
Backup Slides

PDA-based Subtree Mining: An Example

Tree



Embedded subtree

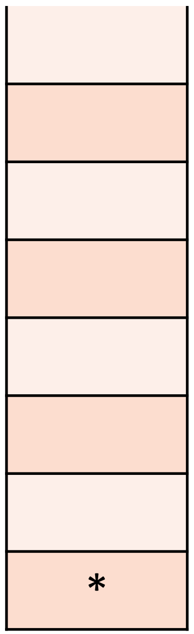


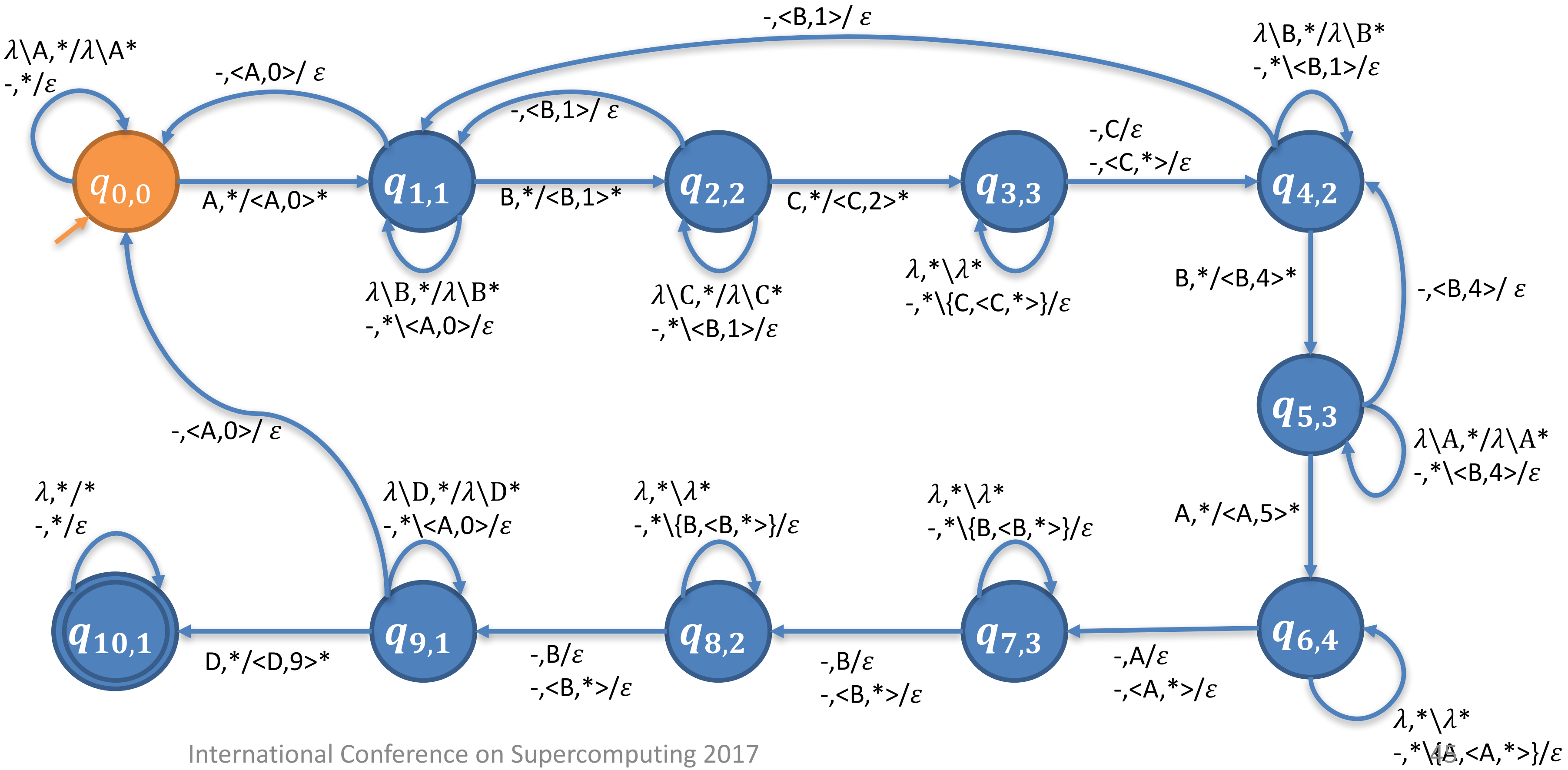
String Encoding:
A A – C D – B D A – C – – B D – A – – – – B D

String Encoding:
A B C – B A – – – D

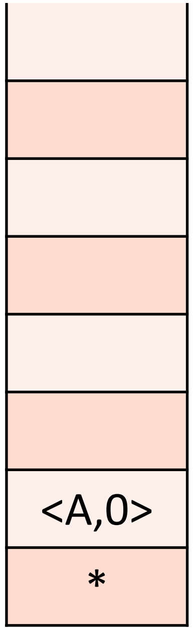
Input tree: A A – C D – B D A – C – – B D – A – – – – – B D

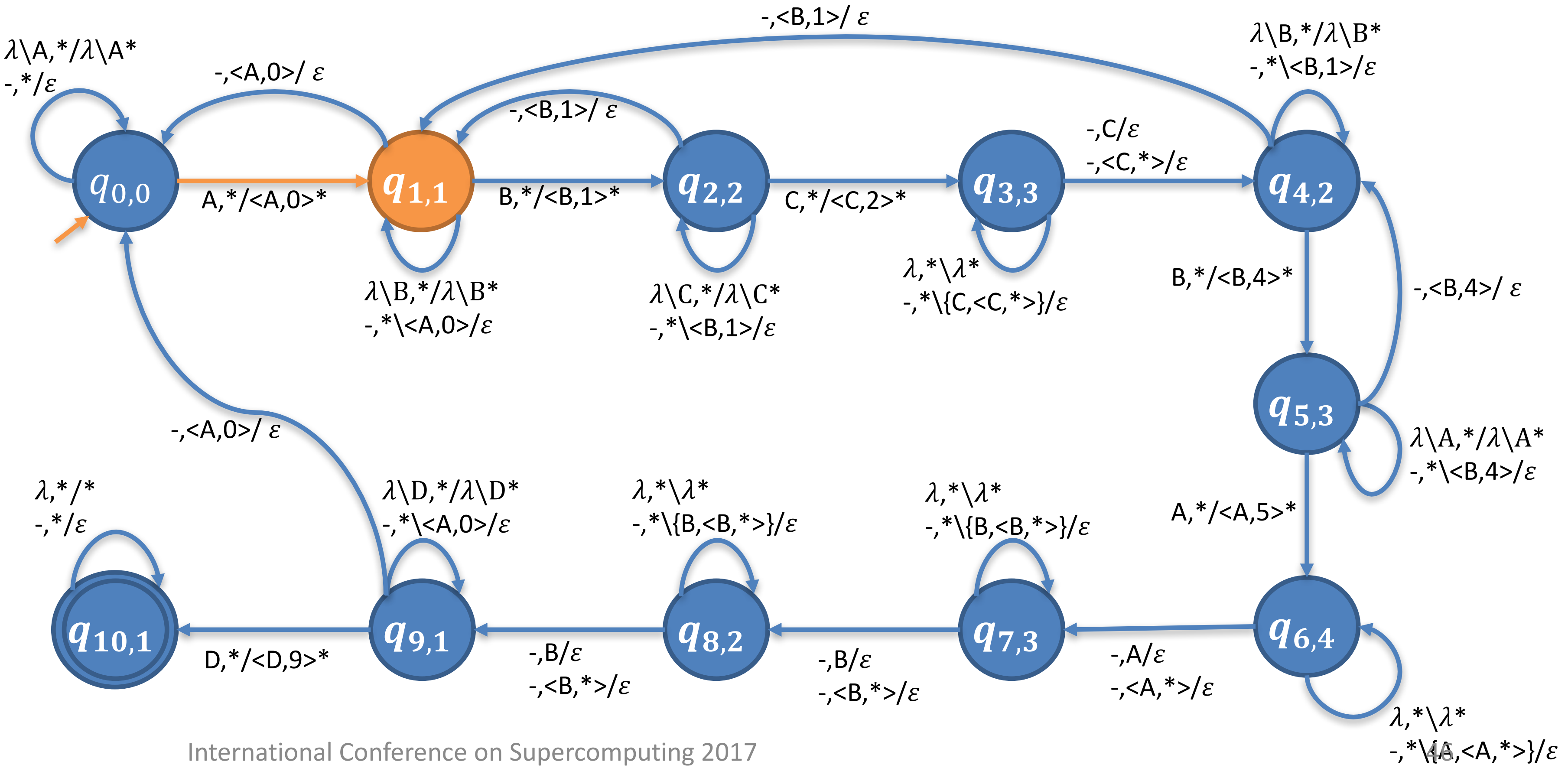
Deterministic PDA



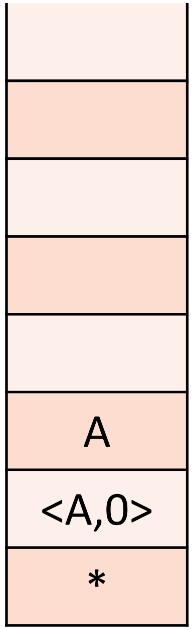


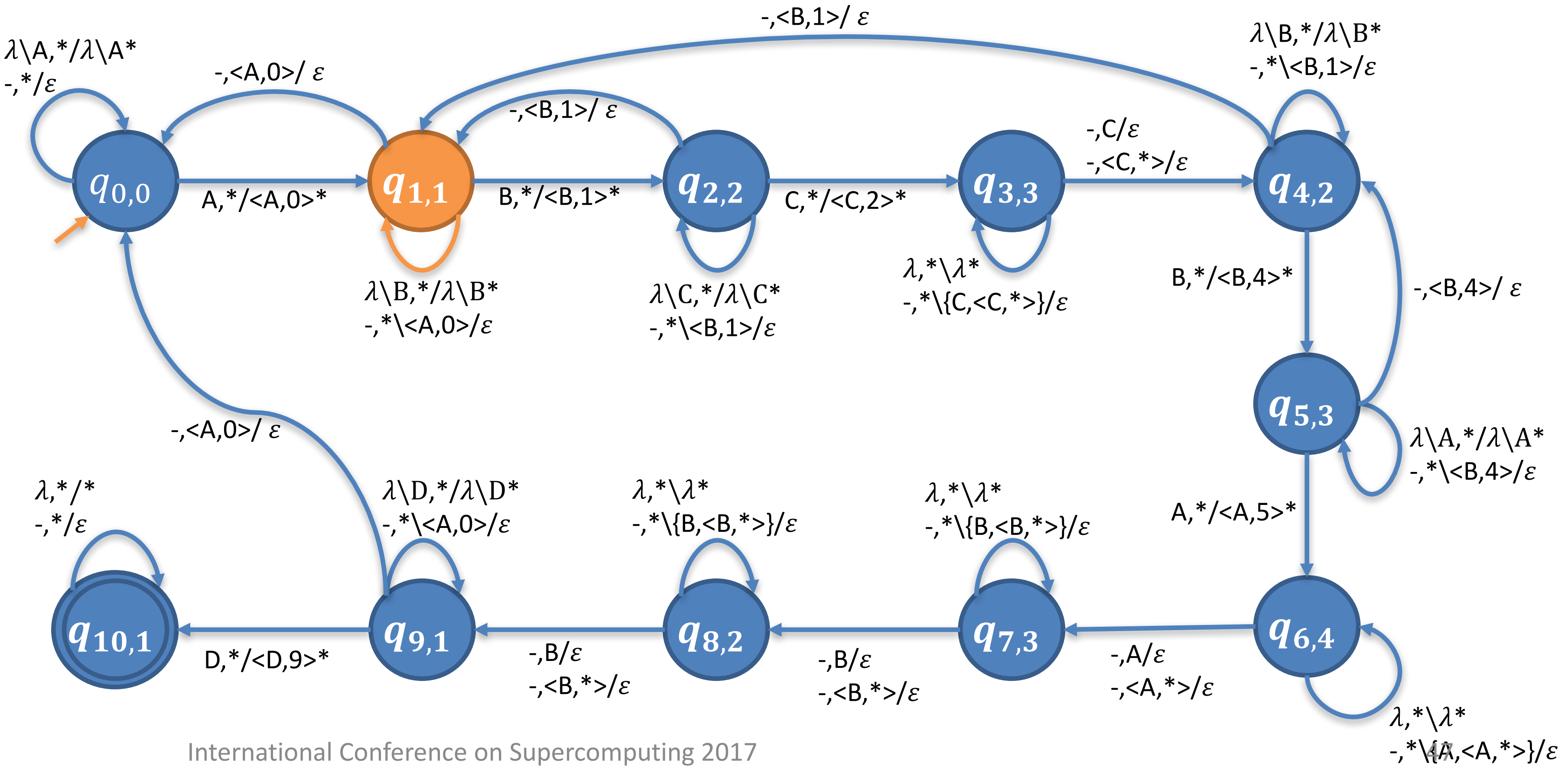
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



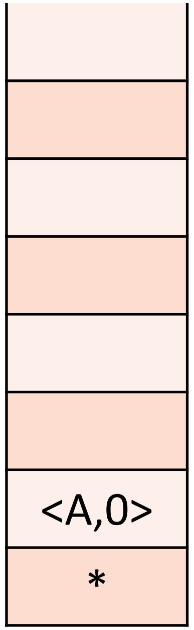


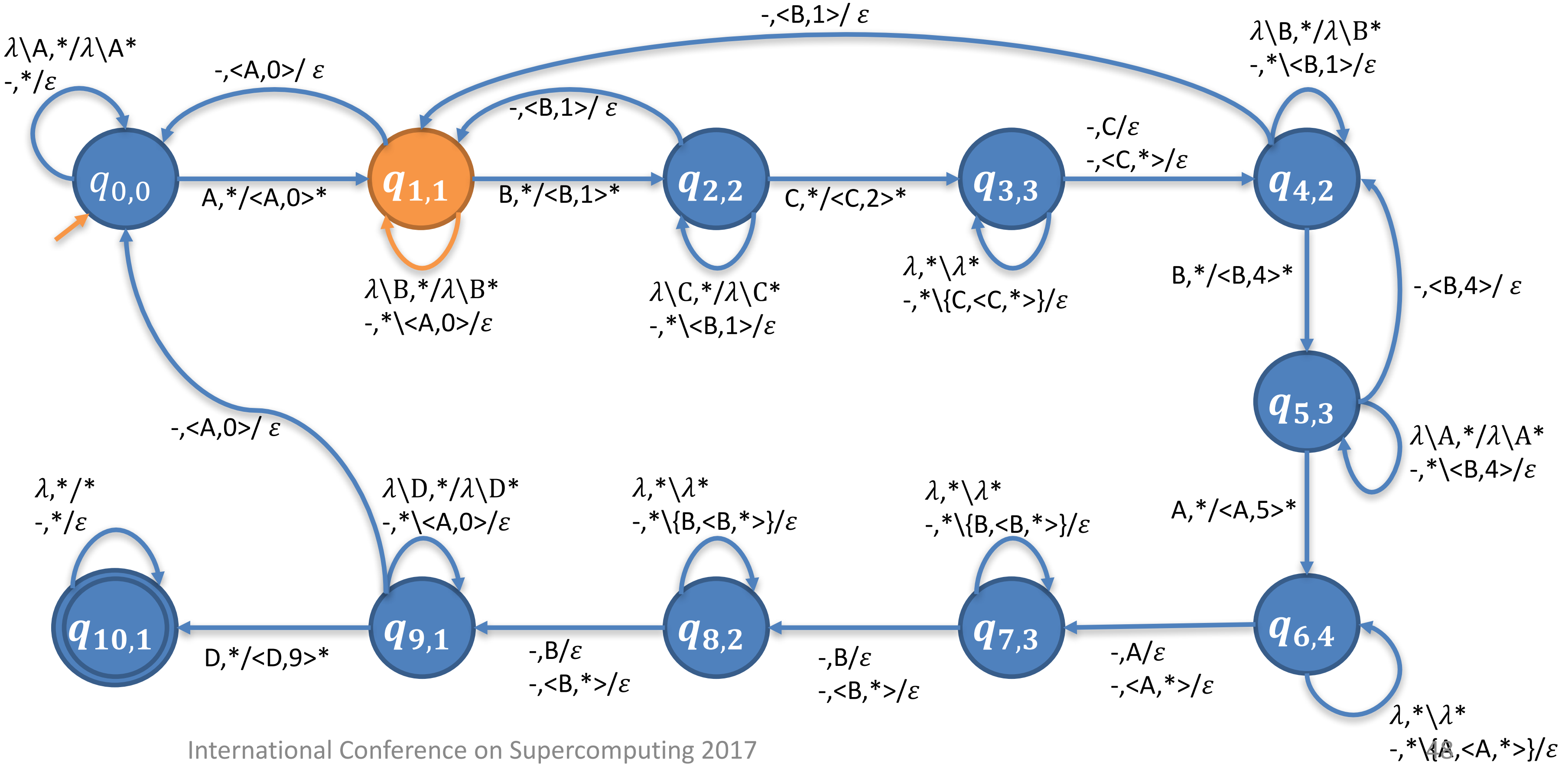
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



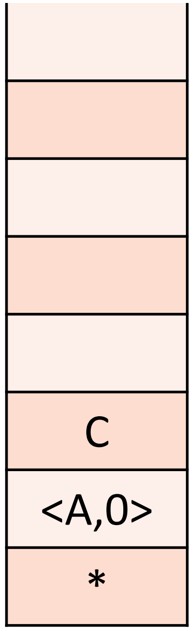


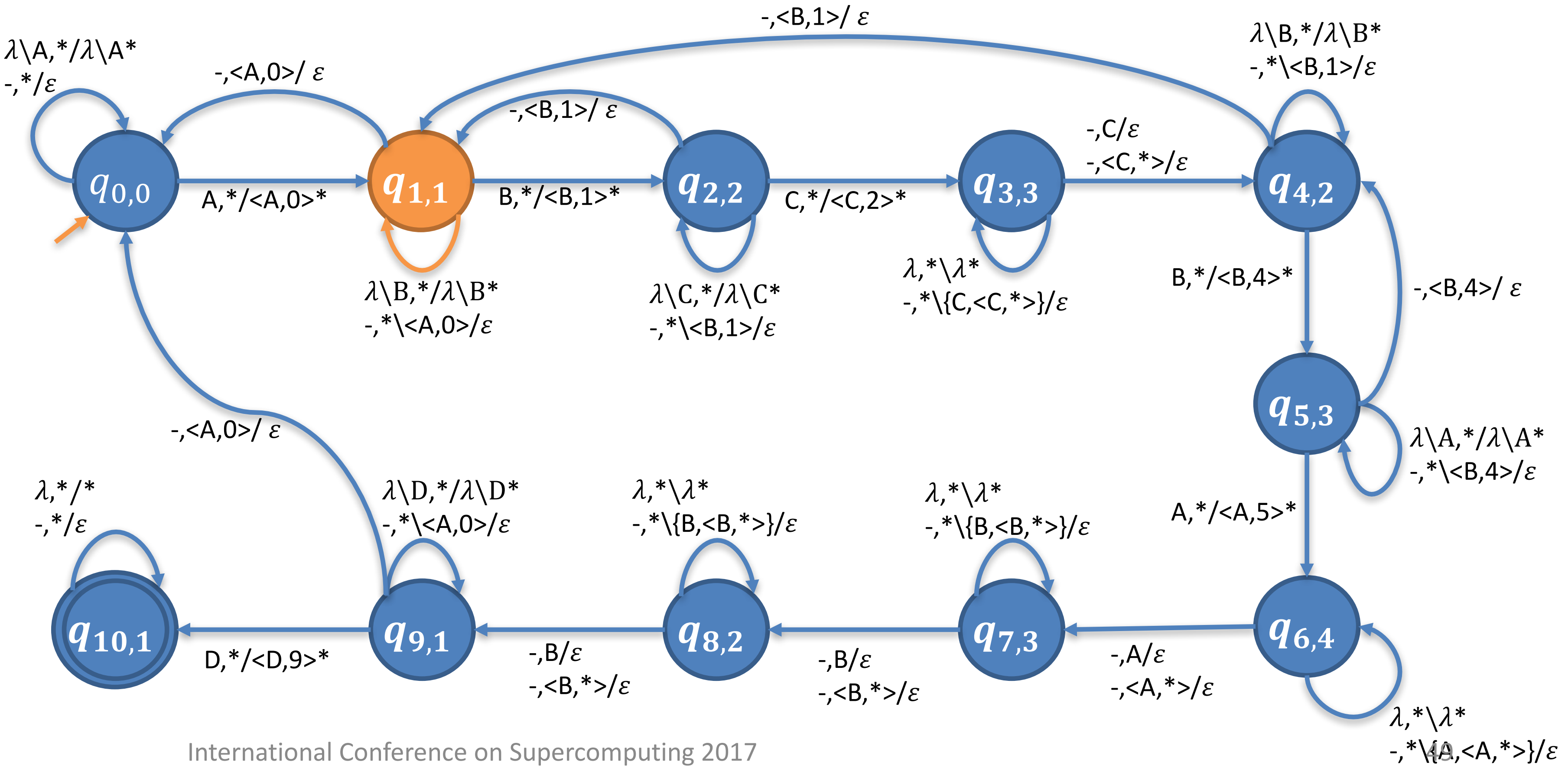
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



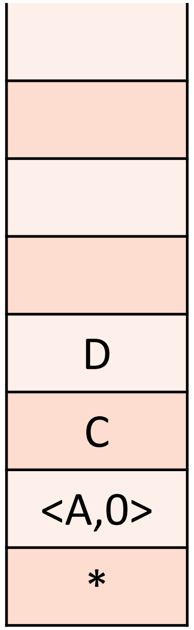


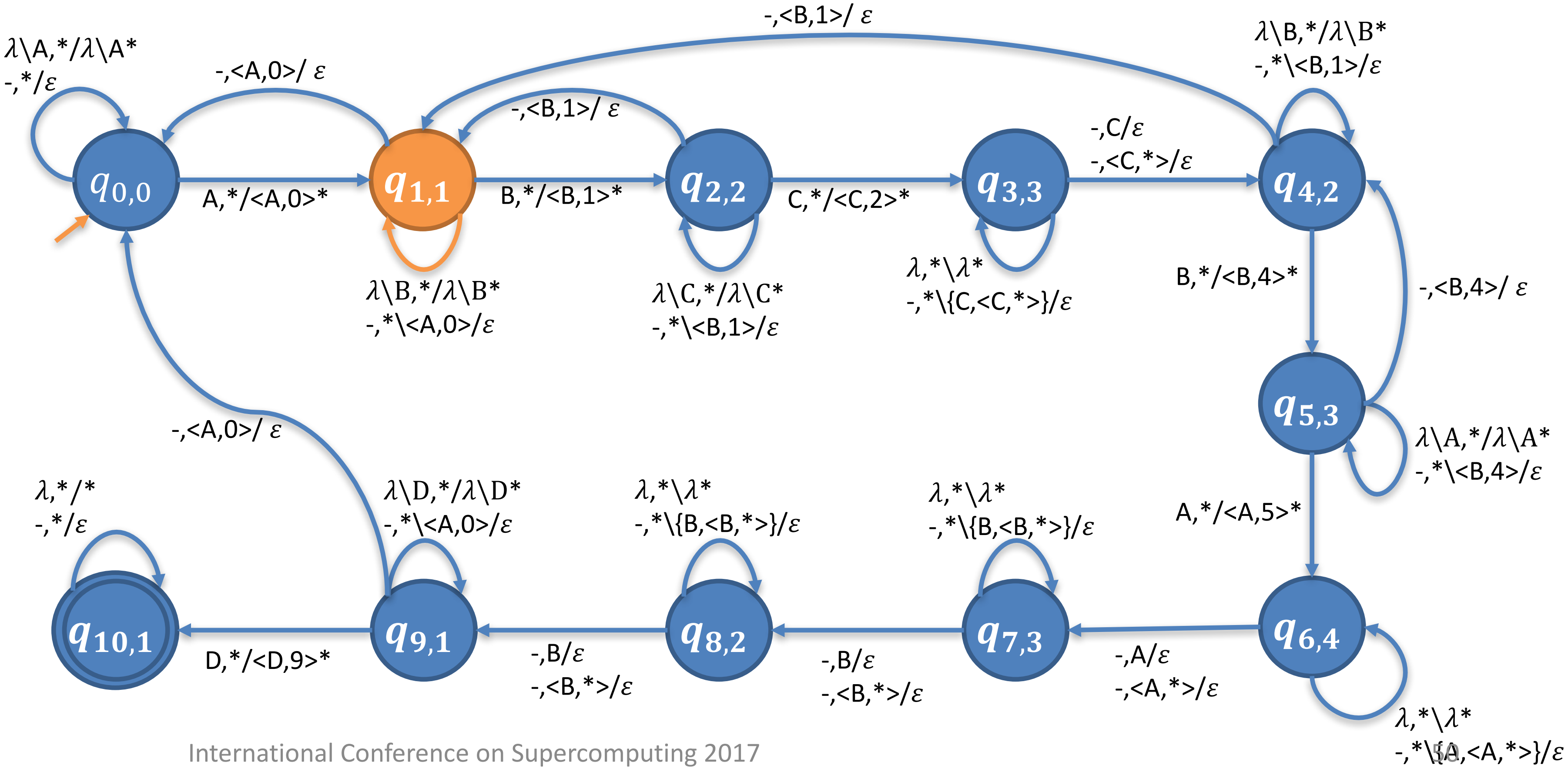
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



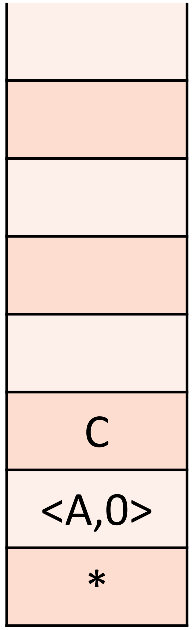


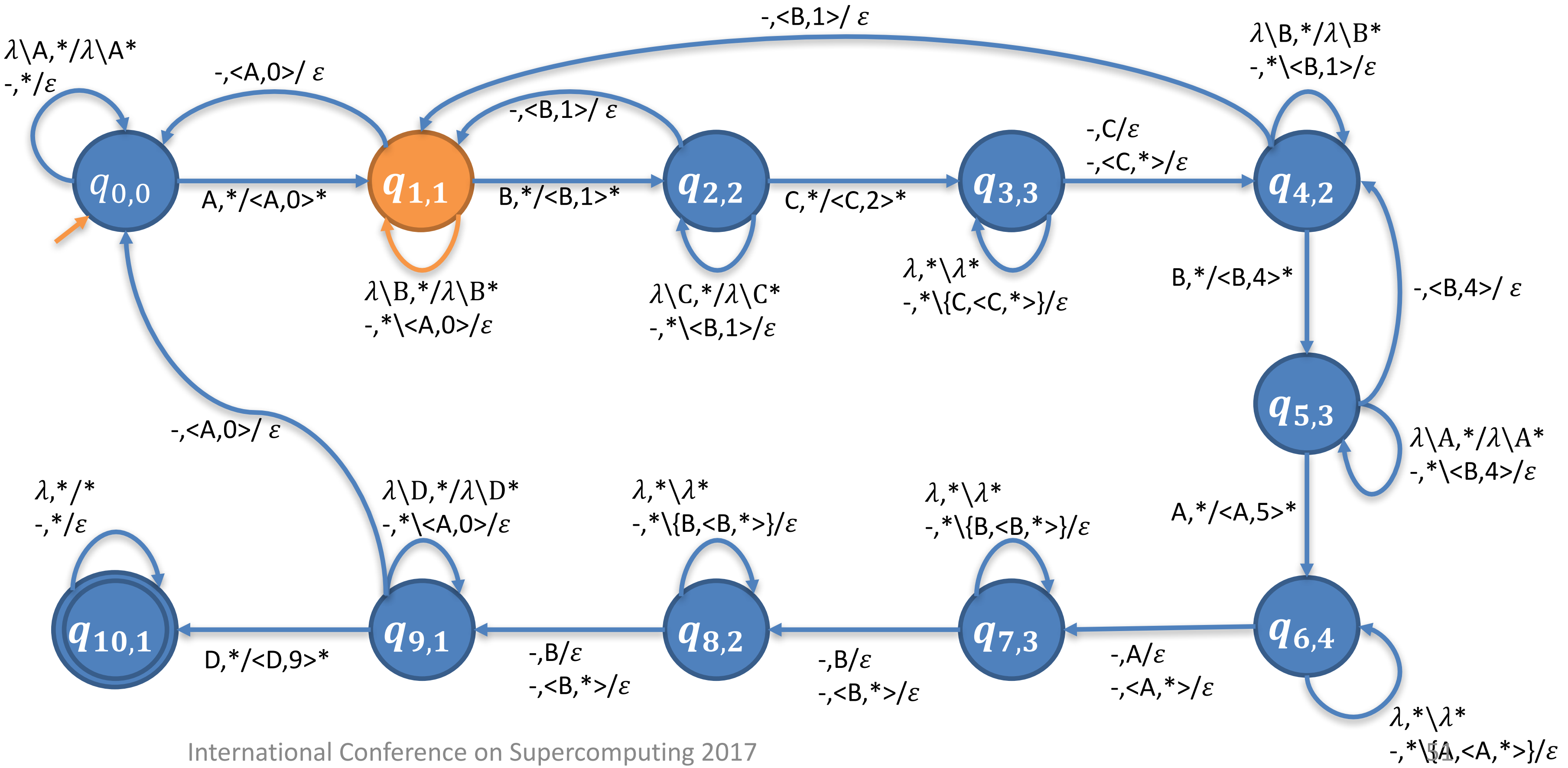
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



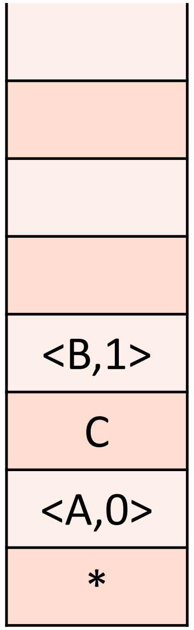


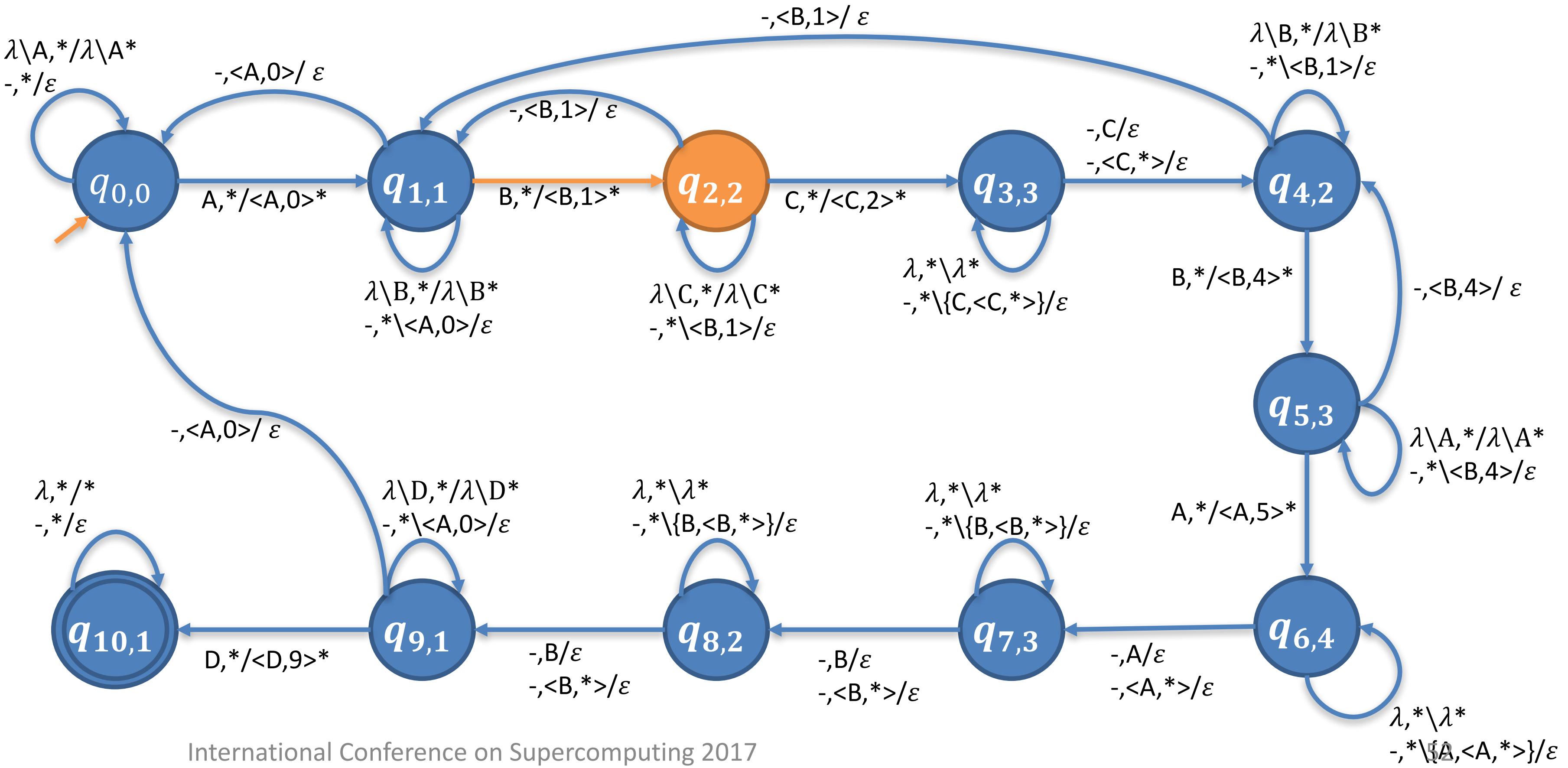
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



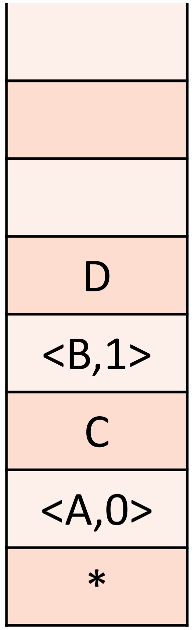


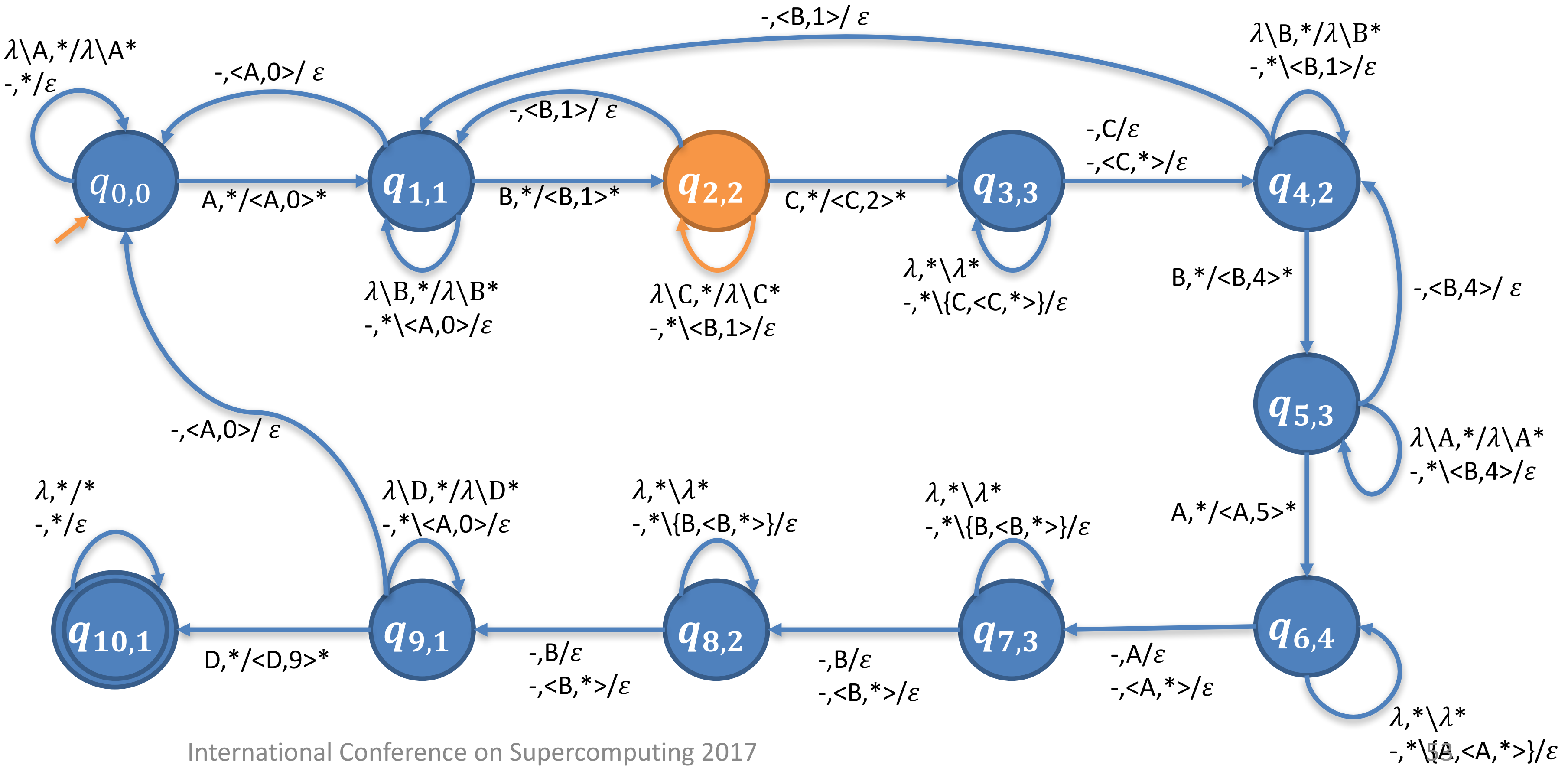
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



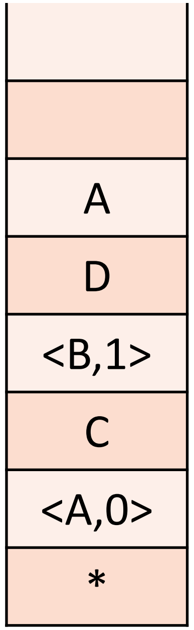


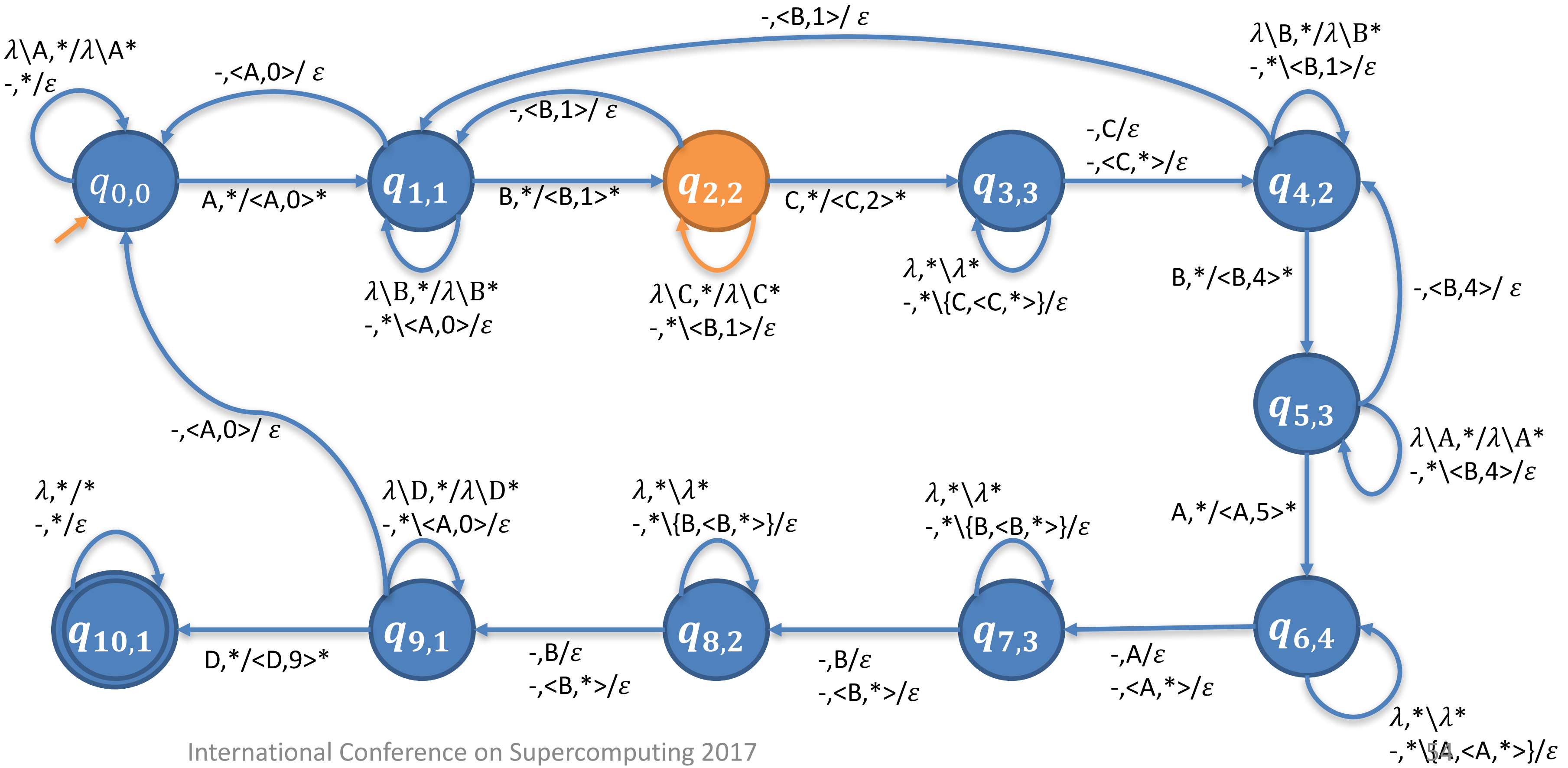
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



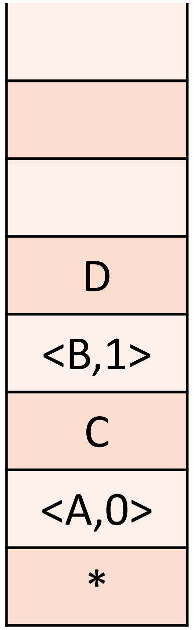


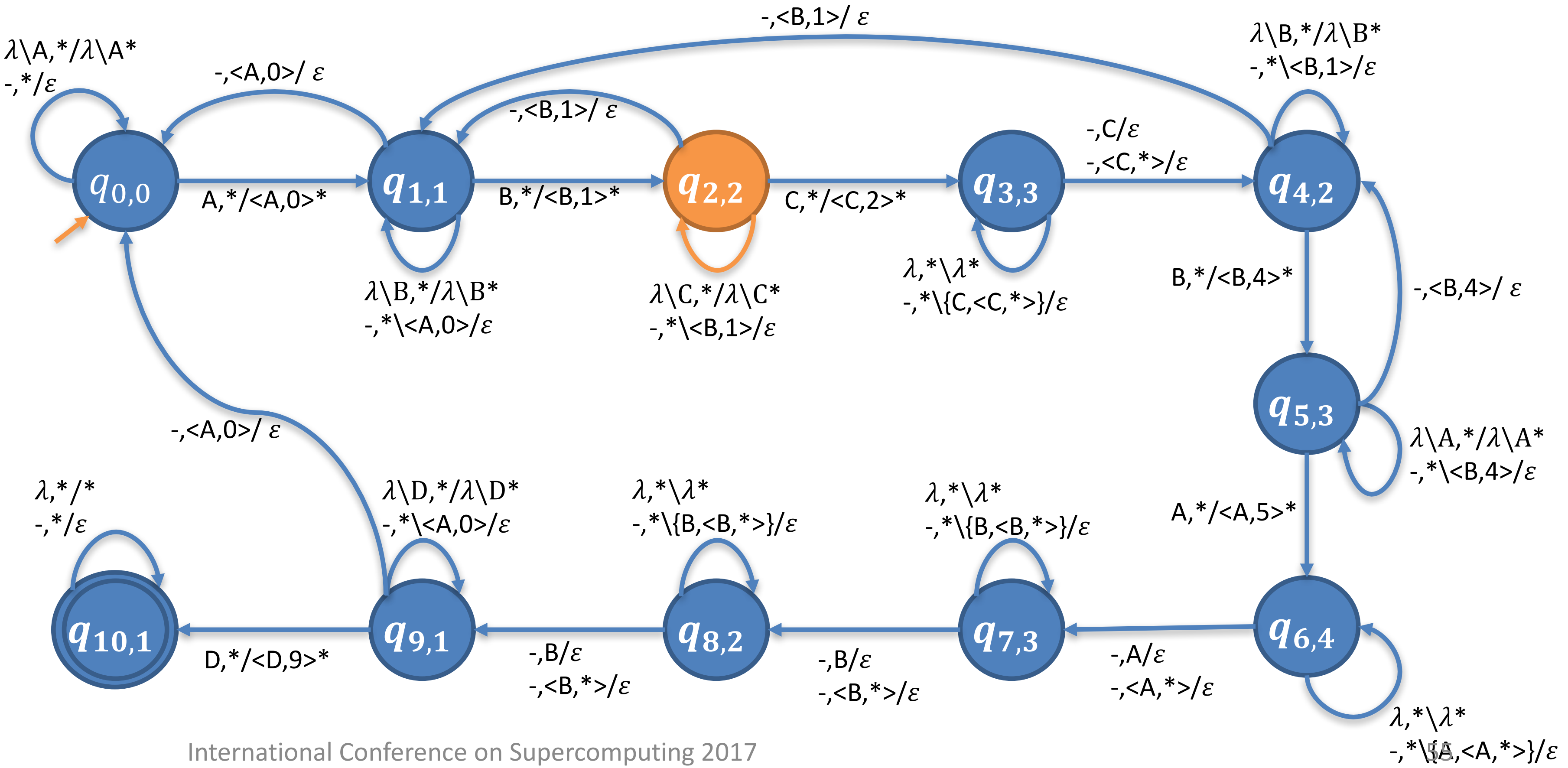
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



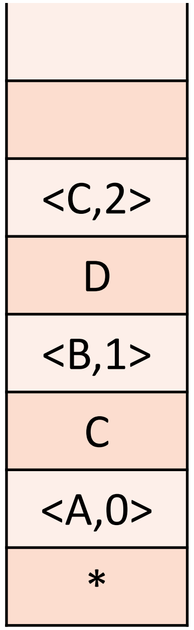


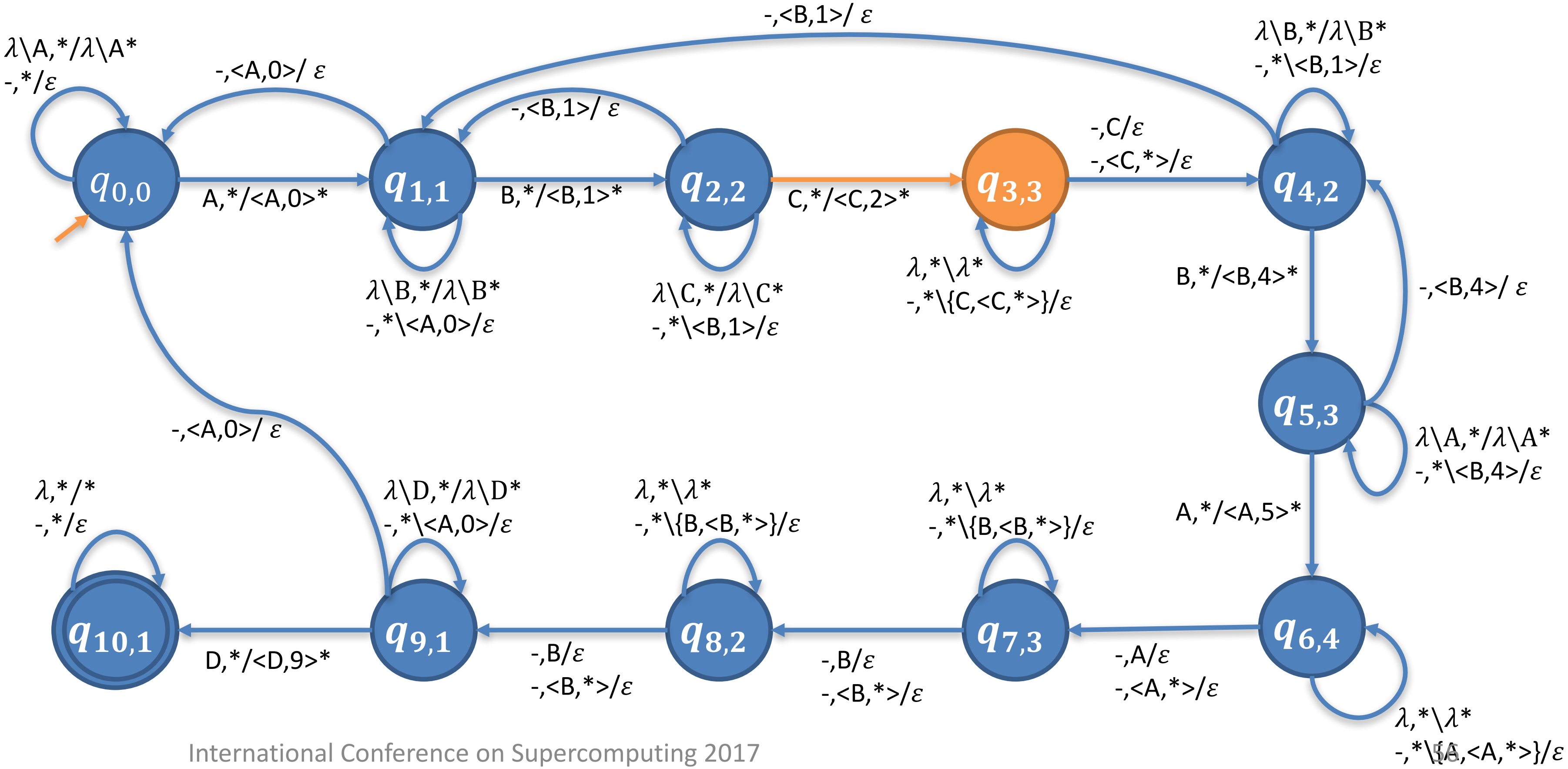
Input tree: A A – C D – B D A – C – – B D – A – – – – – B D



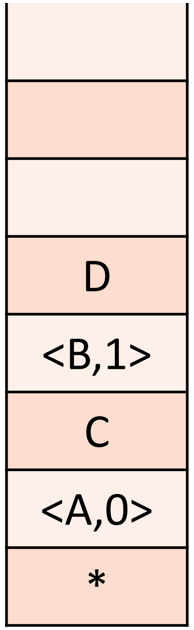


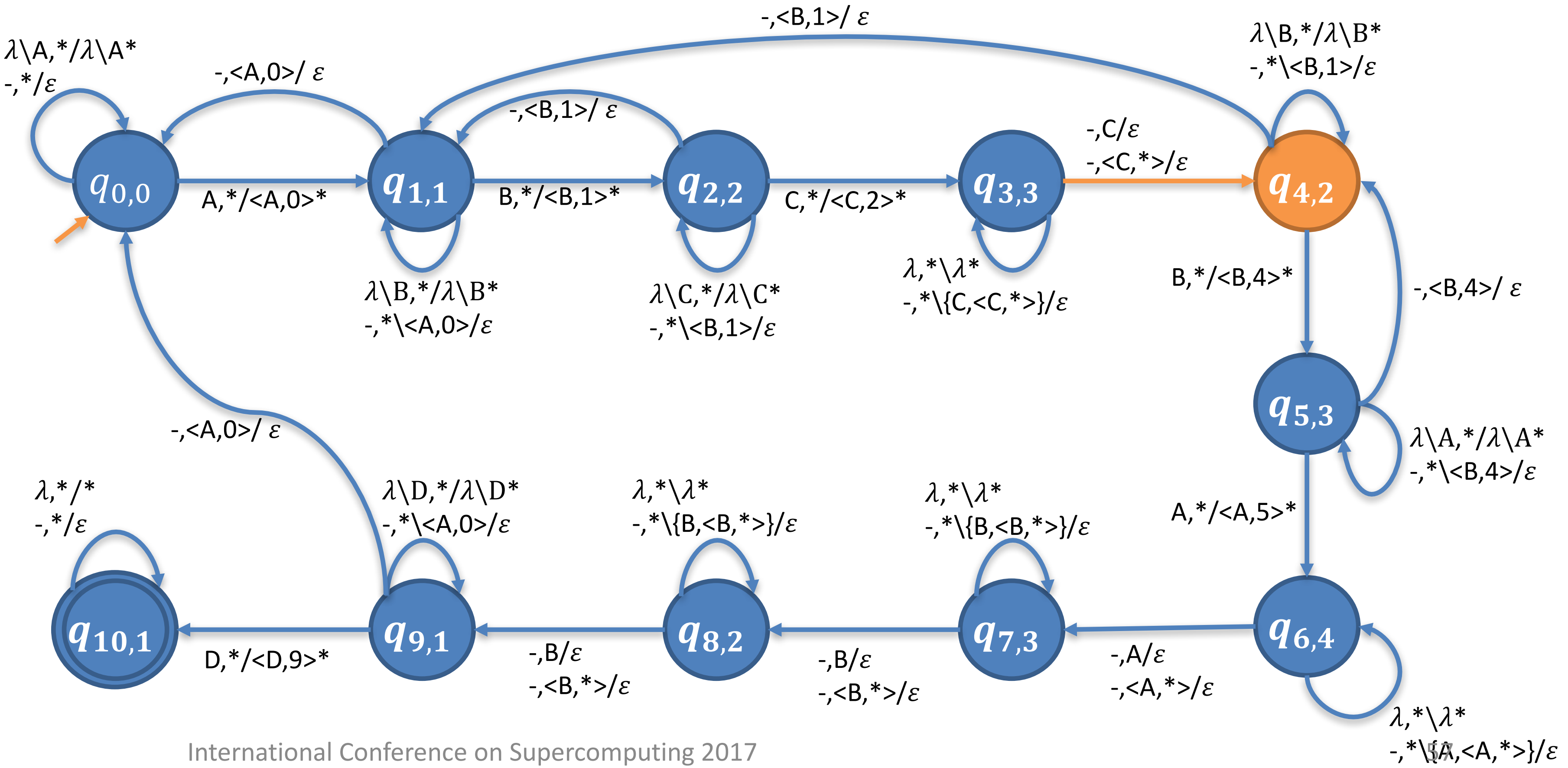
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



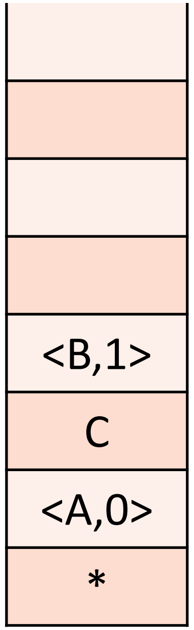


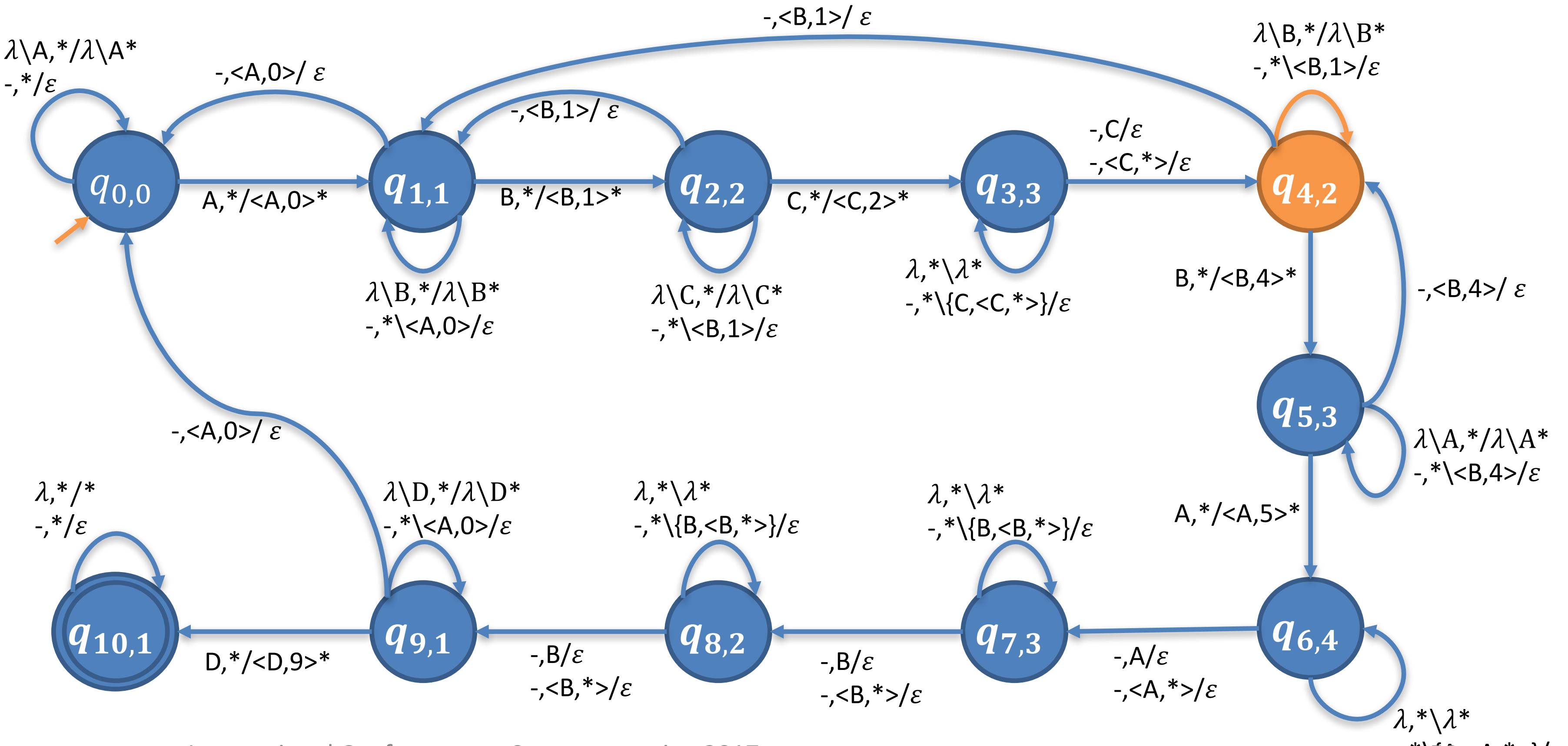
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



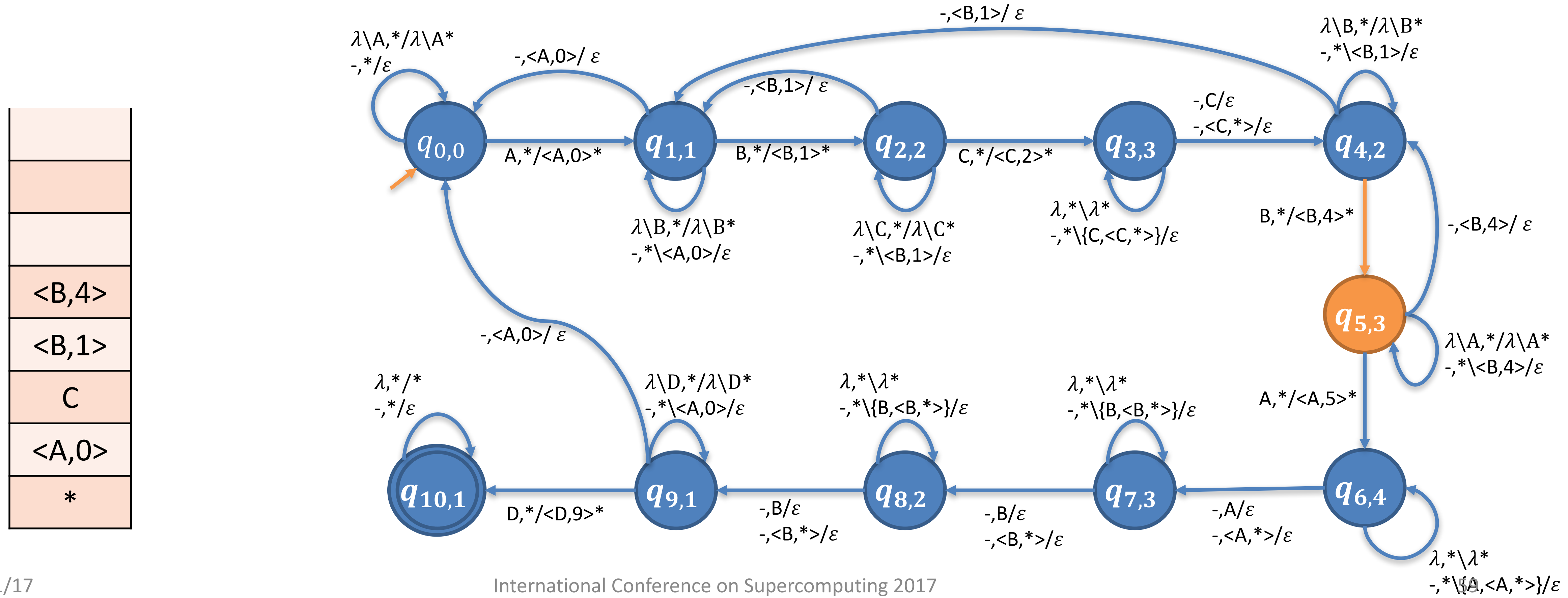


Input tree: A A – C D – B D A – C – – B D – A – – – – – B D

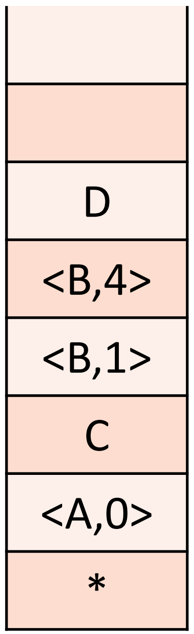


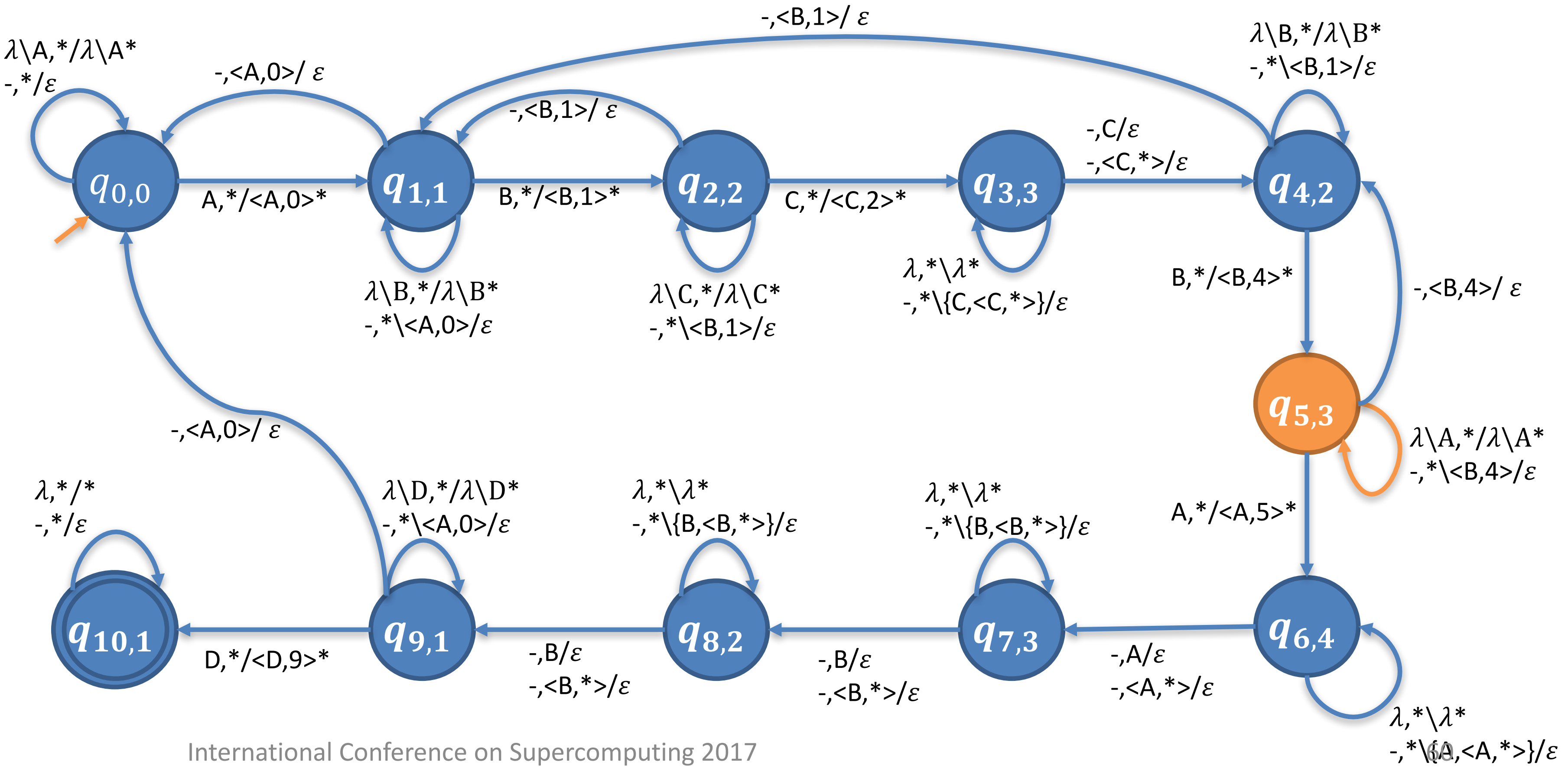


Input tree: A A – C D – B D A – C – – B D – A – – – – – B D

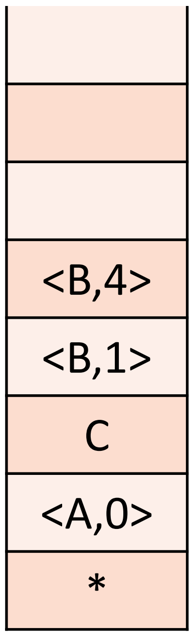


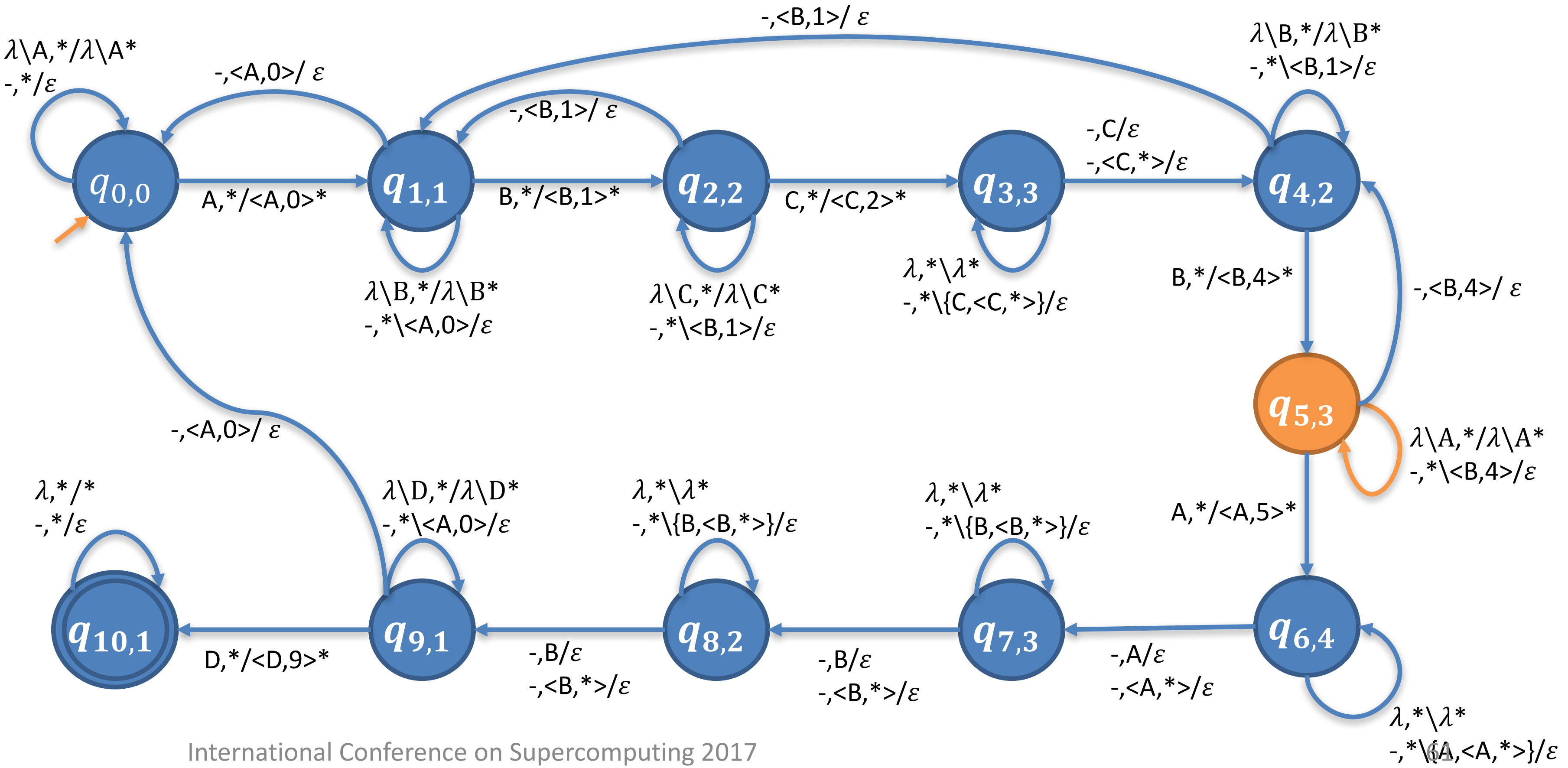
Input tree: A A – C D – B D A – C – – B D – A – – – – – B D



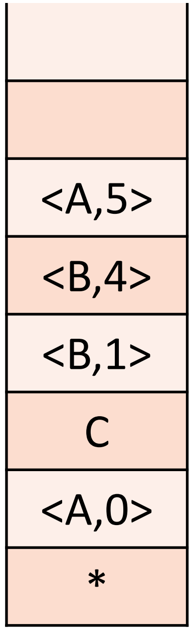


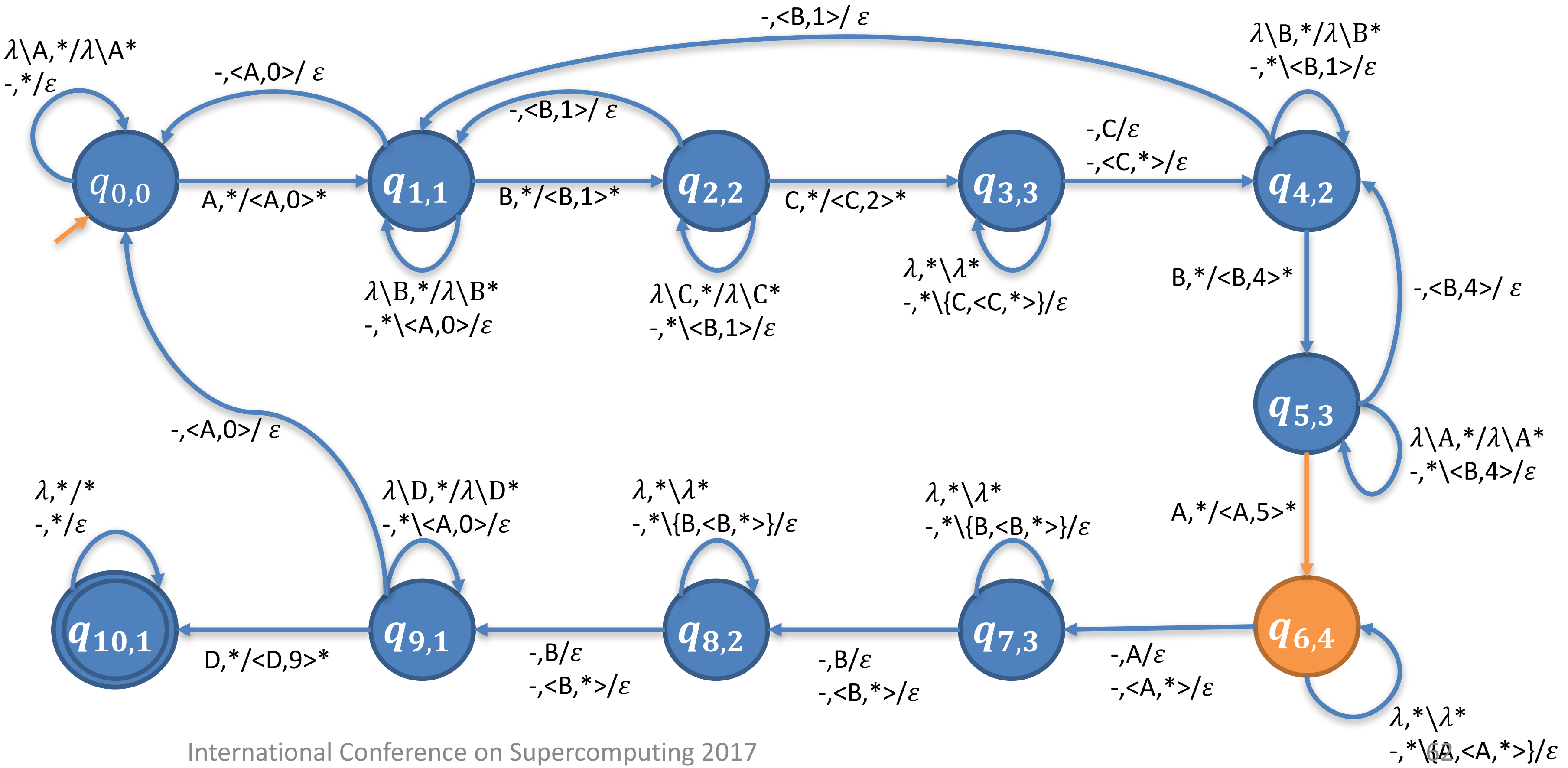
Input tree: A A – C D – B D A – C – – B D – A – – – – B D





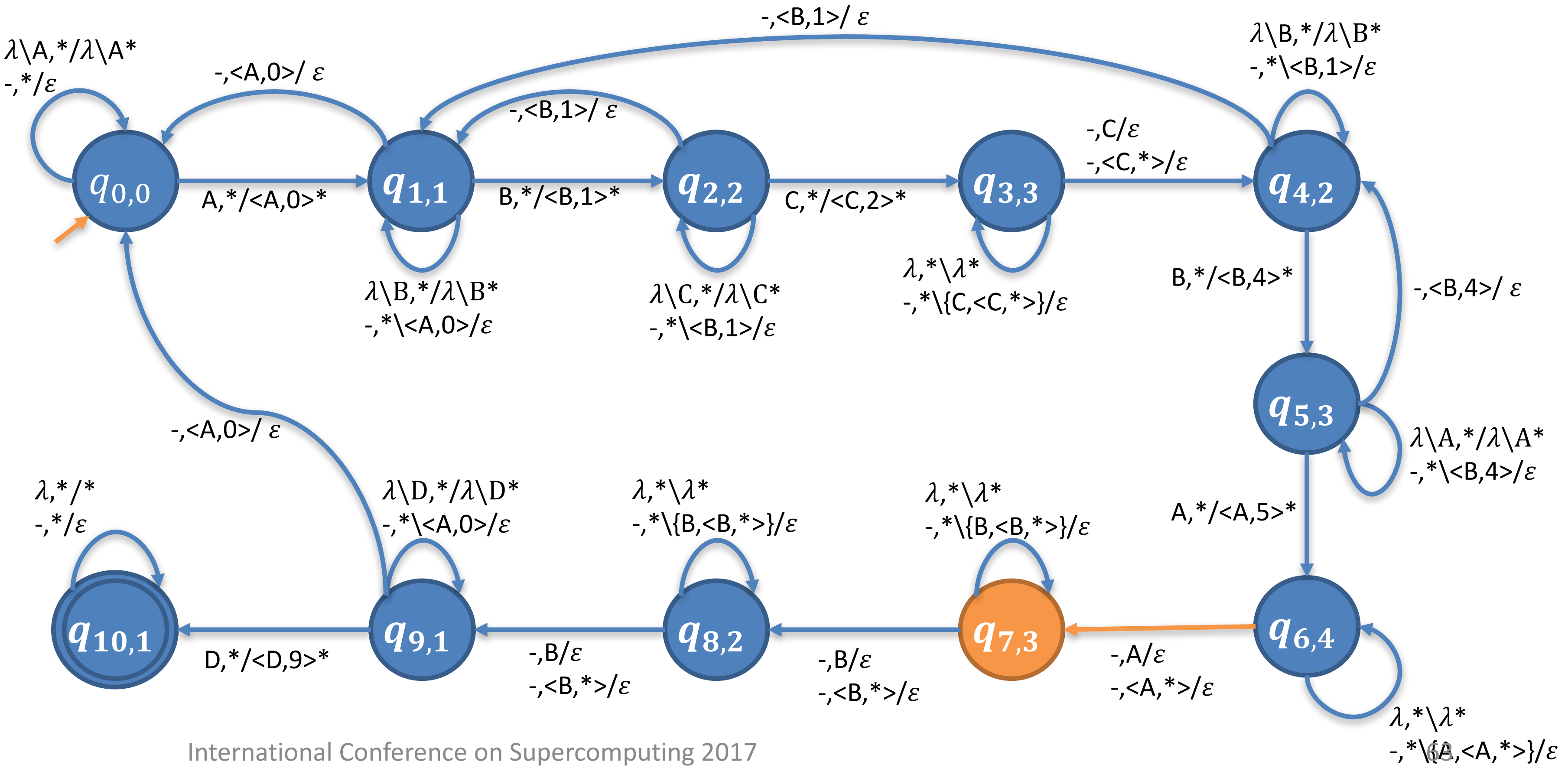
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



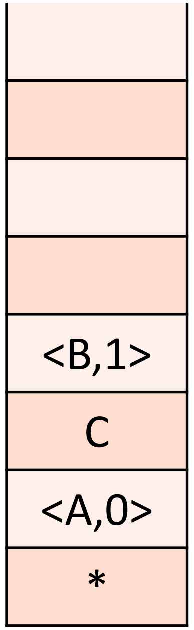


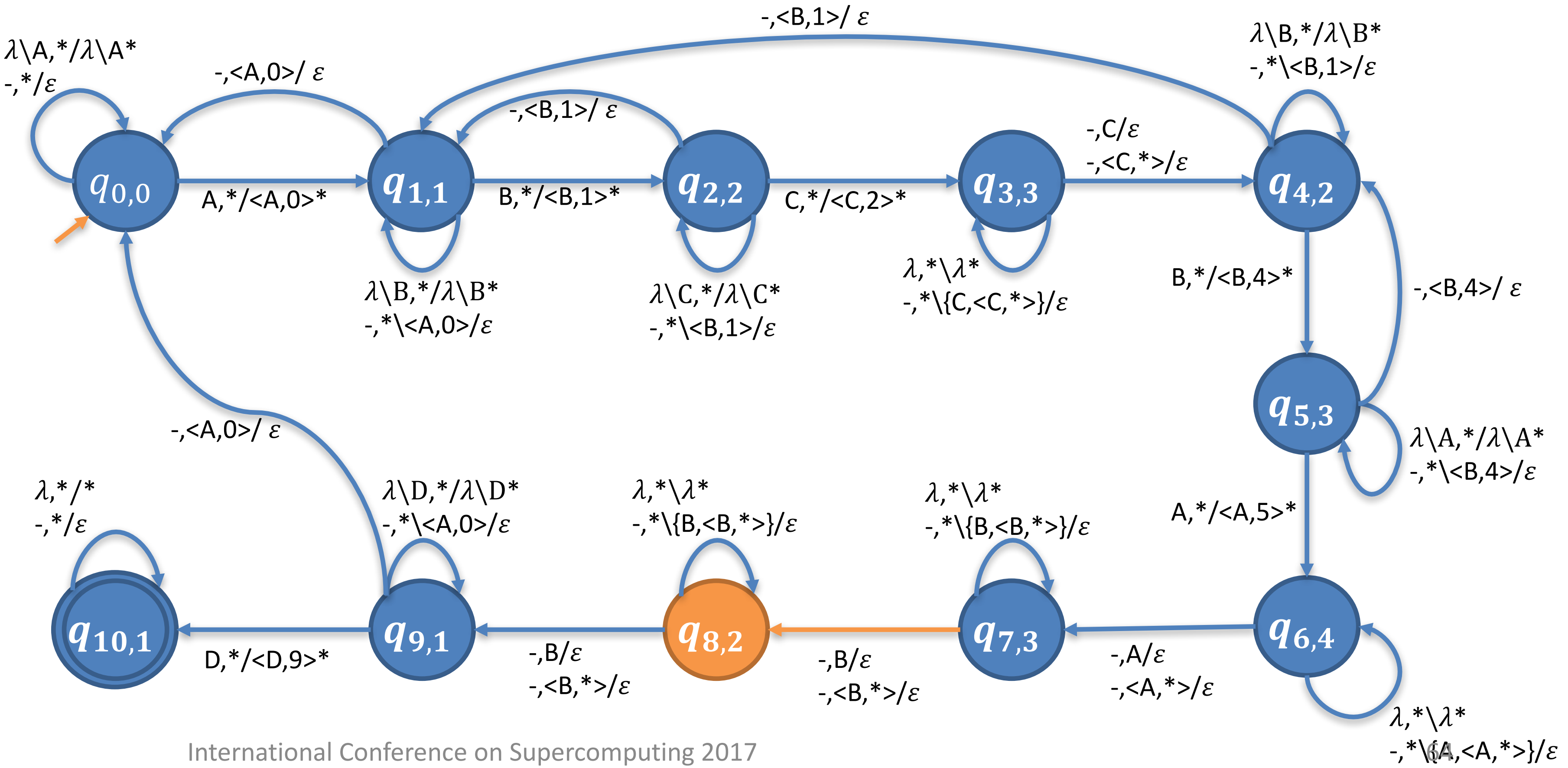
Input tree: A A – C D – B D A – C – – B D – A – – – – – B D

<B,4>
<B,1>
C
<A,0>
*

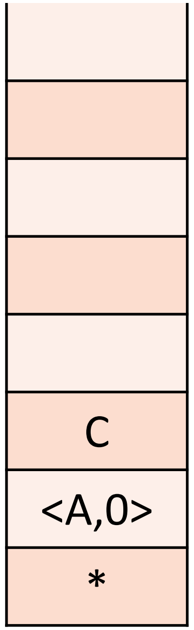


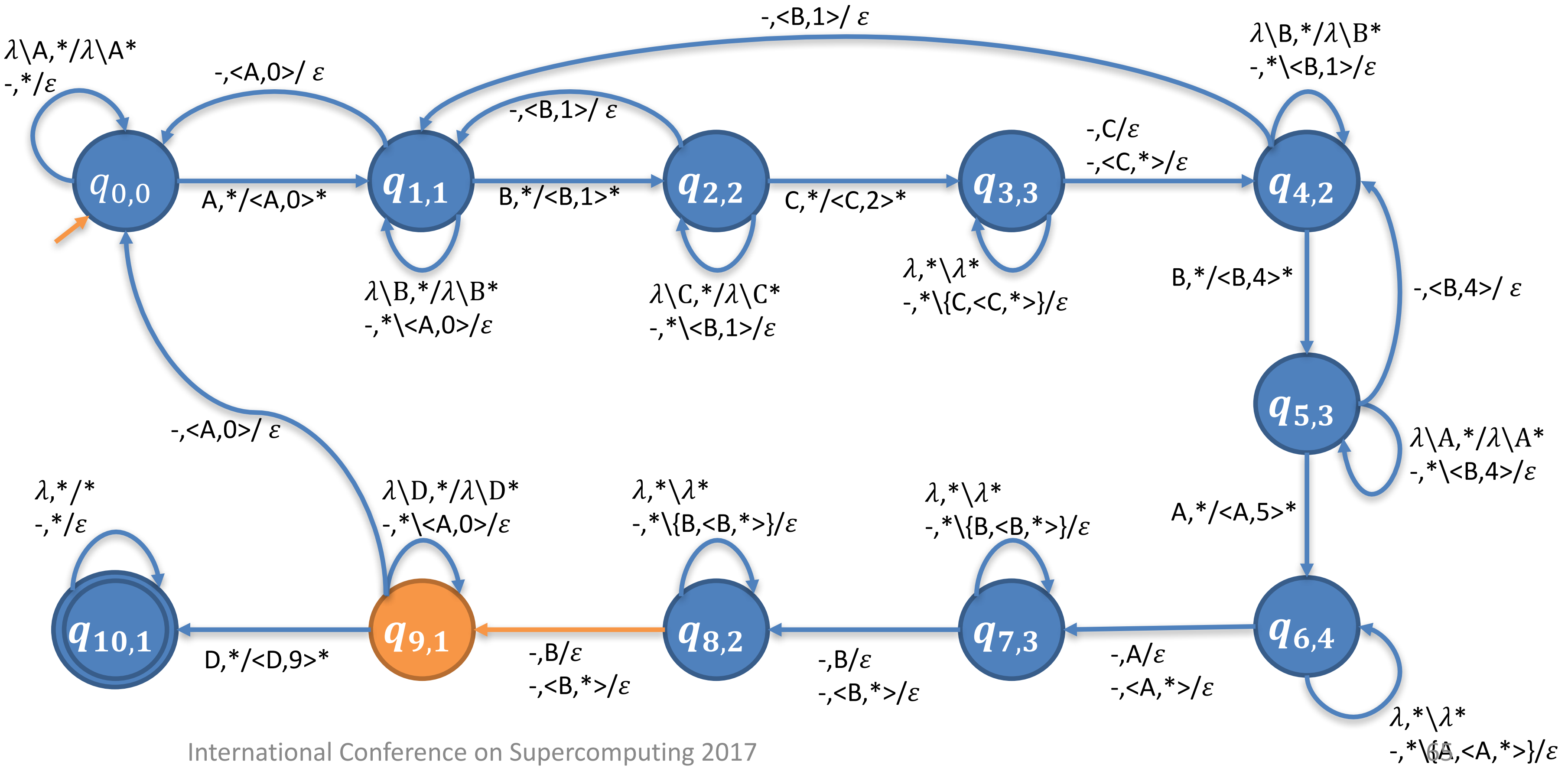
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



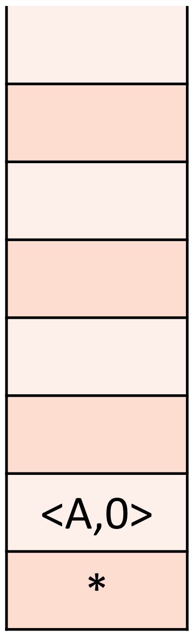


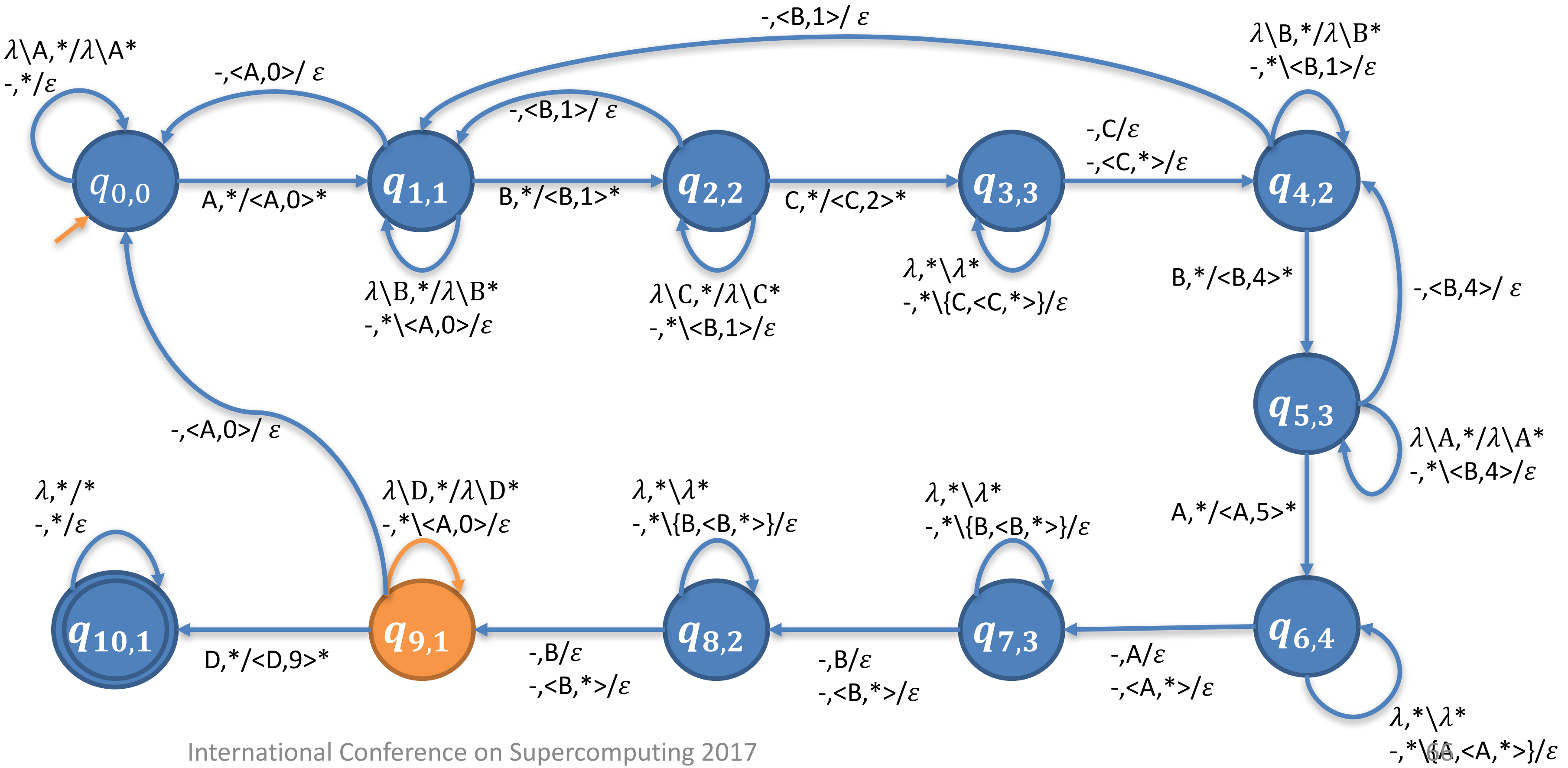
Input tree: A A – C D – B D A – C – – B D – A – – – – B D



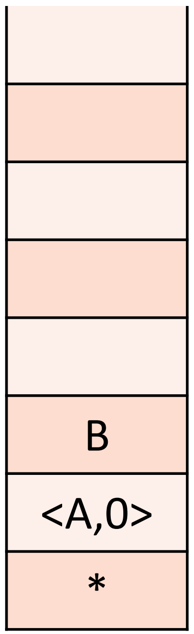


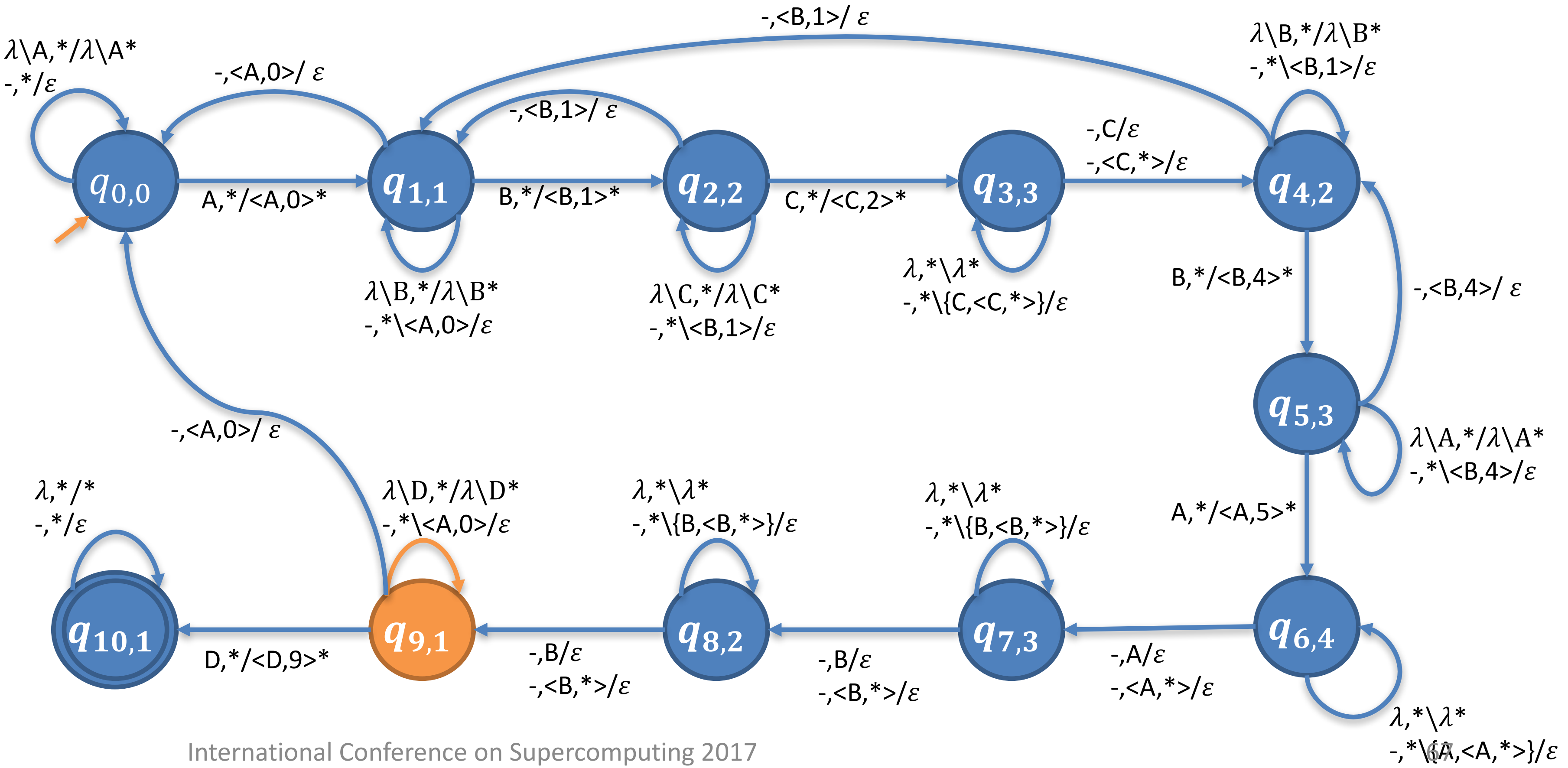
Input tree: A A – C D – B D A – C – – B D – A – – – – B D





Input tree: A A – C D – B D A – C – – B D – A – – – – – B D





Input tree: A A – C D – B D A – C – – B D – A – – – – – B D