Efficiently learning simple timed automata

Sicco Verwer

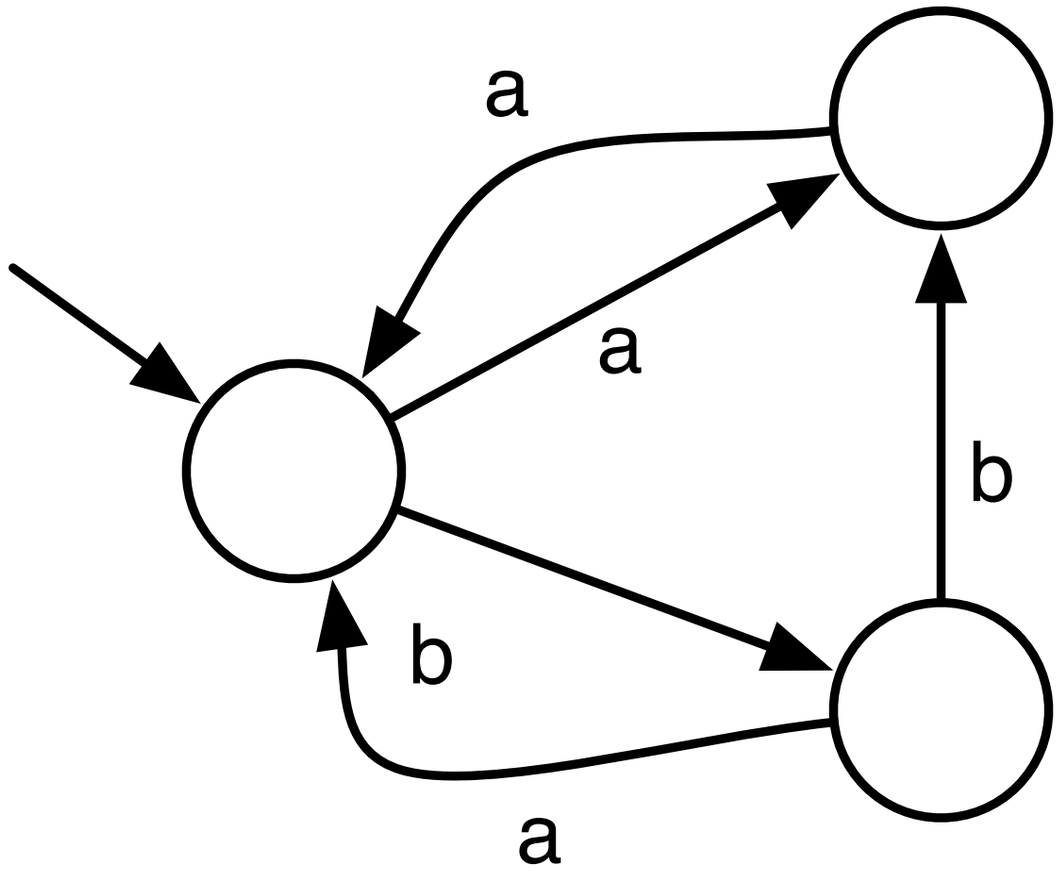
2009

TU Delft

Overview

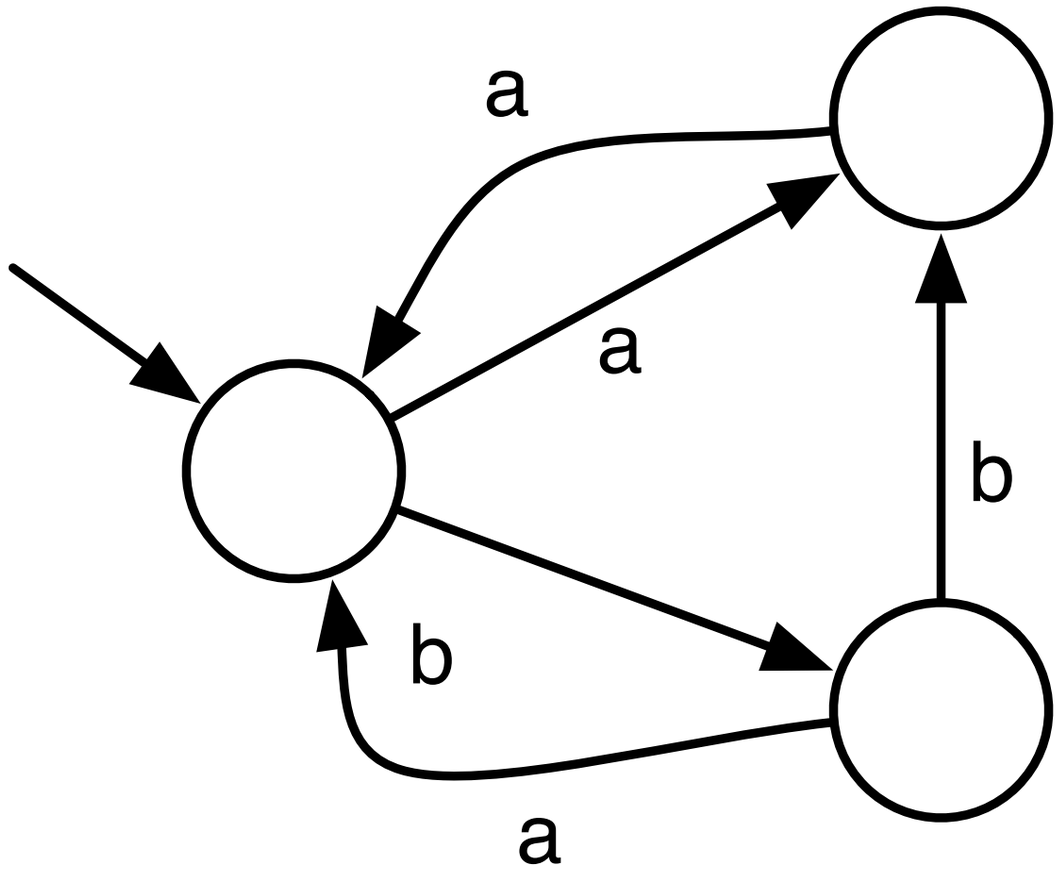
- Automata & how to learn them
- Real-time automata
- Learning real-time automata from labeled data
  - Results
- Learning real-time automata from unlabeled data
  - Results
- Conclusions

Automata



TU Delft

Automata and events



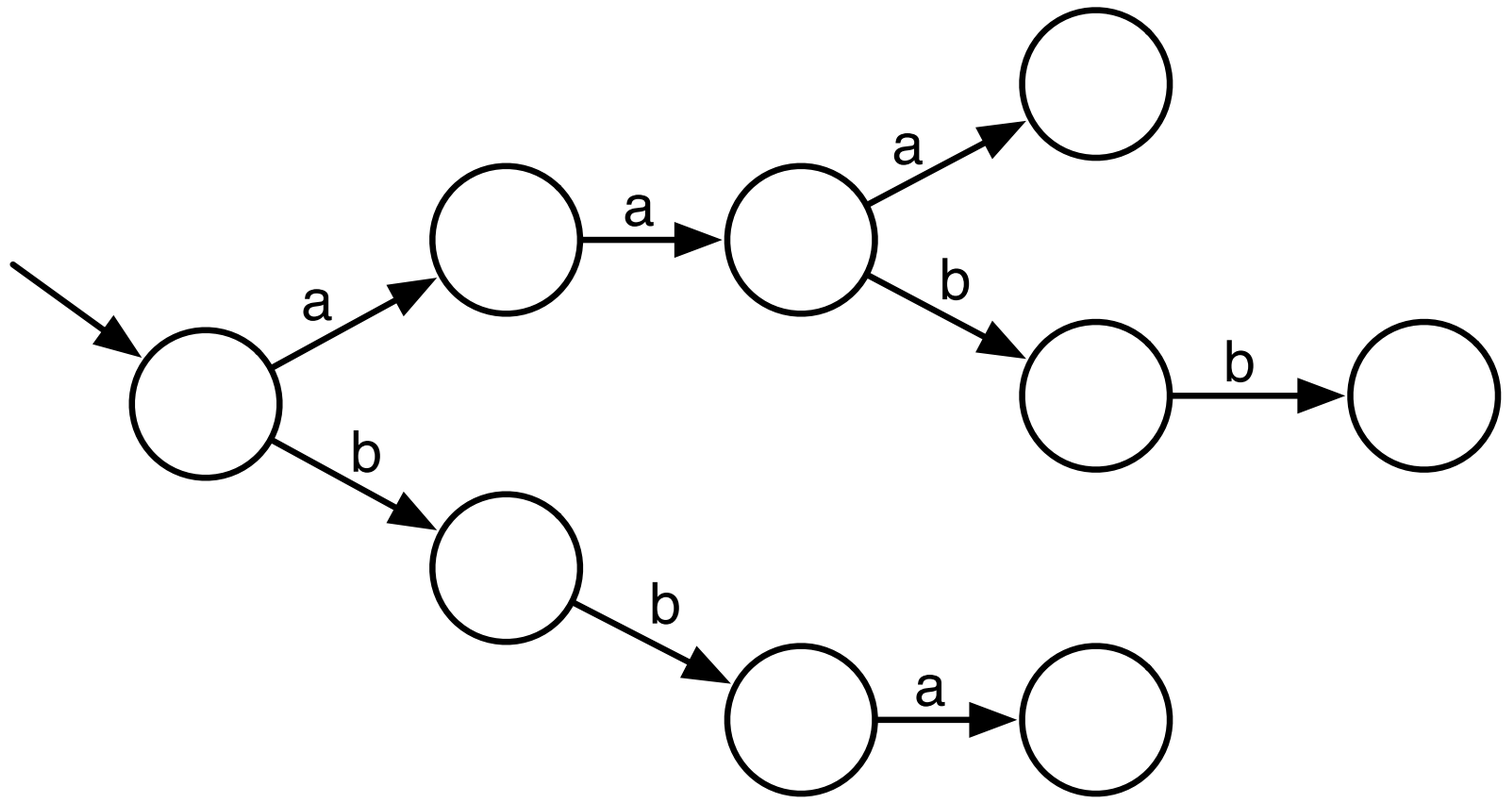
Produces strings: aa, aabb, aaaab, babba, bbab

TU Delft

Learning DFA

- Input:
  - Data
- Goal:
  - Find the smallest DFA that is consistent with the data
- NP-hard but learnable in the limit from polynomial time and data

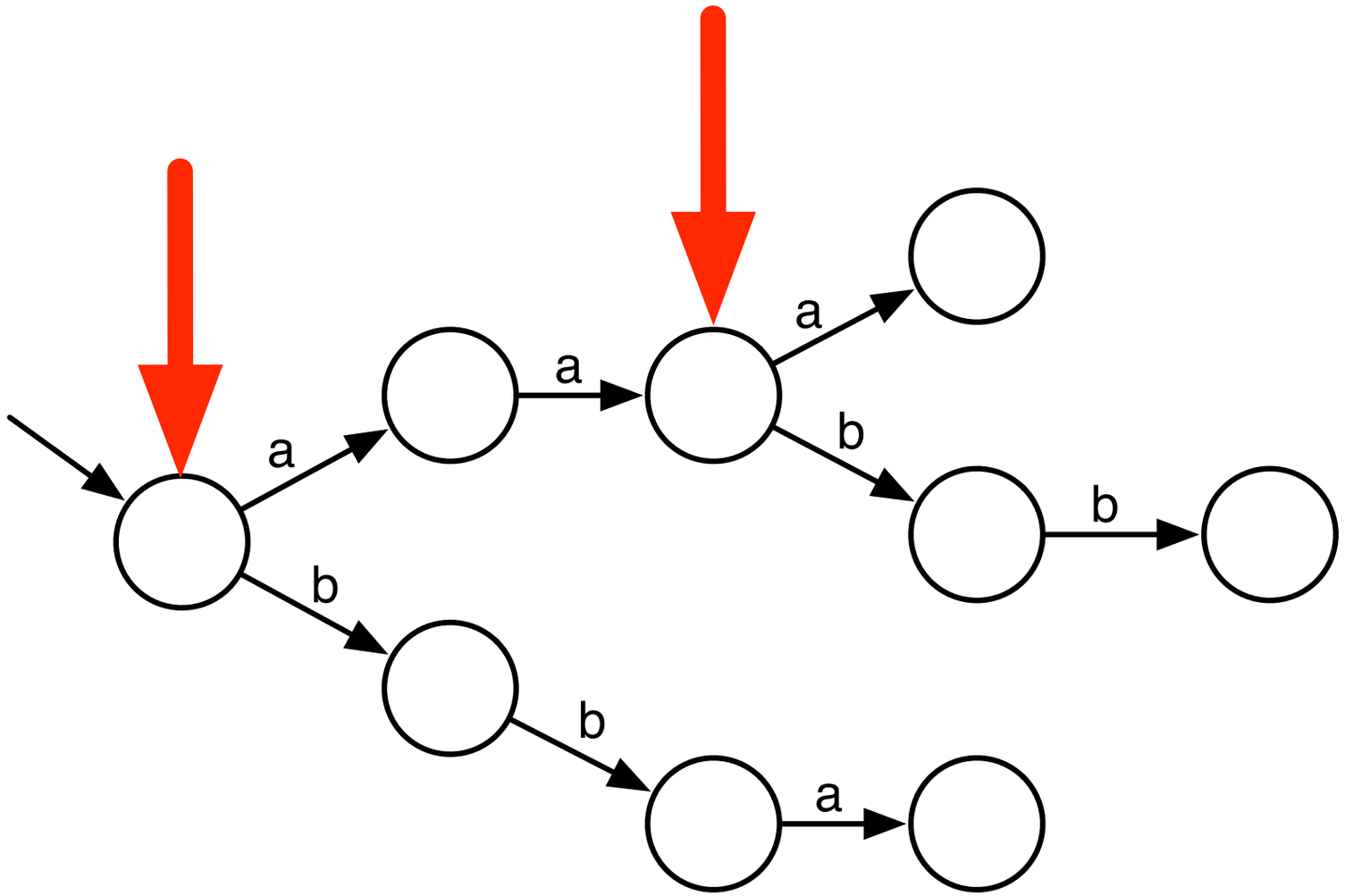
Learning DFAs



Some observations: a, aaa, aaa, aabb, b, bb, bb, bba
represented as a prefix tree

TU Delft

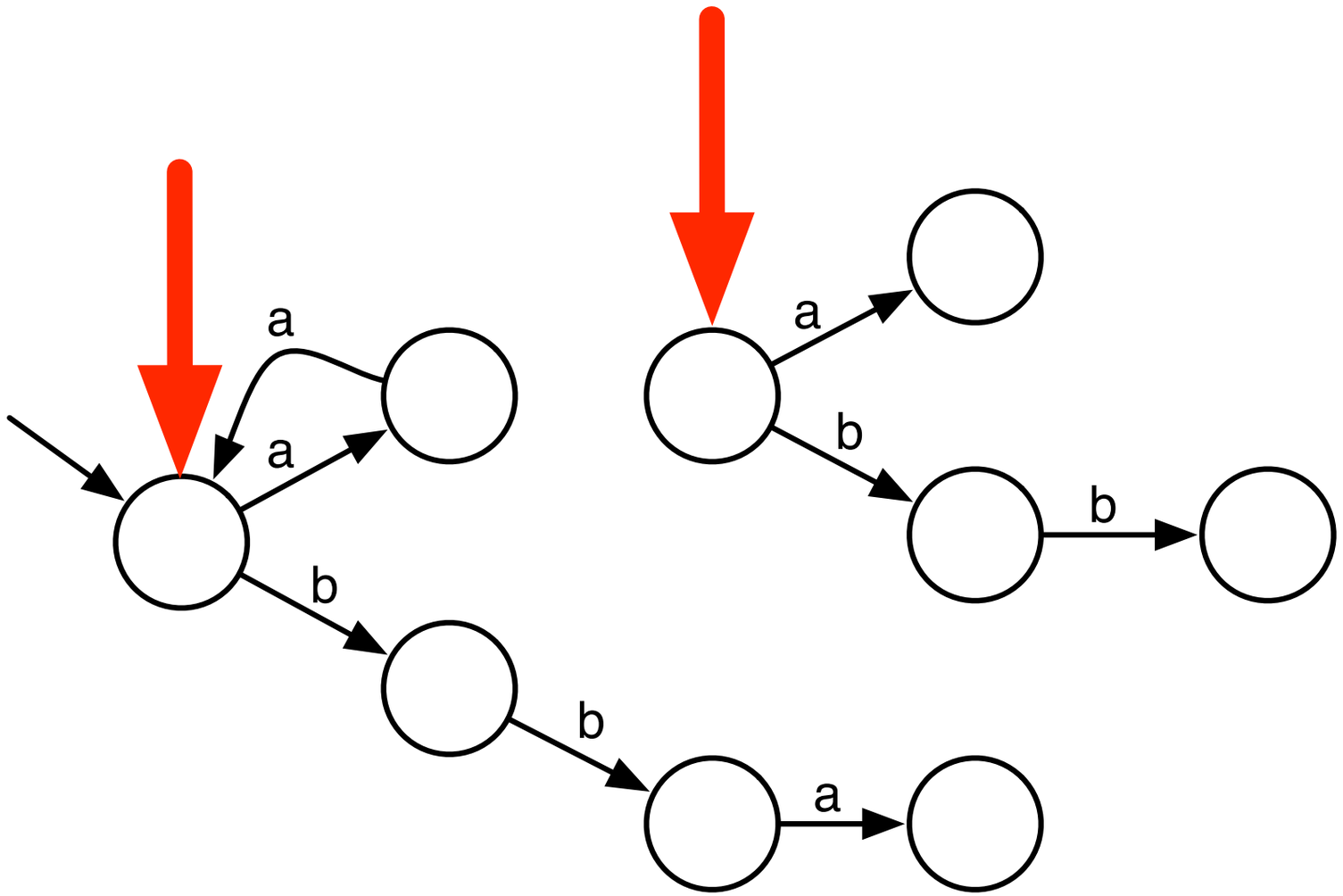
Learning DFAs



State merging:
select two nodes

TU Delft

Learning DFAs

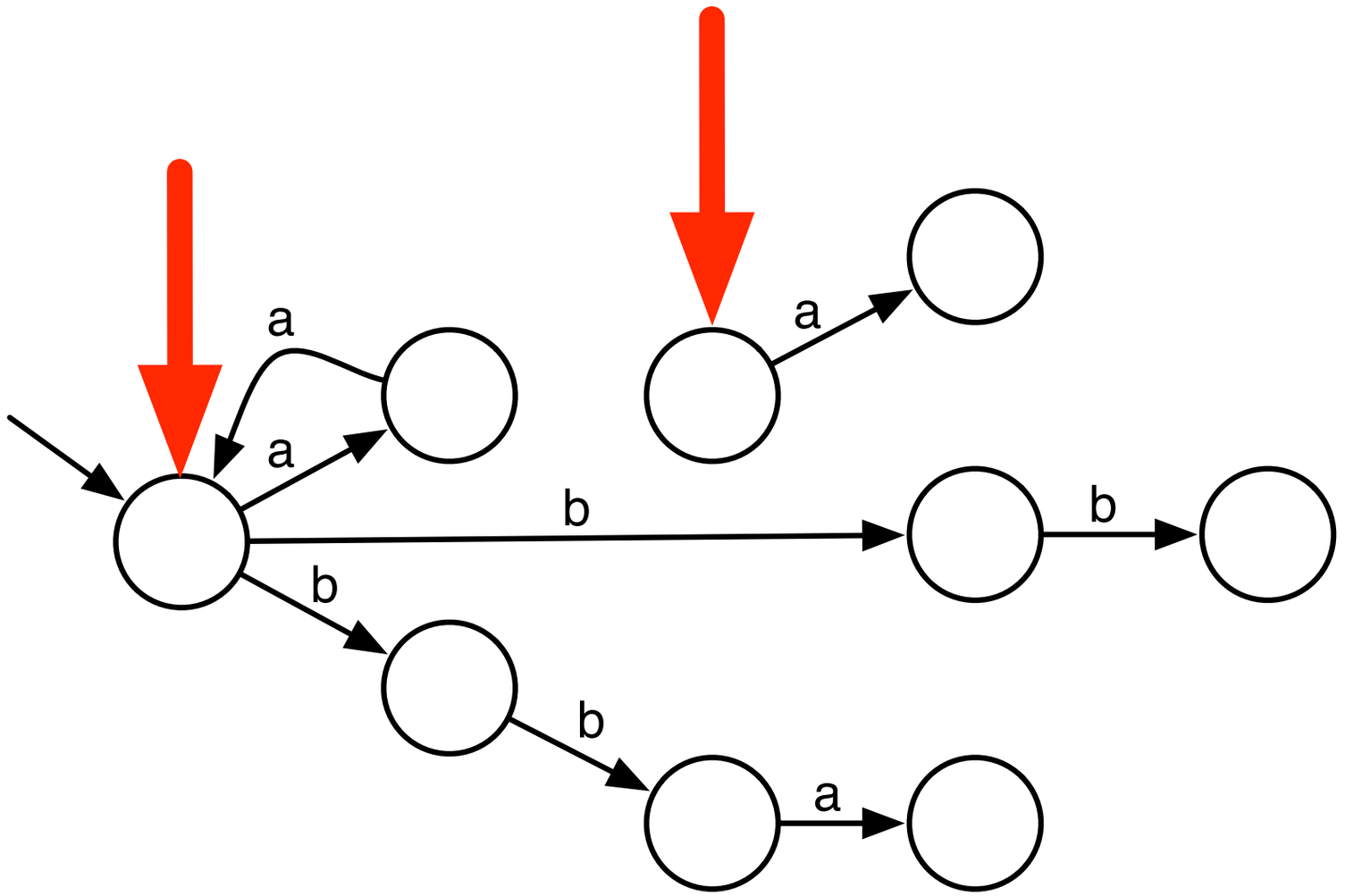


State merging:

move input transitions from one state to the other

TU Delft

Learning DFAs

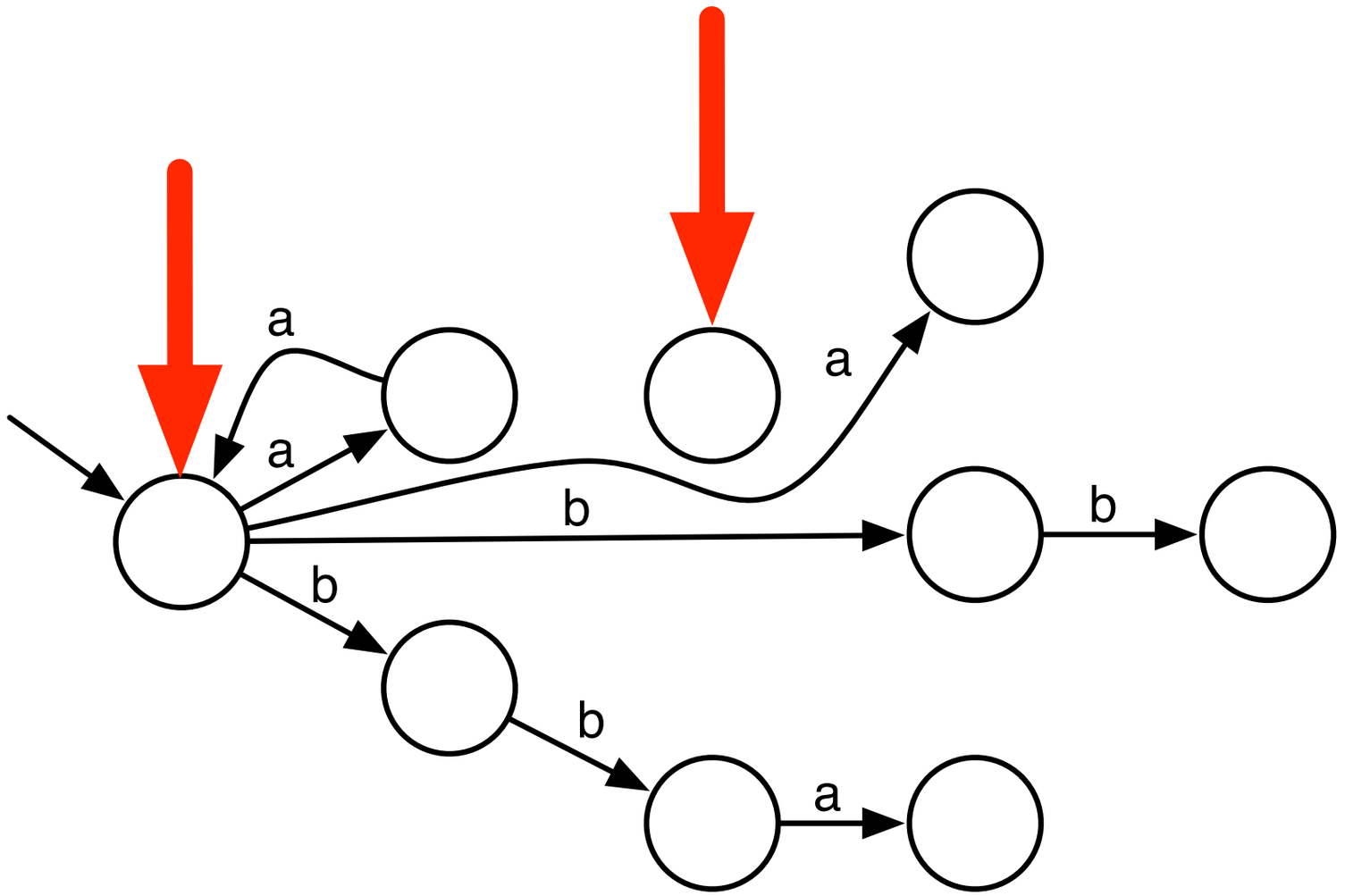


State merging:

move output transitions from one state to the other

TU Delft

Learning DFAs

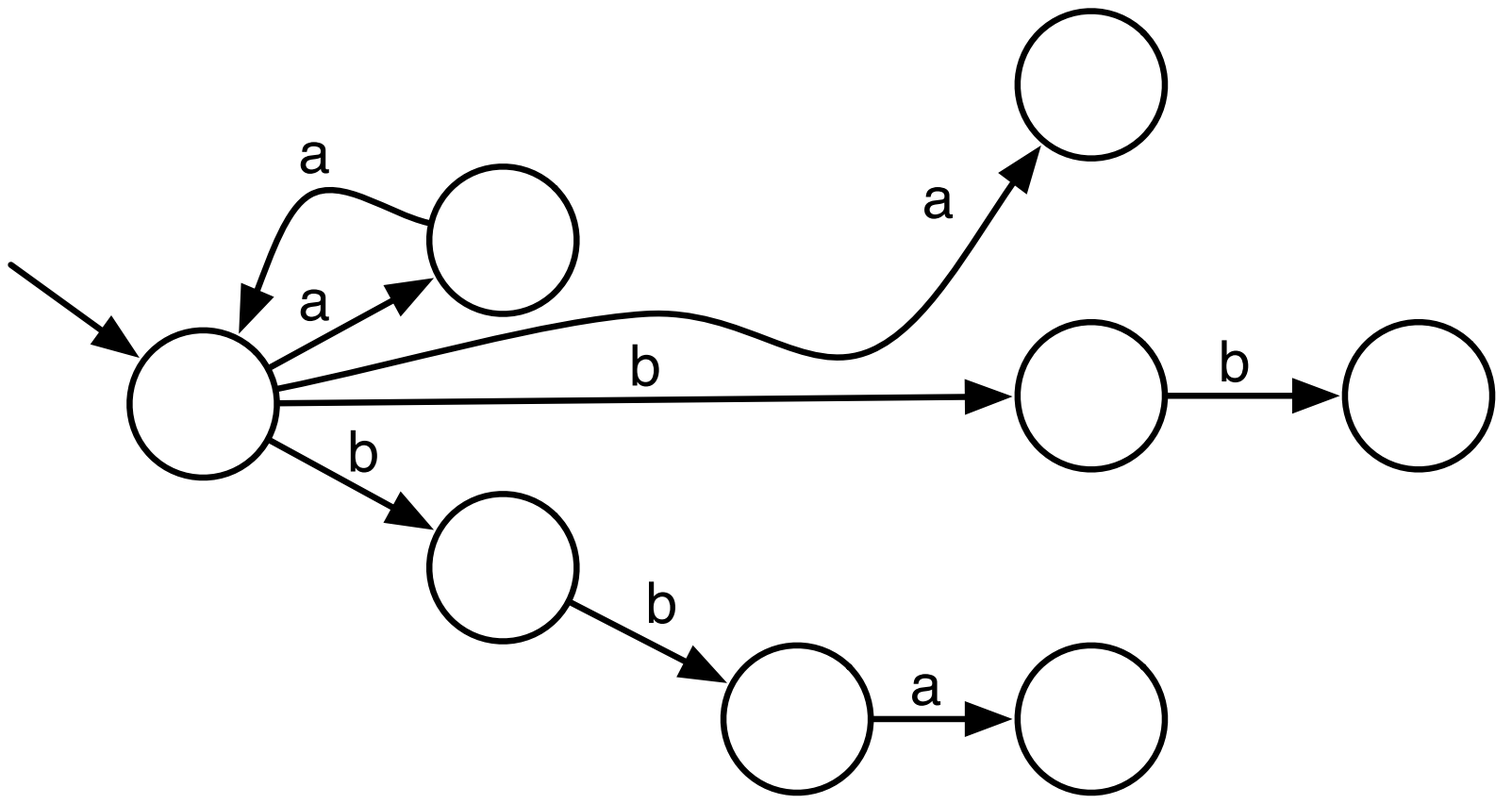


State merging:

move output transitions from one state to the other

TU Delft

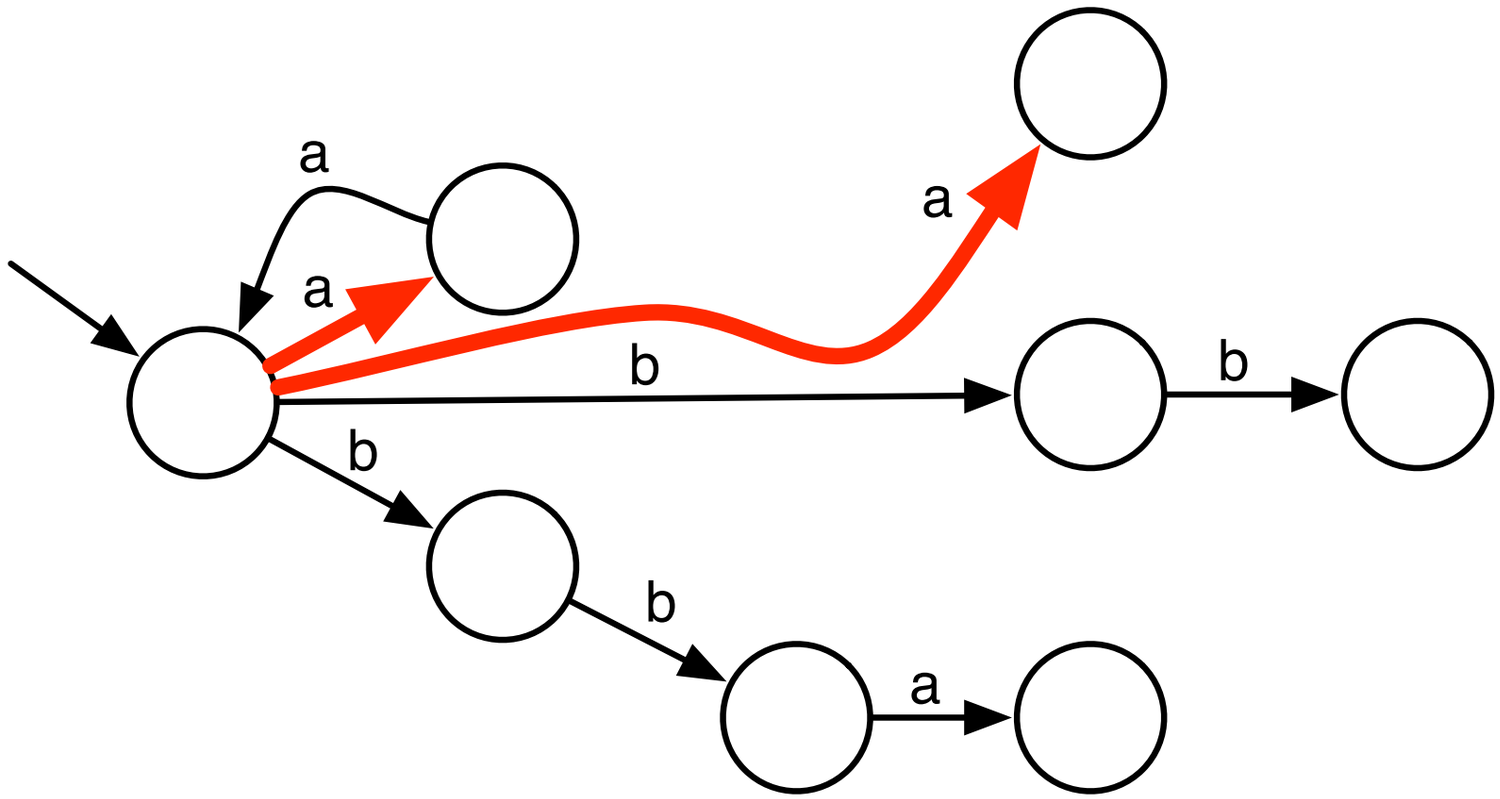
Learning DFAs



State merging:
delete the obsolete state

TU Delft

Learning DFAs

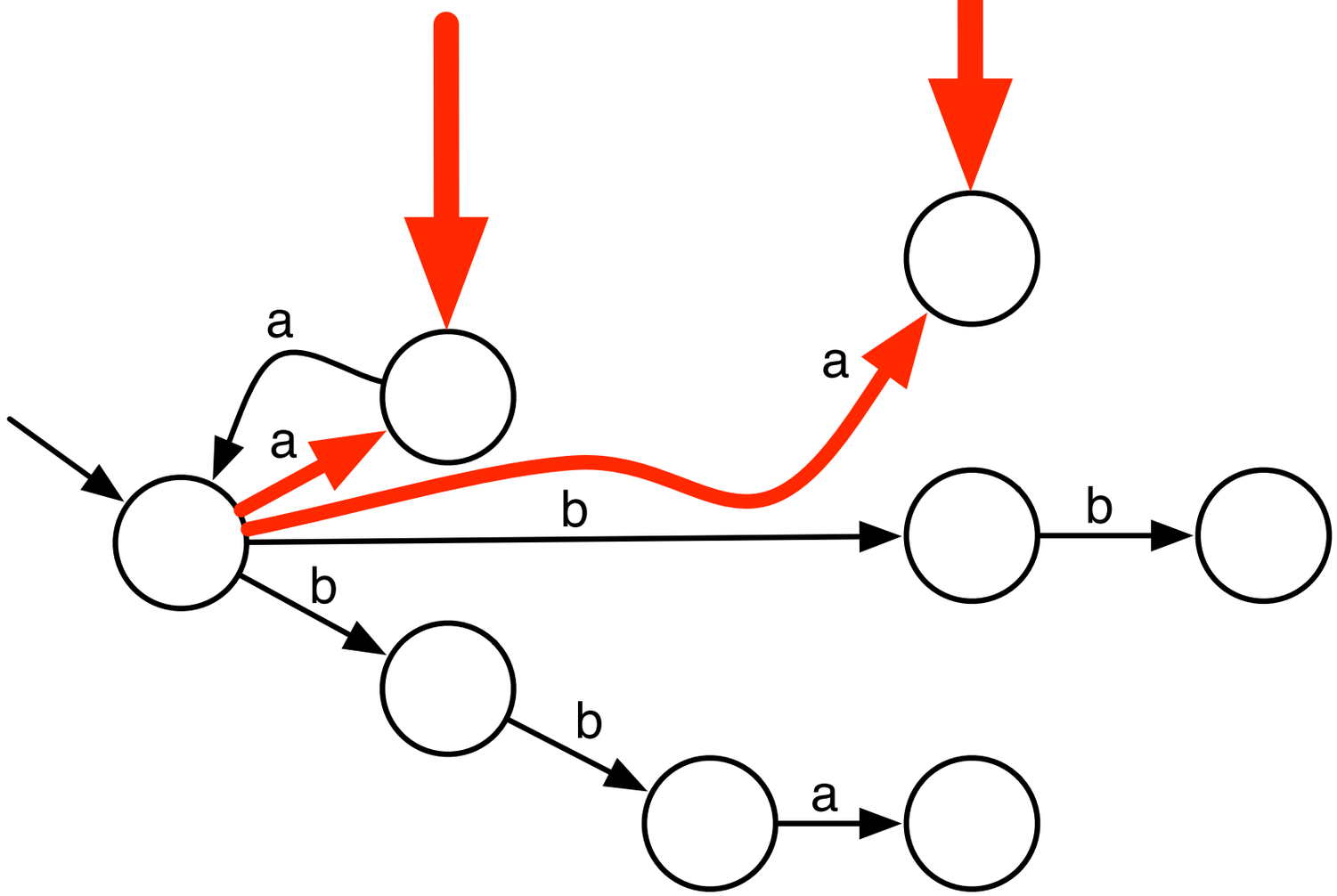


Determinization:

merge the targets of non-deterministic transitions

TU Delft

Learning DFAs

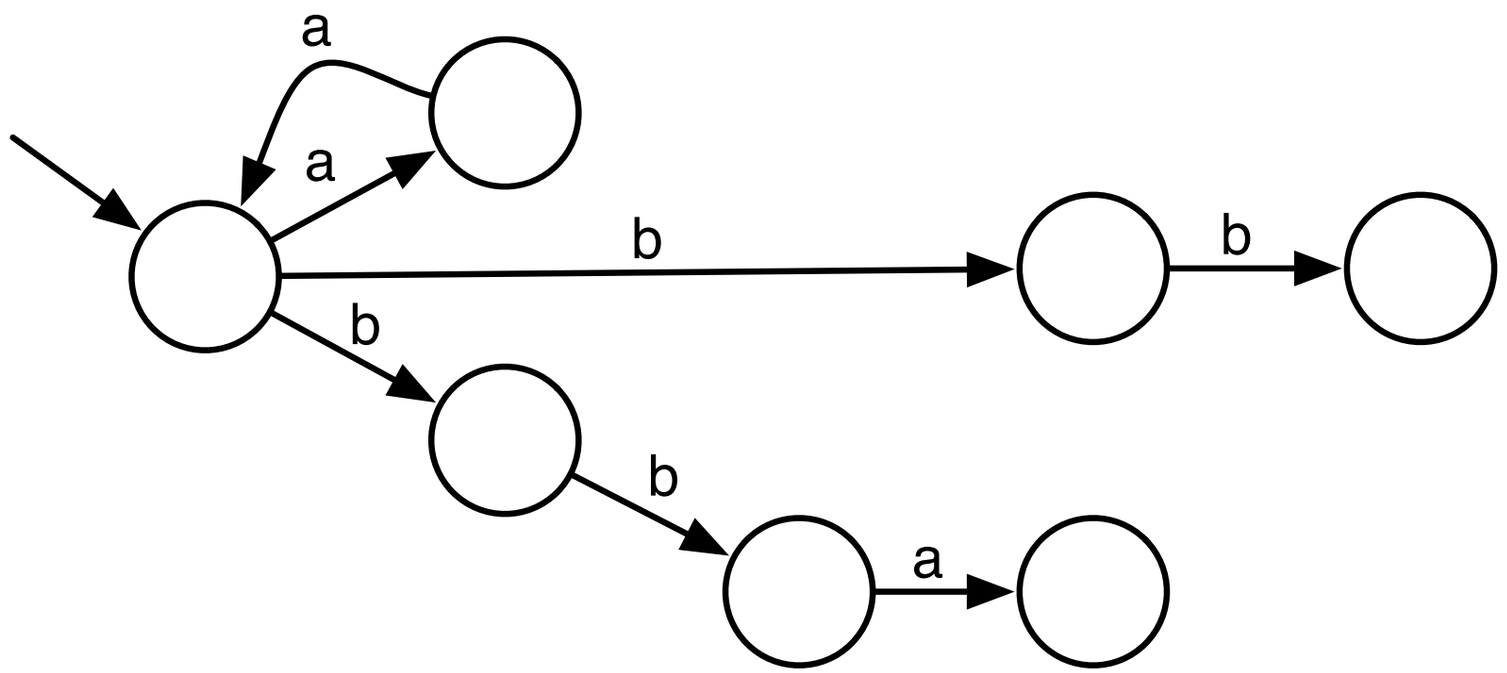


Determinization:

merge the targets of non-deterministic transitions

TU Delft

Learning DFAs

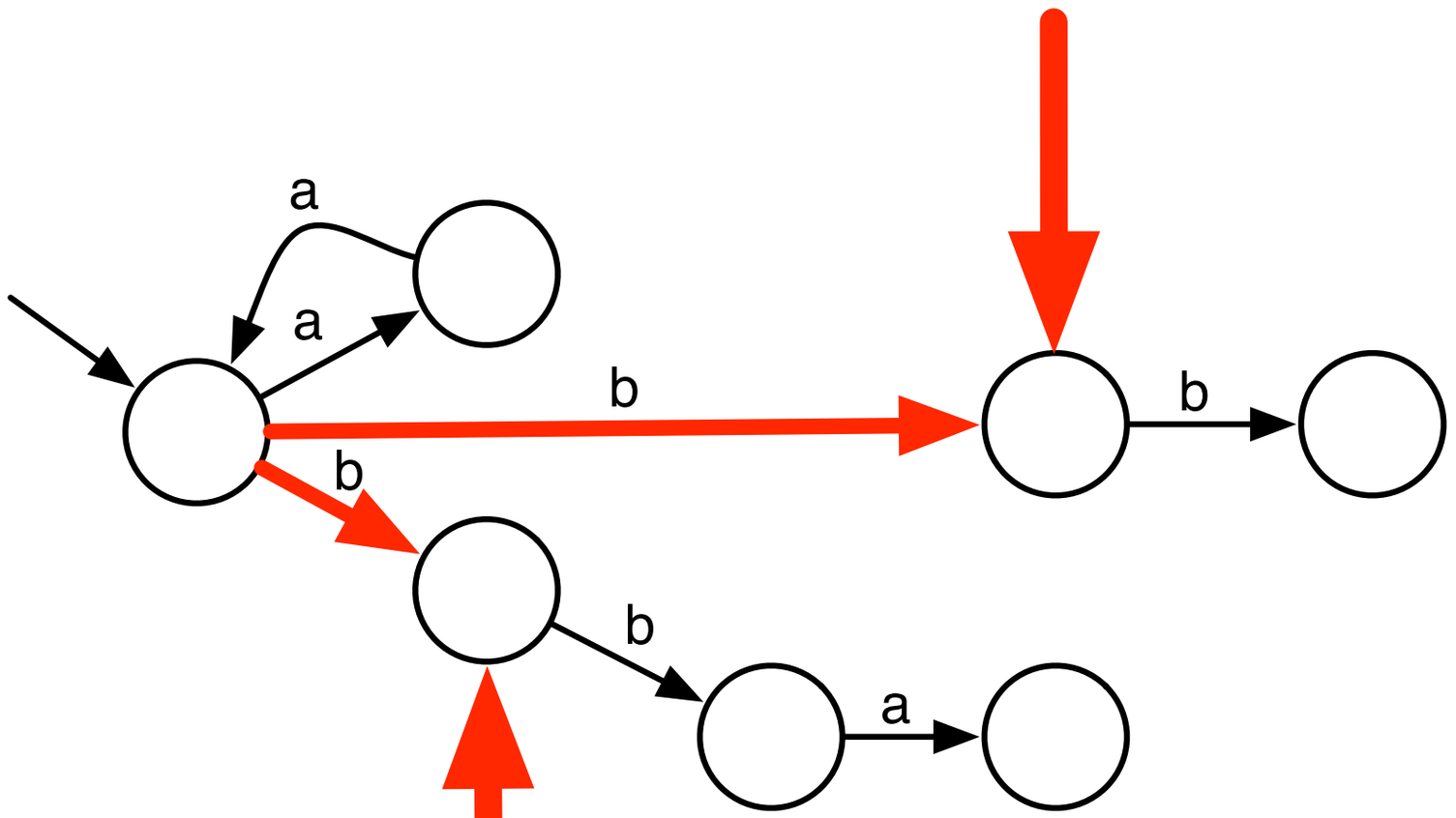


Determinization:

merge the targets of non-deterministic transitions

TU Delft

Learning DFAs

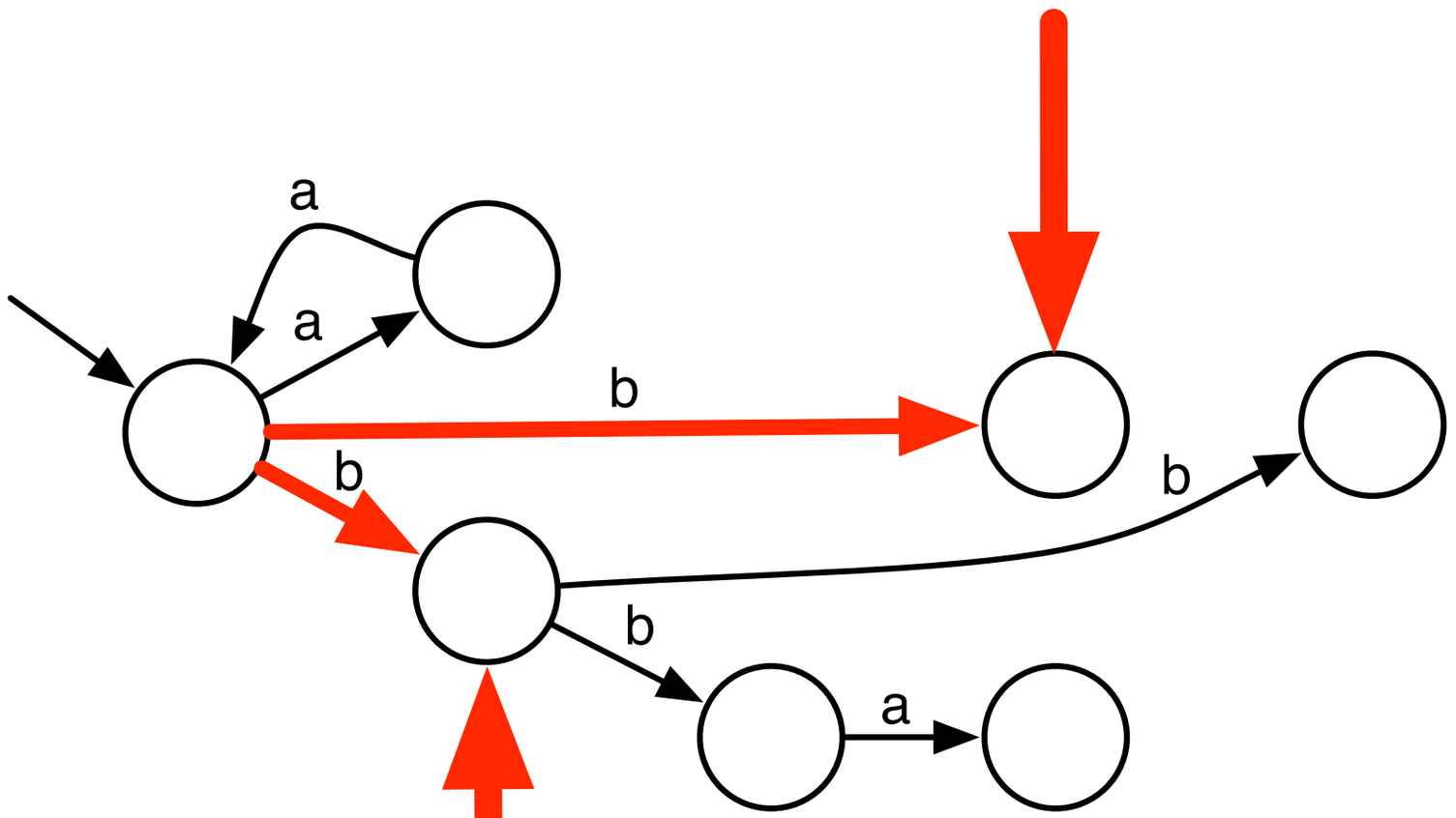


Determinization:

merge the targets of non-deterministic transitions

TU Delft

Learning DFAs

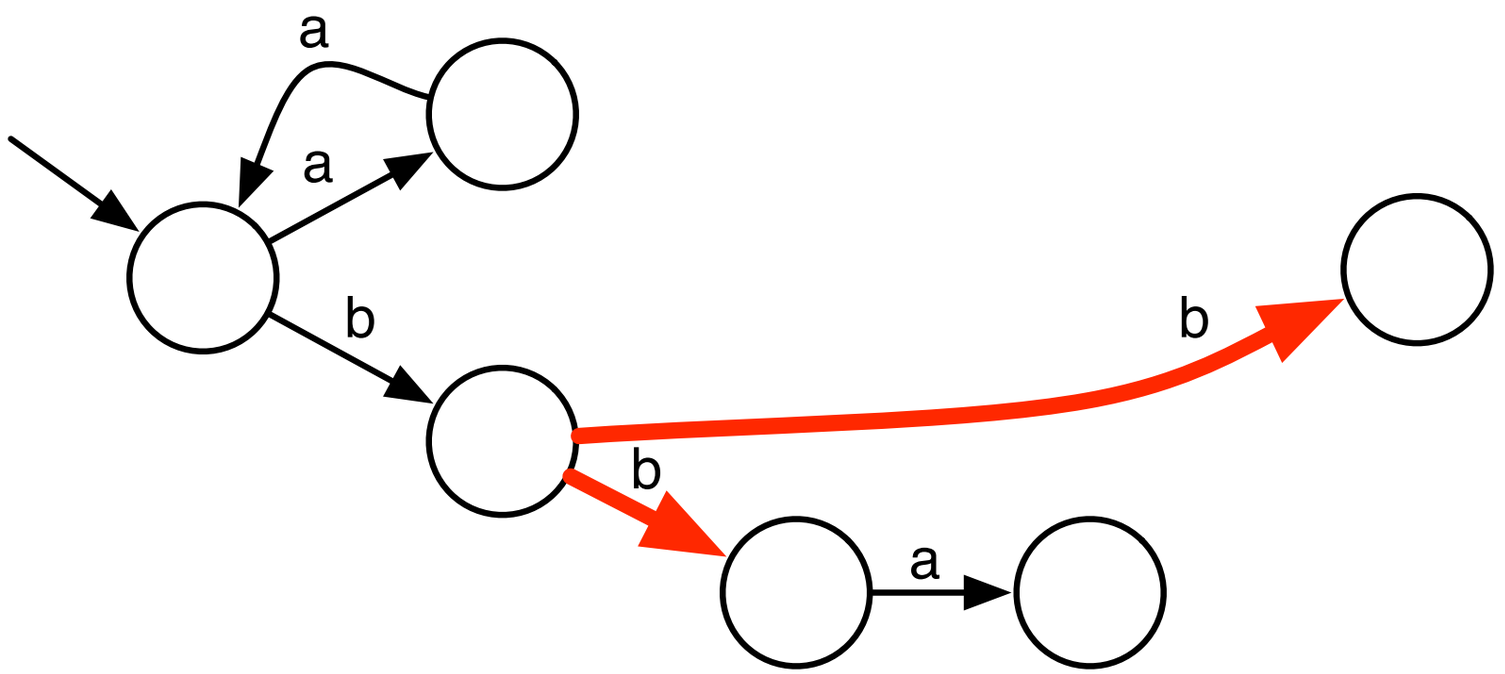


Determinization:

merge the targets of non-deterministic transitions

TU Delft

Learning DFAs

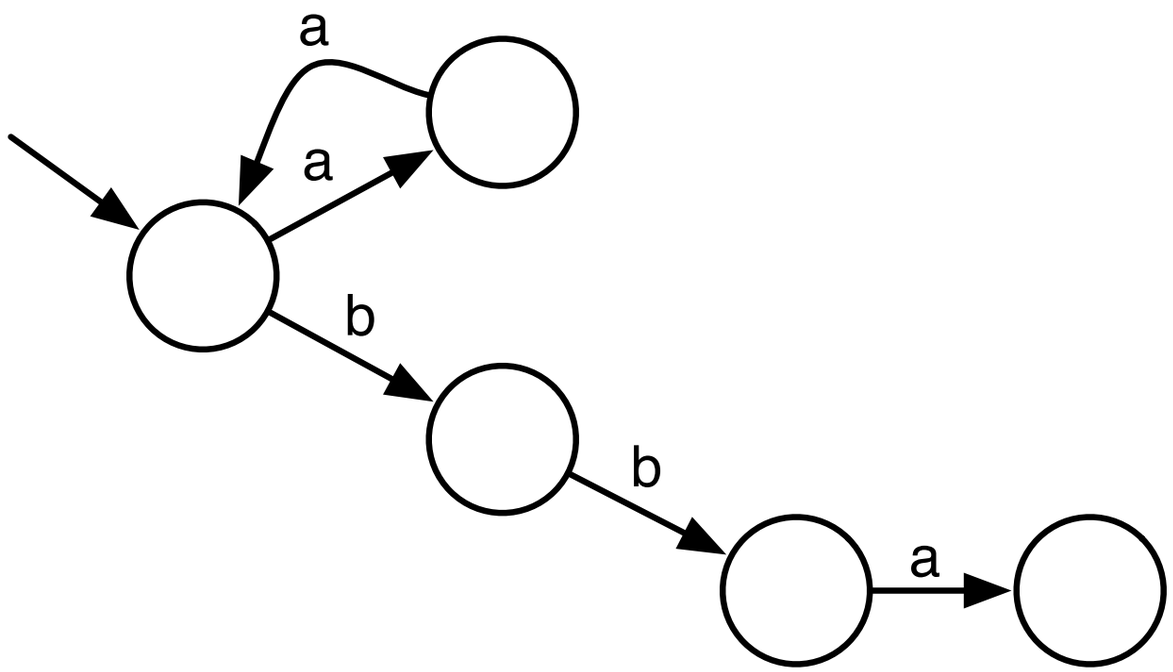


Determinization:

merge the targets of non-deterministic transitions

TU Delft

Learning DFAs



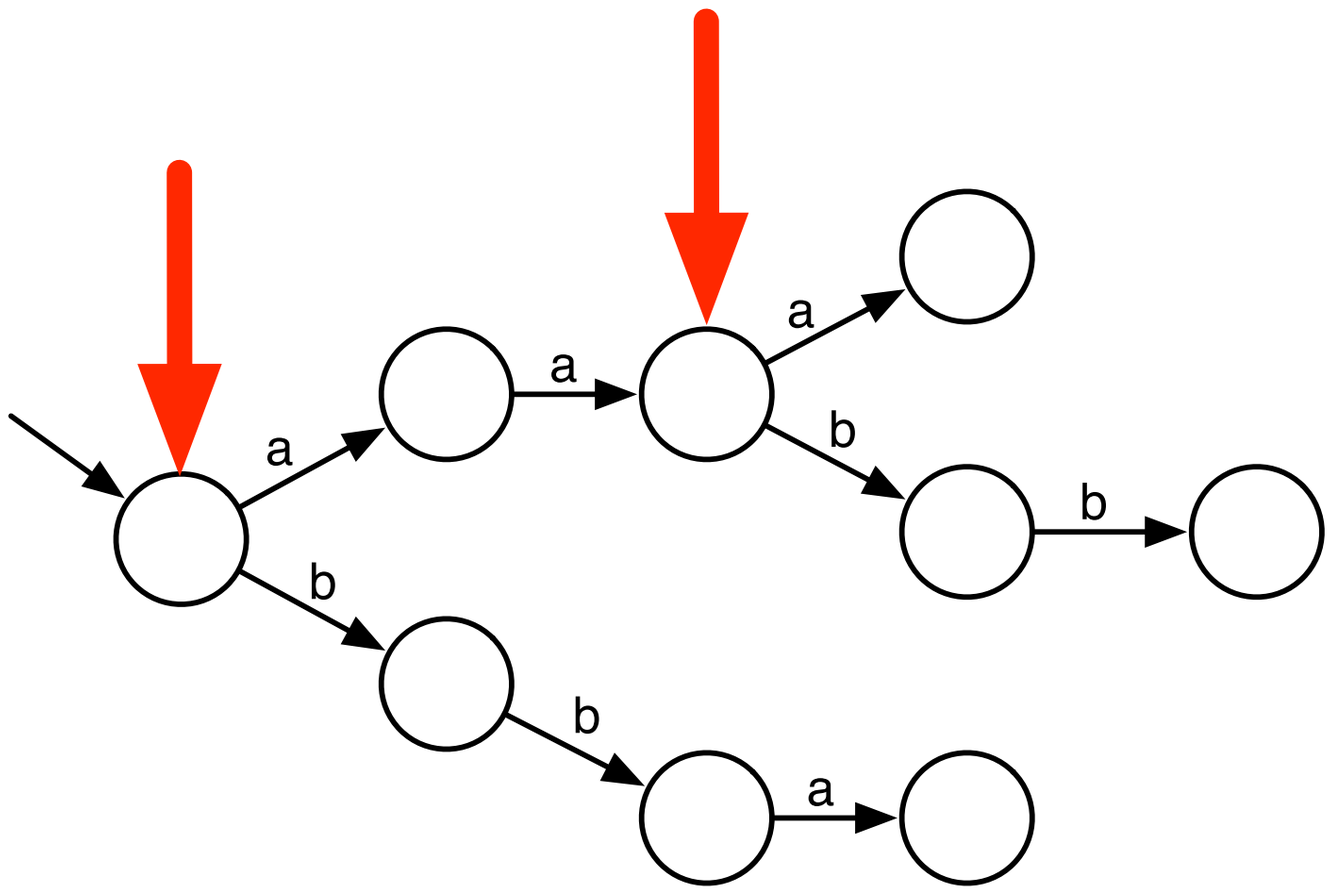
Select two new nodes to merge and iterate

TU Delft

Learning DFAs

- State merging:
  - Start from a tree
  - Try all possible merges, including determinization
  - Perform the one that scores best
  - Iterate

Learning DFAs



The score of a merge can be determined using
labels or statistics

TU Delft

EDSM

- For every string s it is known whether s is an element of the language or not, it is positive or negative
- Evidence driven state merging (EDSM):
  - Initially, the states *q* of the prefix tree are labeled according to positive/negative strings that end in *q*
  - It is not possible to merge positively labeled states with negatively labeled states
  - Score = #positive merges + # negative merges

ALERGIA

- It is not known whether strings are in the language or not
- ALERGIA:
  - Use a norm or statistic, like L_∞ or chi squared
  - Define a bound *b*, for the similarity between states
  - It is not possible to merge states for which the norm or statistical dissimilarity is greater than *b*
  - Score = value of norm or statistical difference

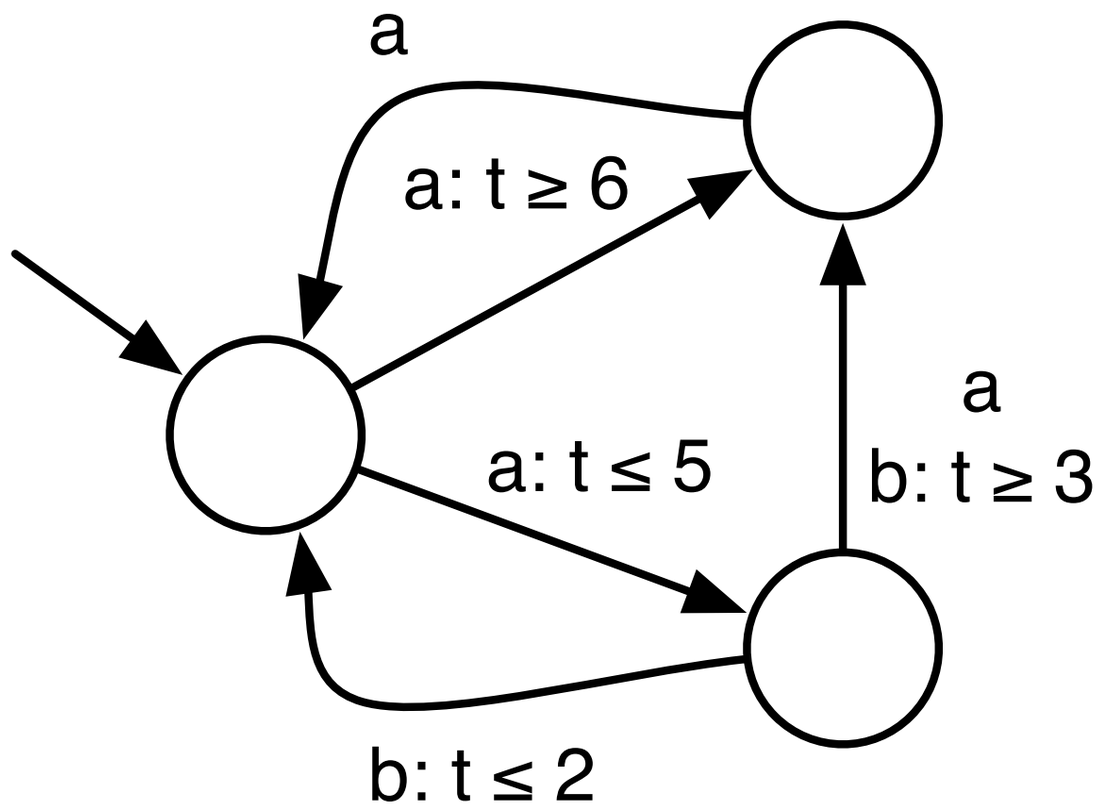
Algorithms for learning DFA

- State merging using EDSM performs best for labeled data
- The idea in ALERGIA has been used in many other algorithms, also in PAC learning DFA distributions
- Both algorithms converge in the limit to the correct DFA
- For both algorithms it is possible to compute the amount of data necessary to converge with sufficient probability

Overview

- Automata & how to learn them
- Real-time automata
- Learning real-time automata from labeled data
  - Results
- Learning real-time automata from unlabeled data
  - Results
- Conclusions

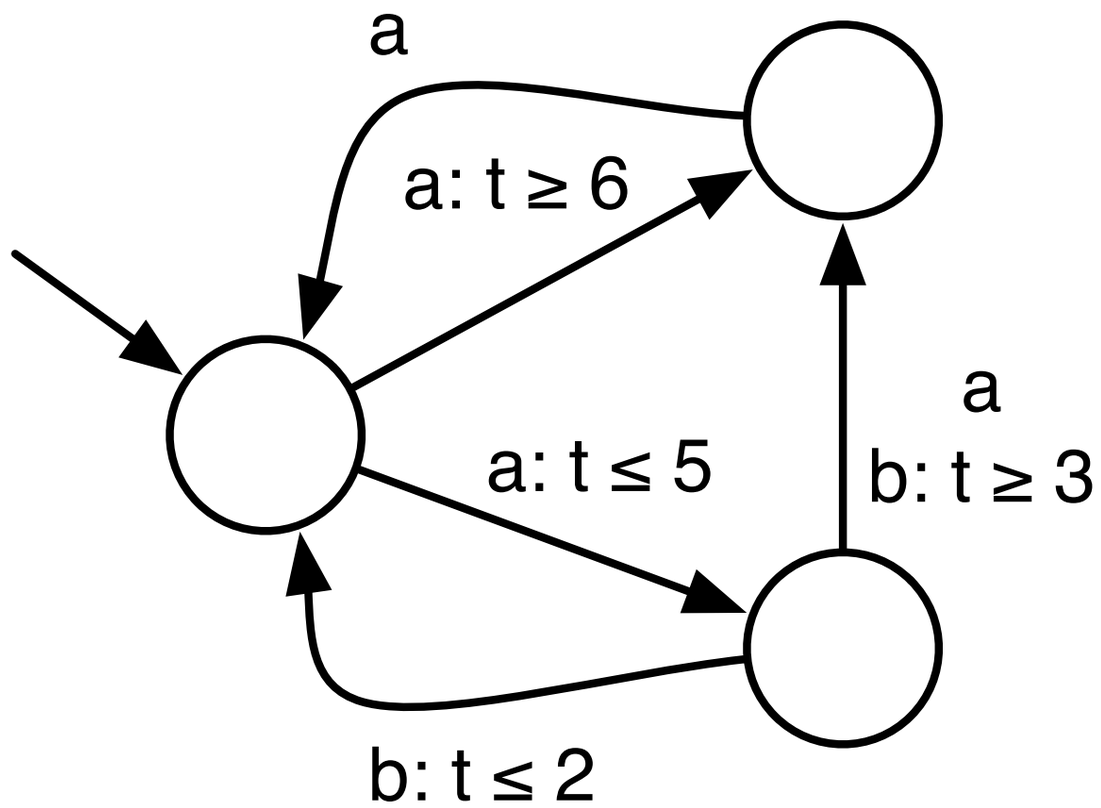
Real-time automata (DRTAs)



Transitions contain guards on time values

TU Delft

DRTAs and events

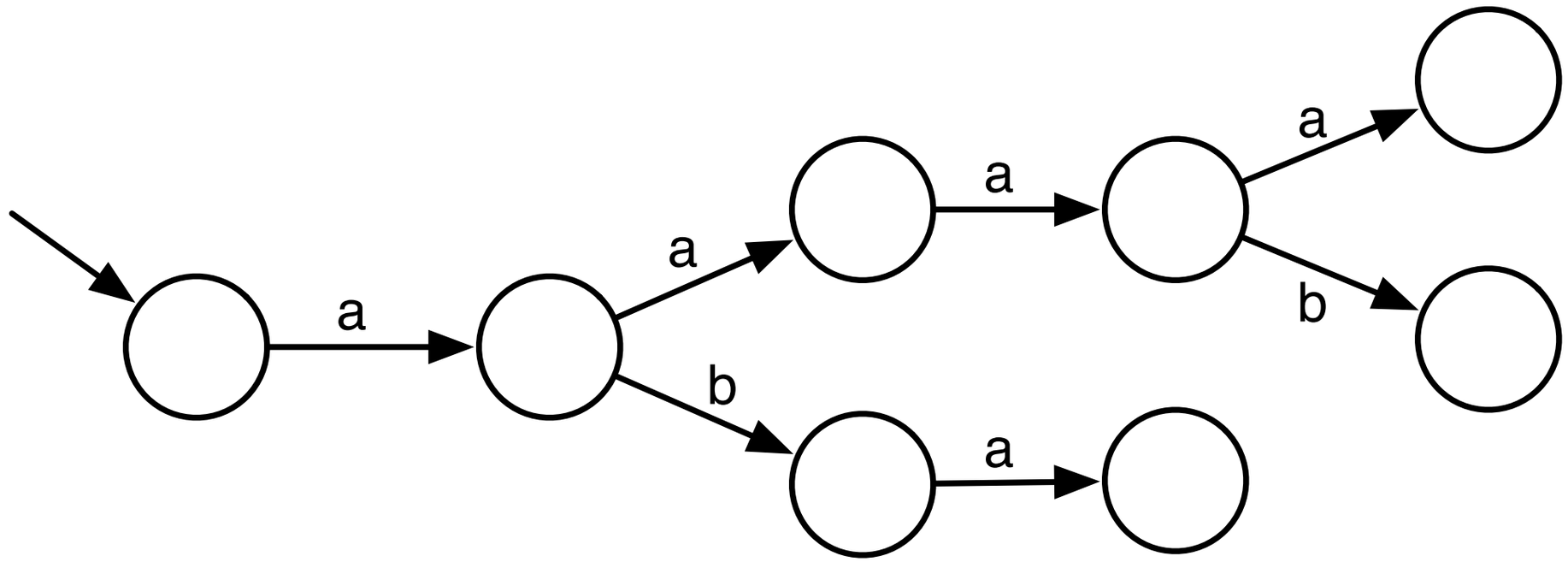


Produces timed strings:

(a,6)(a,2)(a,3); (a,2)(b,1)(a,1)(b,8)

TU Delft

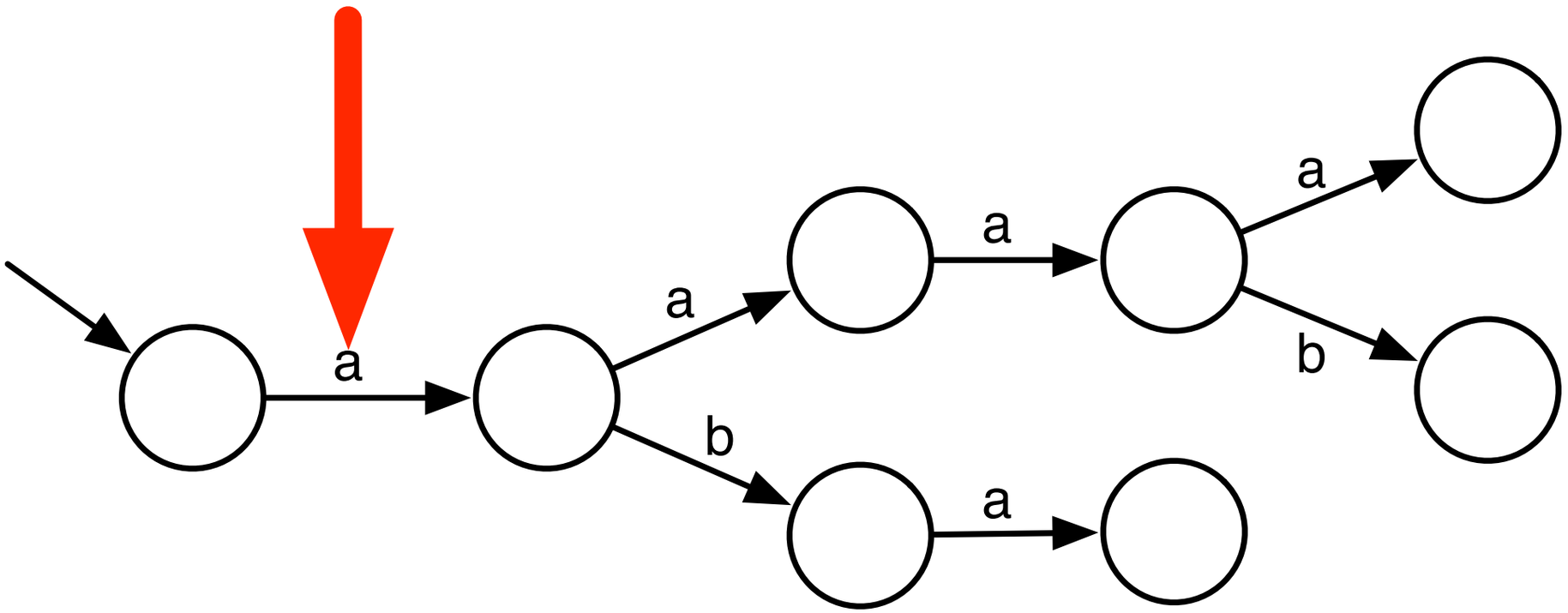
Learning DRTAs



A prefix tree
all guards are set to true, [0, ∞)

TU Delft

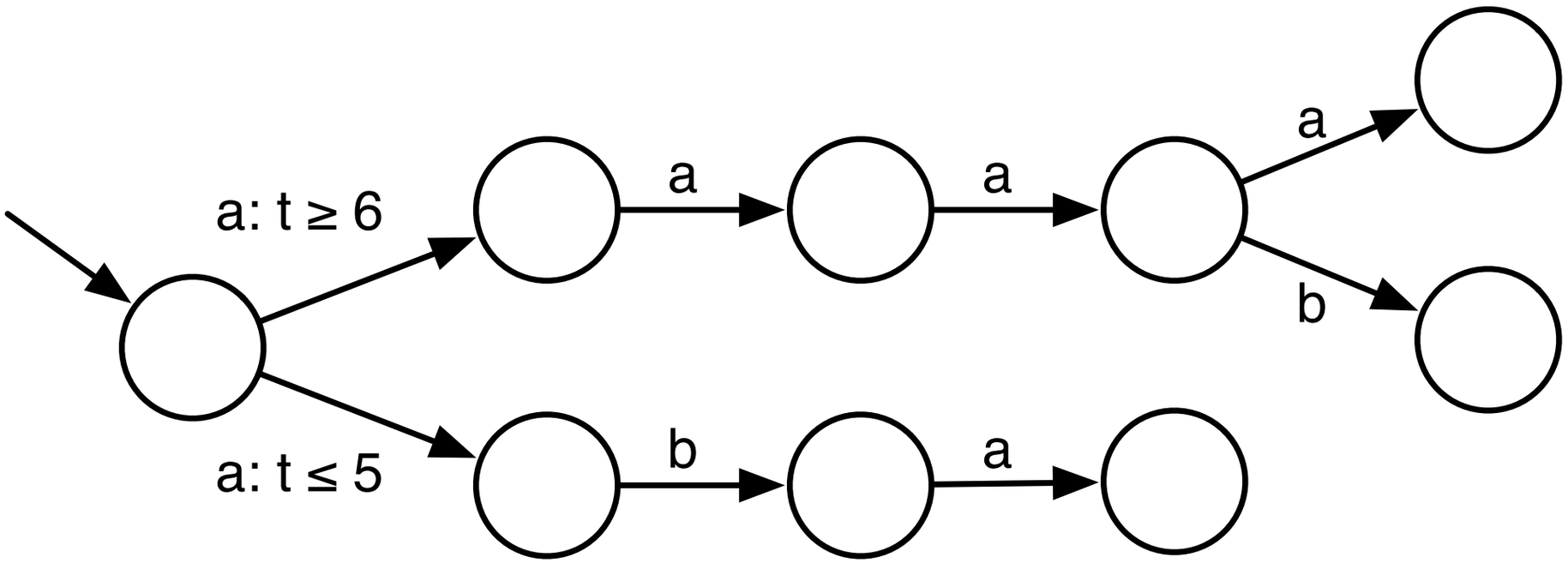
Learning DRTAs



Transition splitting:
choose a transition

TU Delft

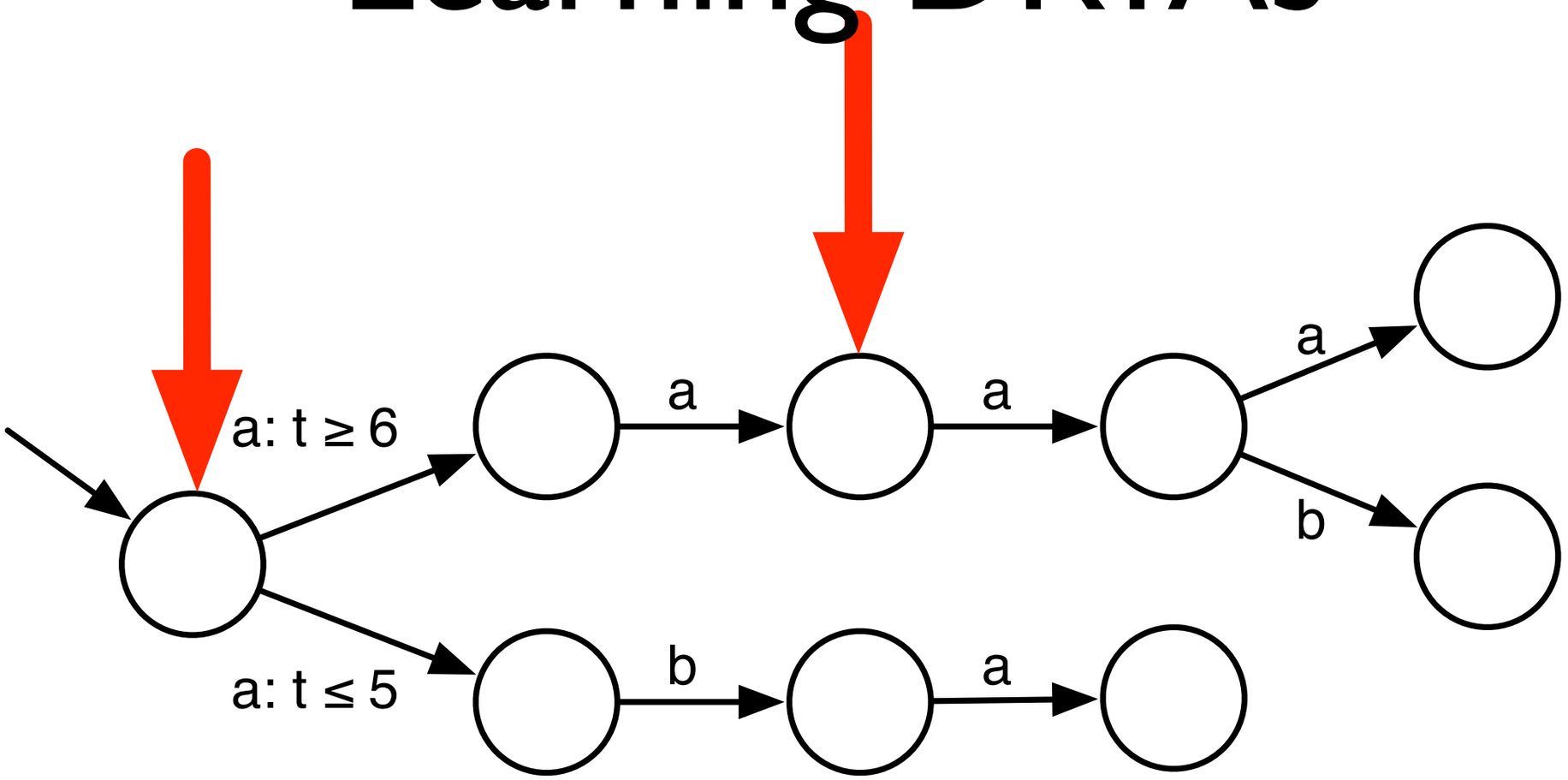
Learning DRTAs



Split the transition and recalculate the subsequent part of the prefix tree

TU Delft

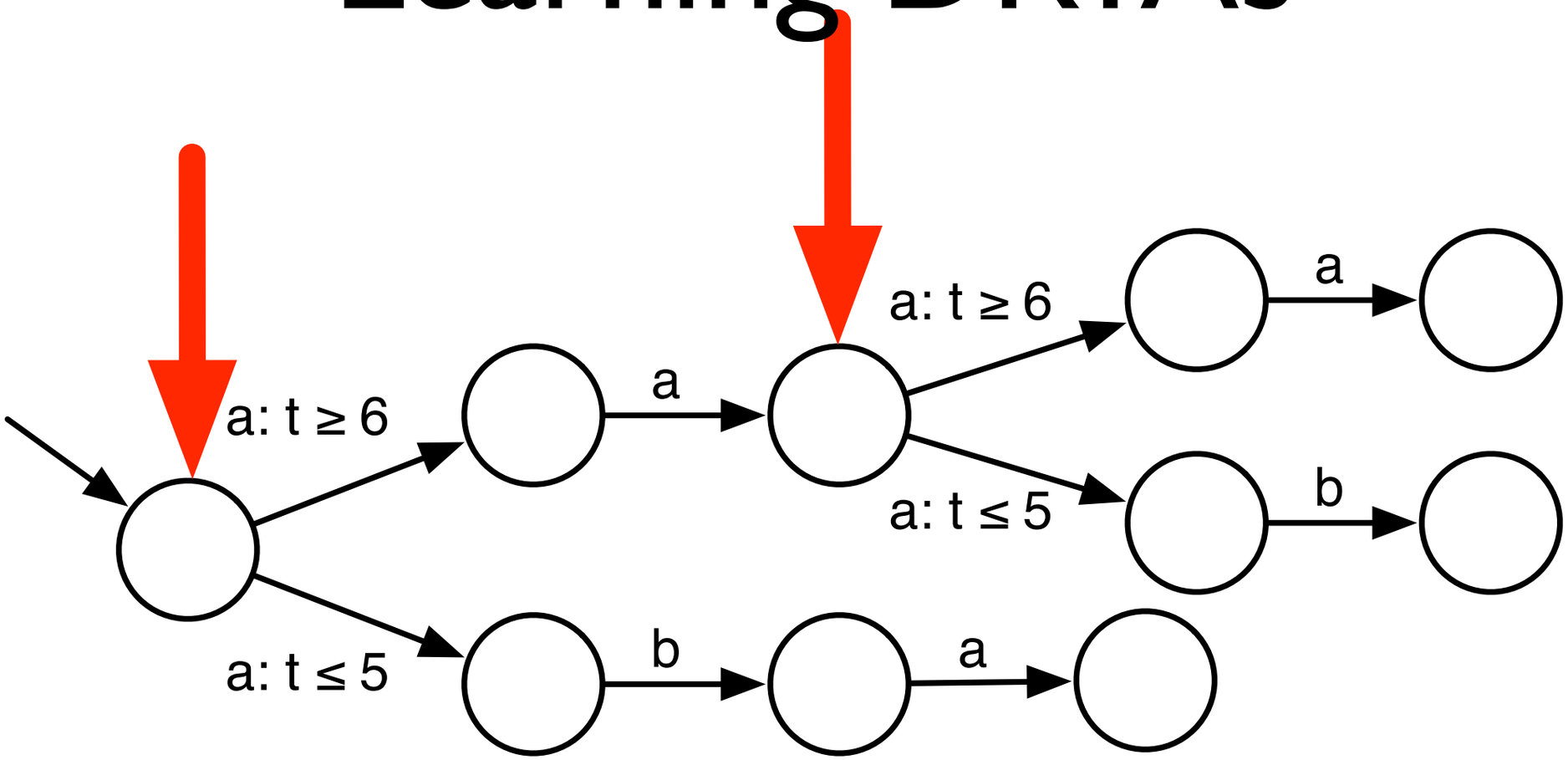
Learning DRTAs



Later, when merging states

TU Delft

Learning DRTAs



First split the transitions such that the guards match

TU Delft

Learning DRTAs

- State merging and transition splitting:
  - Start from a tree
  - Try all possible merges and splits
  - If one scores good:
    - Perform the one that scores best
  - Else
    - break
  - Iterate

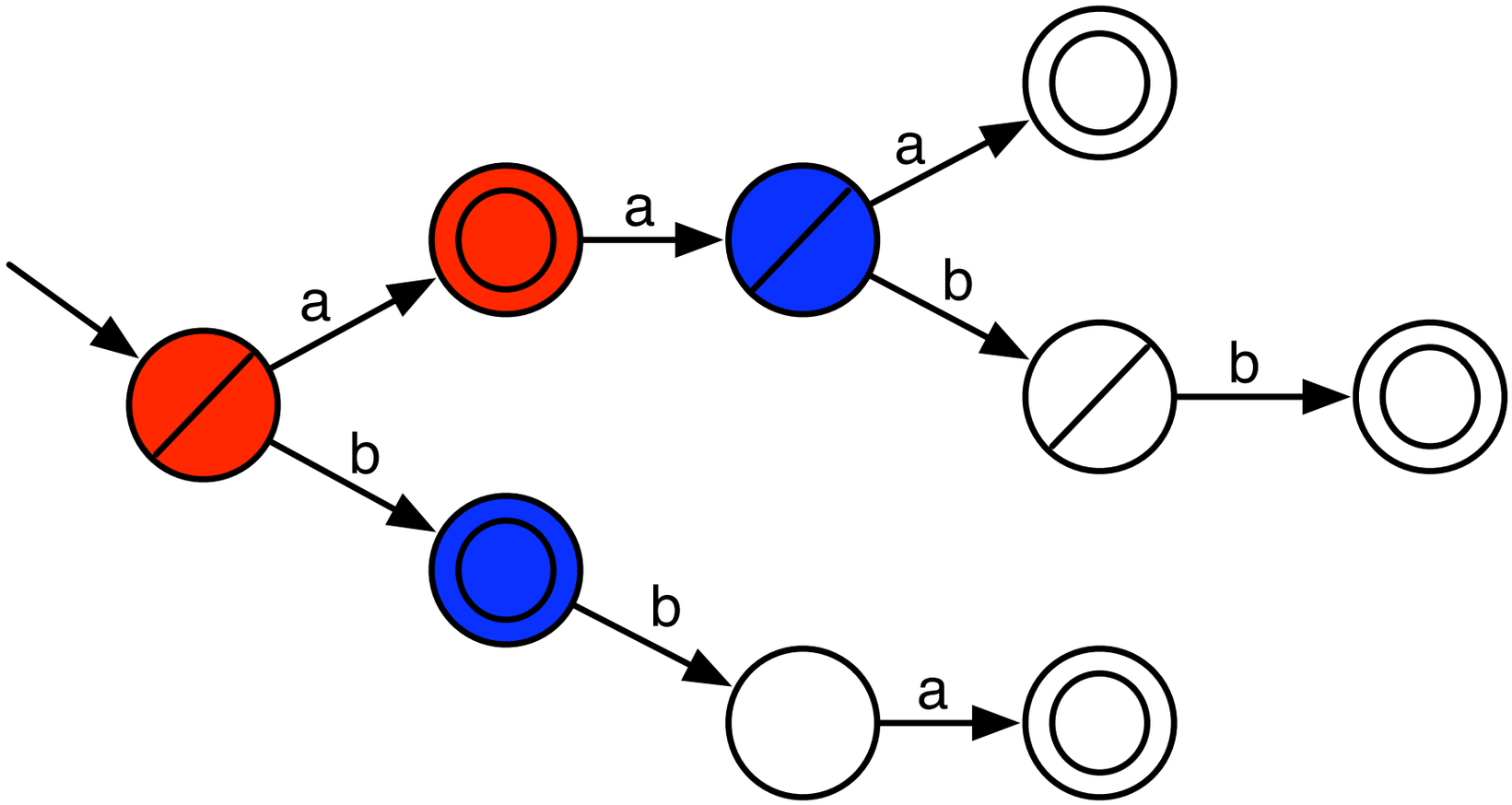
Overview

- Automata & how to learn them
- Real-time automata
- Learning real-time automata from labeled data
  - Results
- Learning real-time automata from unlabeled data
  - Results
- Conclusions

Labeled data

- Input: two sets S_+ and S_- of positive and negative timed strings
- Use the state merging and transition splitting algorithm, but
  - Allow merges between positive and negative states
  - Maintain a core of inferred states using the red-blue framework
  - Do not allow positive-negative merges in the core

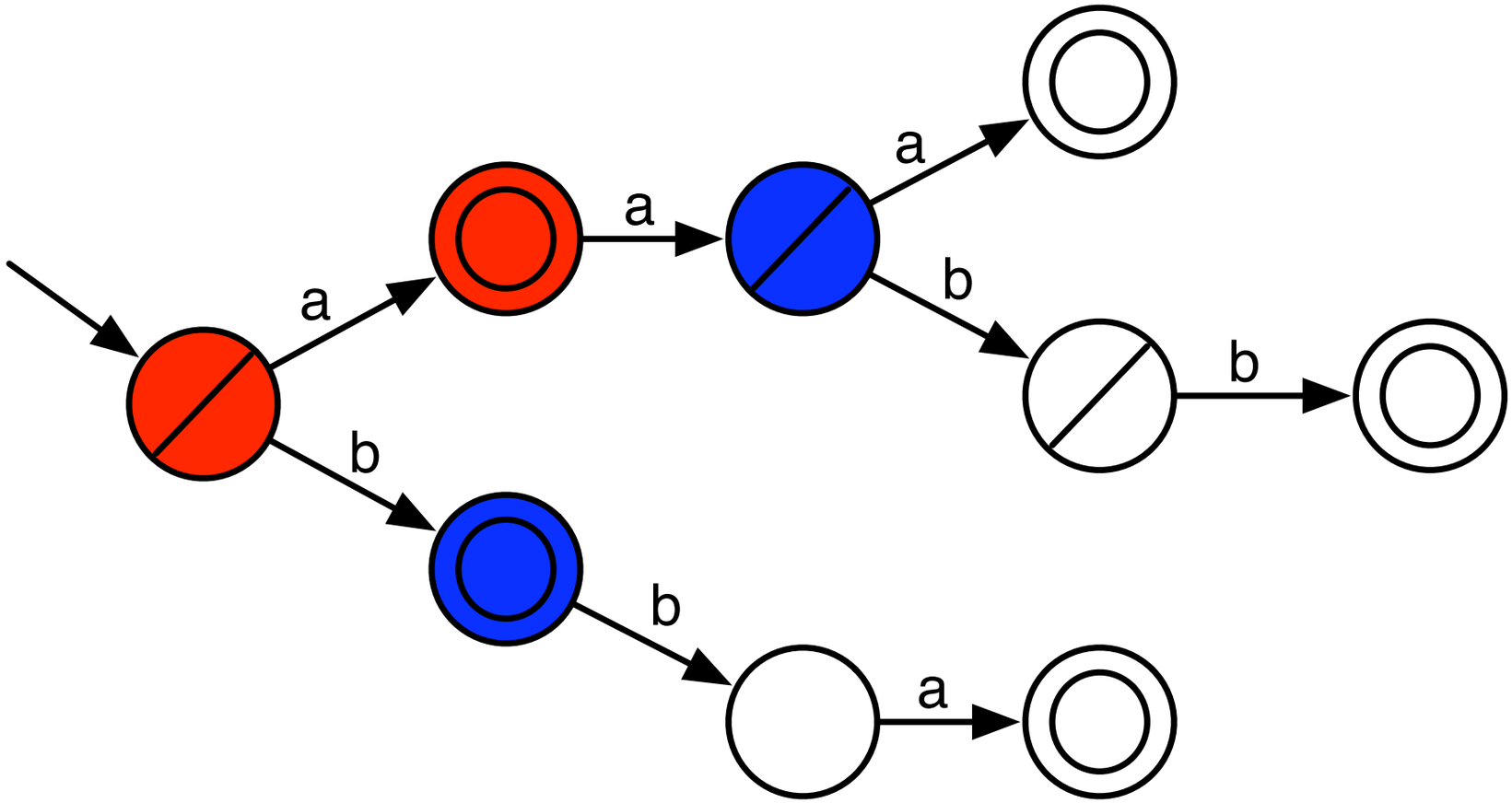
Labeled data



A core or red states, with a fringe of blue states
only allow merges or red and blue states
only split transitions to blue states

TU Delft

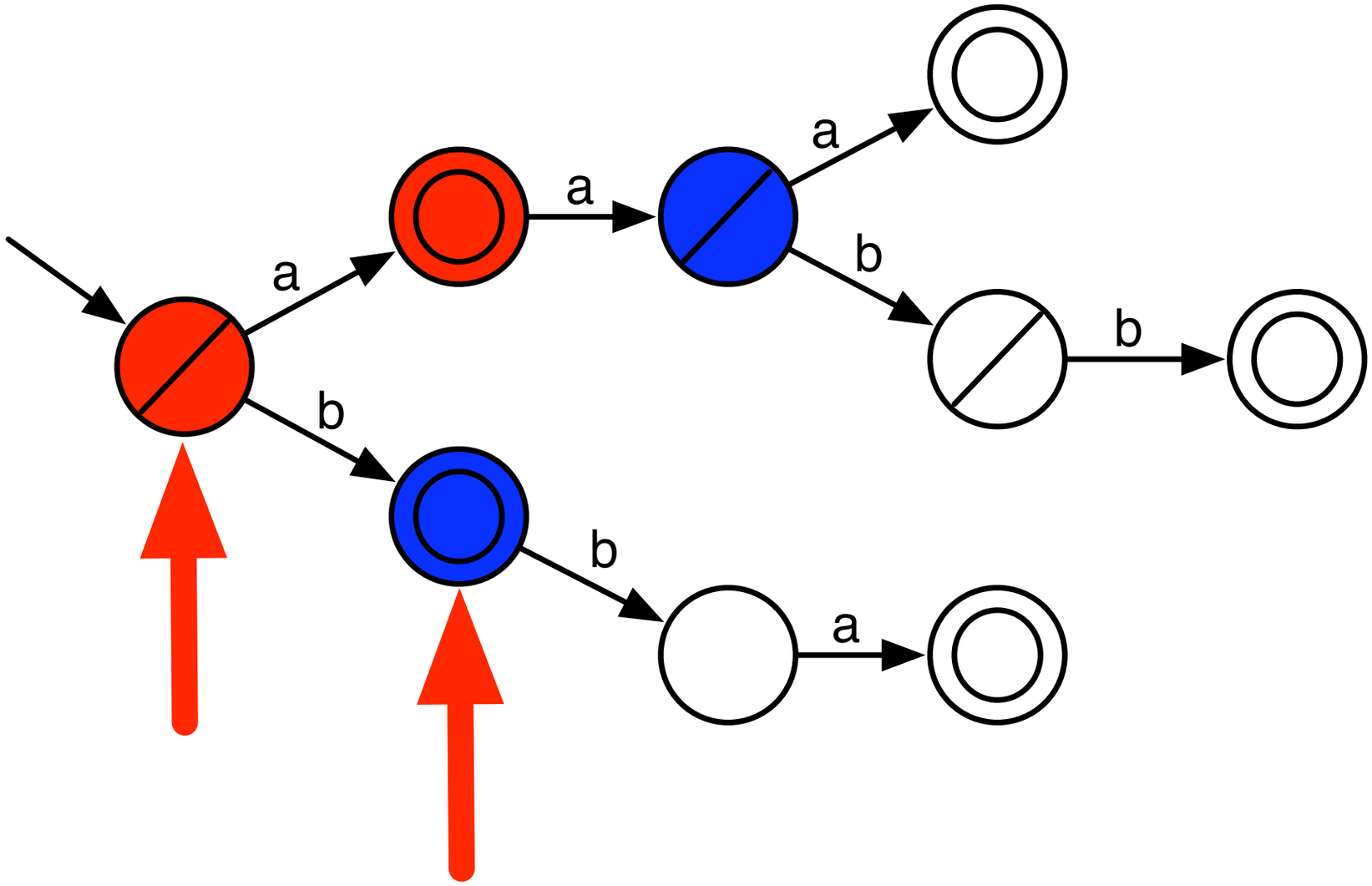
Labeled data



When a blue state cannot be merged or split,
this blue state is colored red

TU Delft

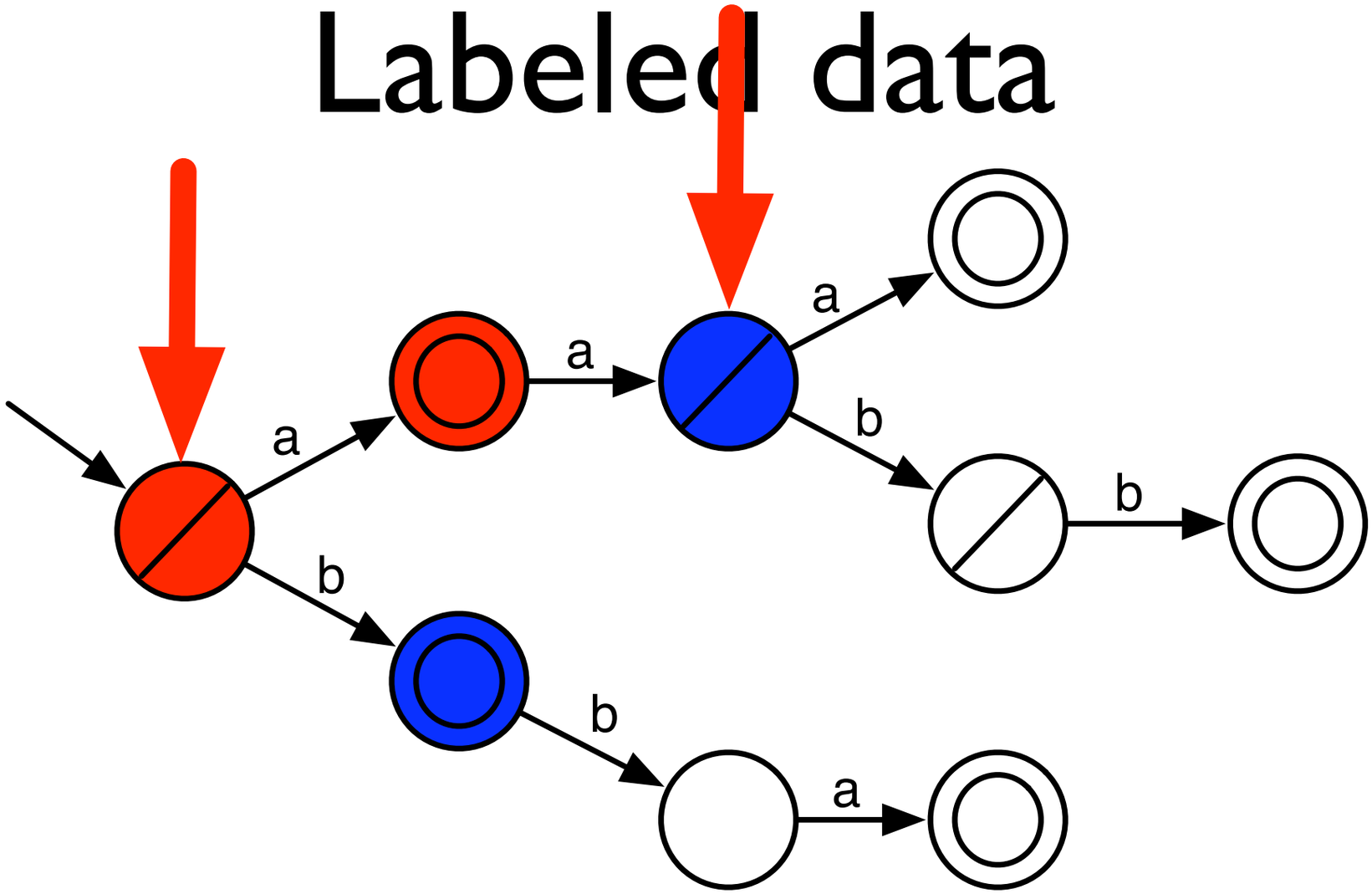
Labeled data



Results in a positive-negative merge in the core

TU Delft

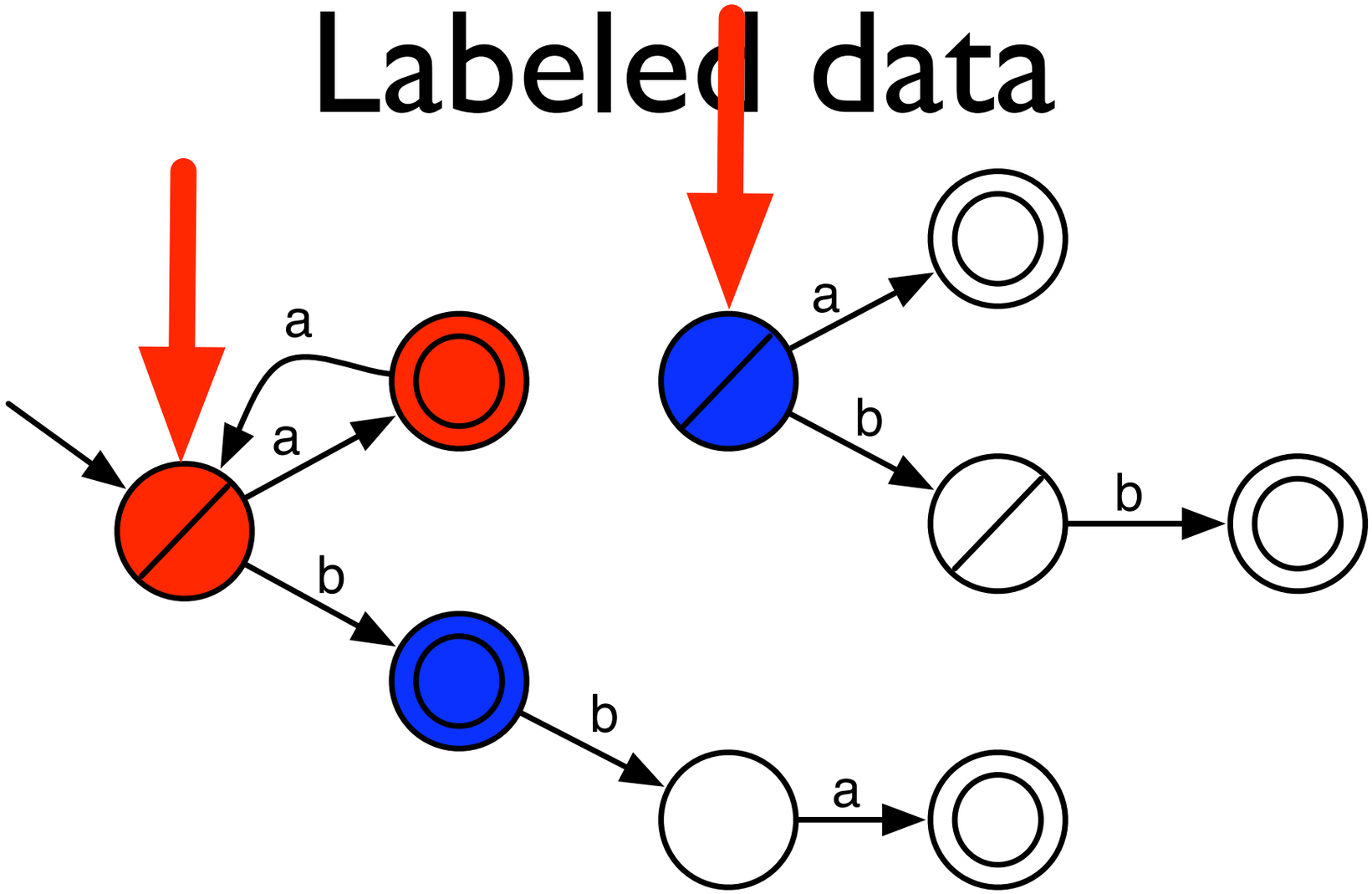
Labeled data



Does not results in a positive-negative merge in the core

TU Delft

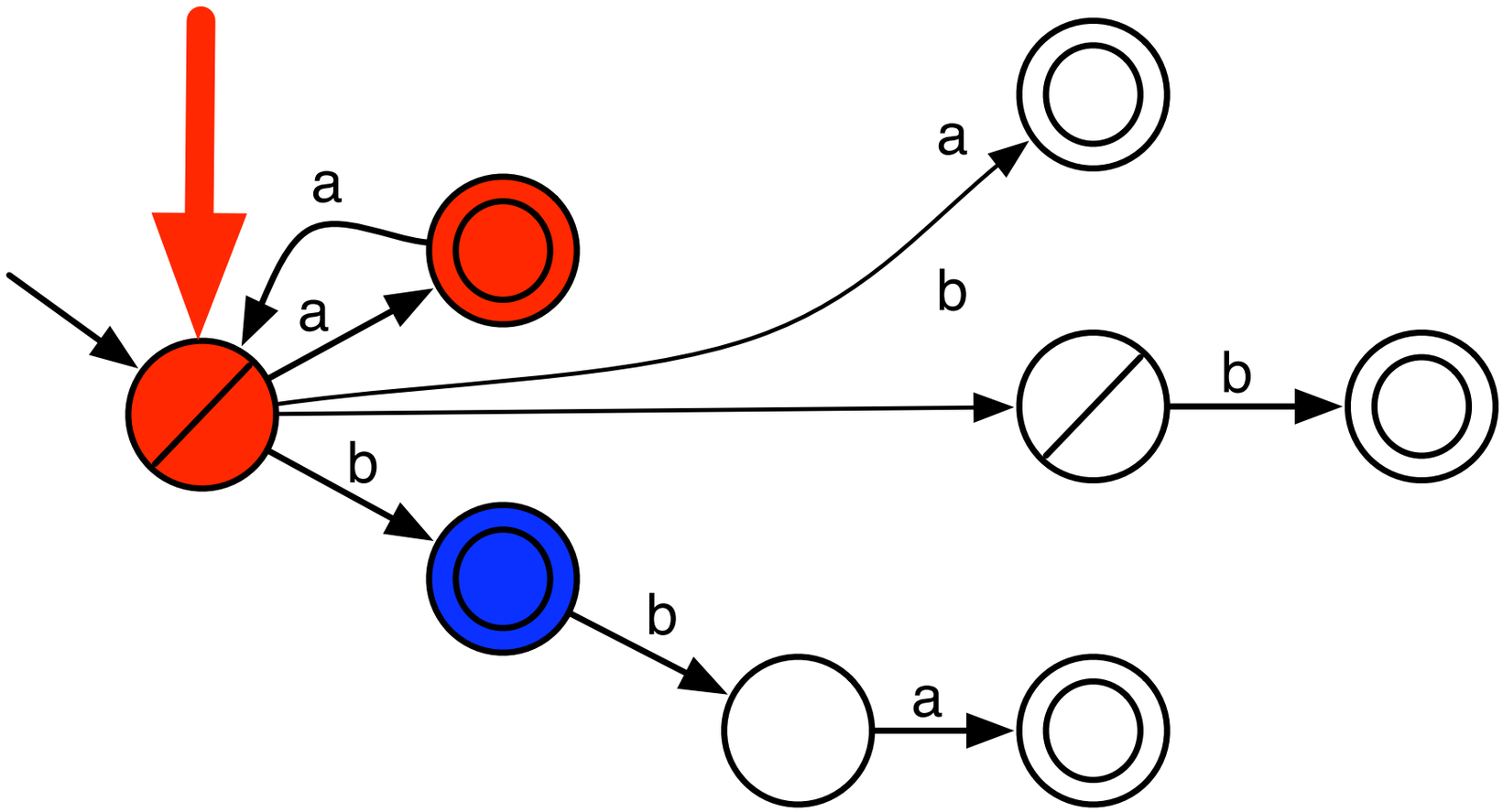
Labeled data



Does not results in a positive-negative merge in the core

TU Delft

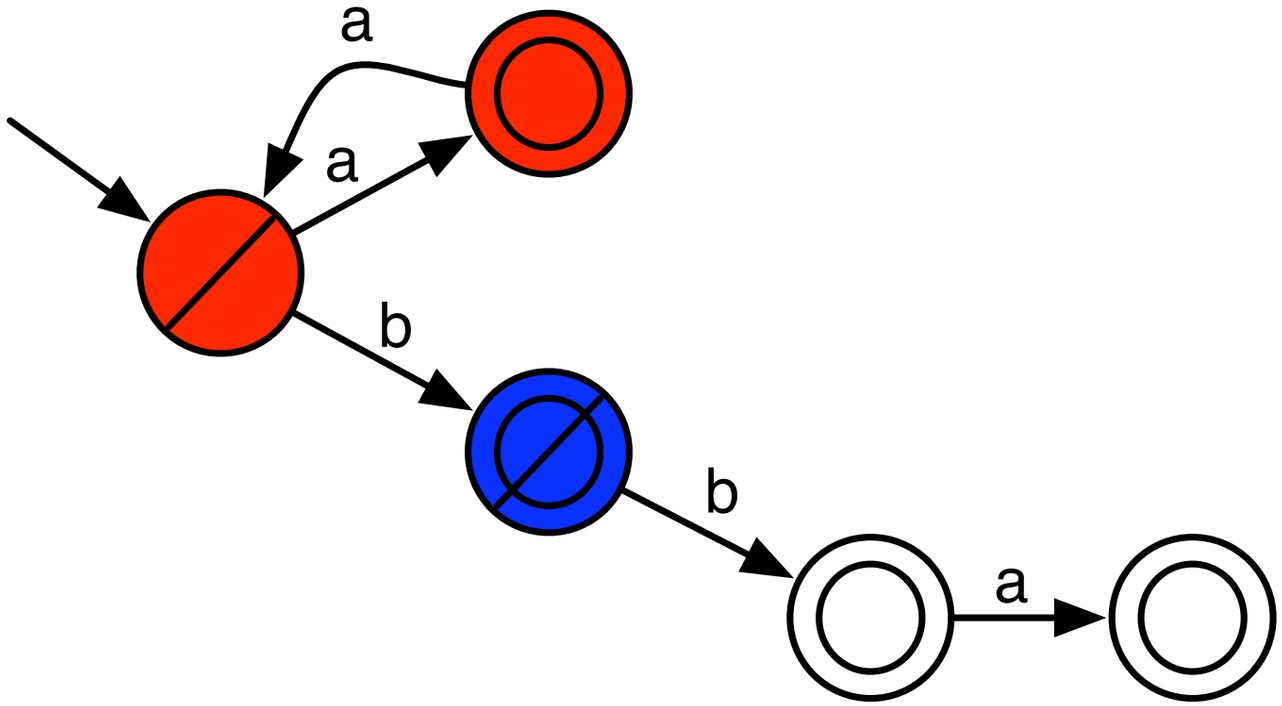
Labeled data



Does not results in a positive-negative merge in the core

TU Delft

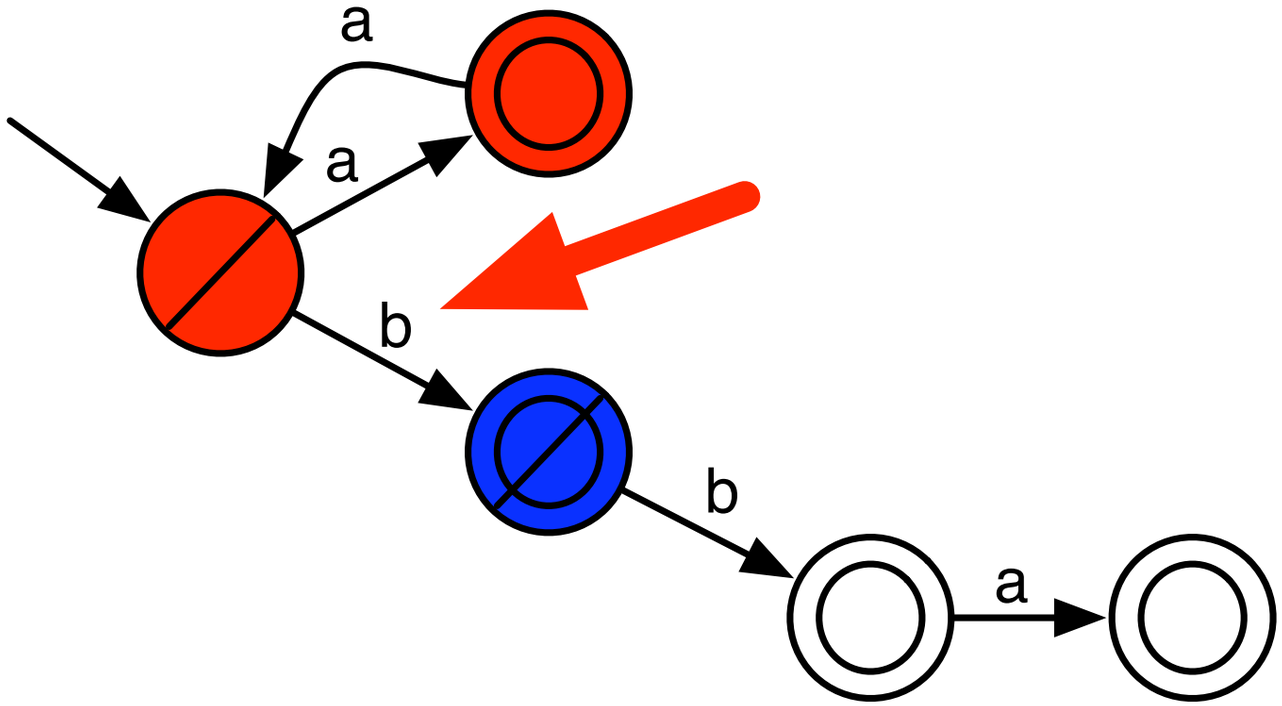
Labeled data



Does not results in a positive-negative merge in the core

TU Delft

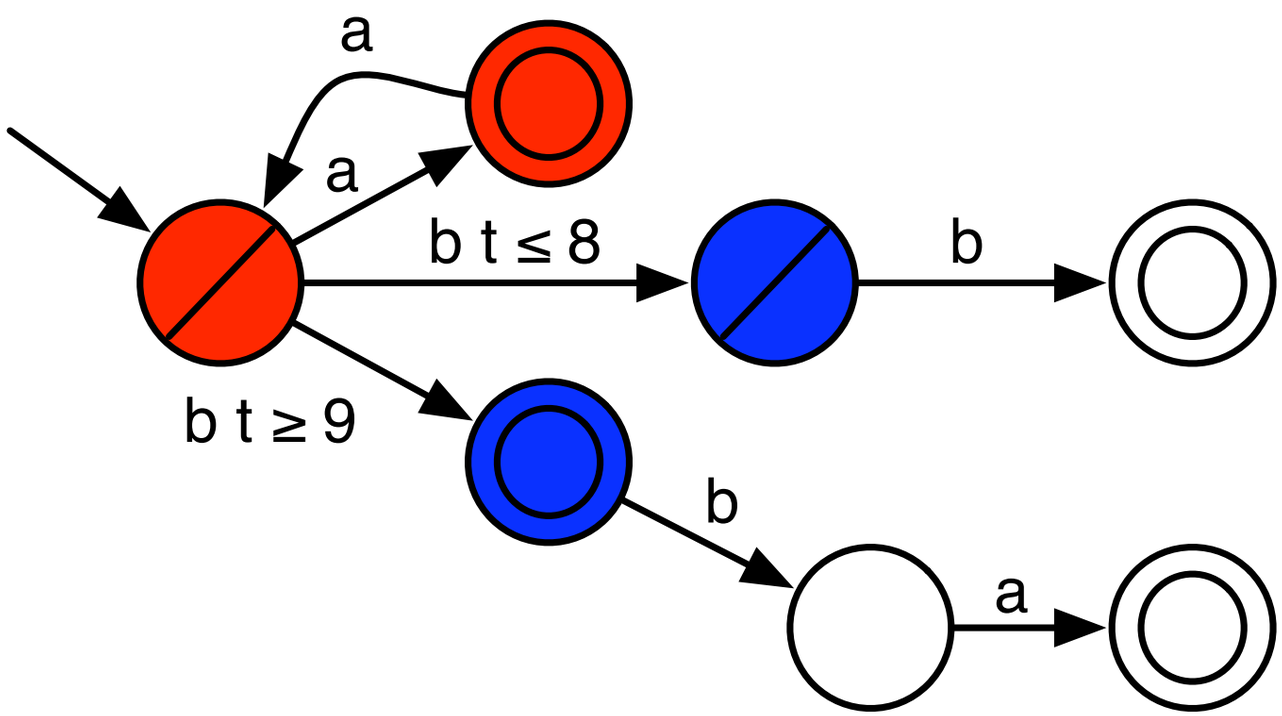
Labeled data



The positive-negative merge can be resolved using a split

TU Delft

Labeled data



The positive-negative merge can be resolved using a split

TU Delft

Labeled data

- Merges/splits can create/remove both consistent (pos-pos, neg-neg) and inconsistent (pos-neg) merges
- An inconsistent merge is only allowed if it can later be resolved using a split
- The inconsistent merges do not occur in the EDSM algorithm for DFAs, but provide useful information
- We adapt EDSM to make use this new information

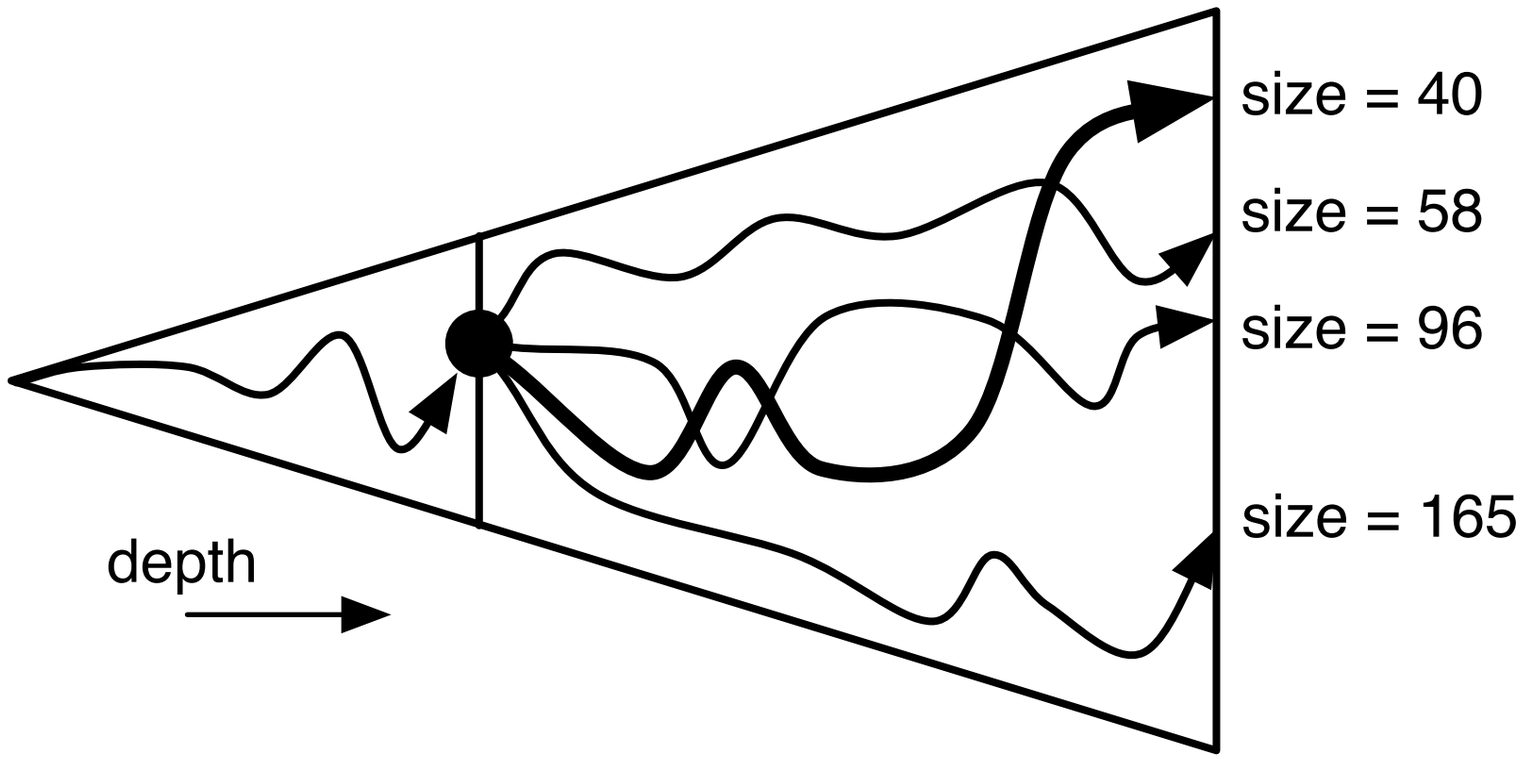
Labeled evidence values

- Consistent EDSM:
  - #consistent merges - # inconsistent merges
- Impact:
  - Weigh each merge by 1.0 - their resolving probability
  - ∑ consistent weights - ∑ inconsistent weights
- Split:
  - #consistent merges - #splits needed to resolve all inconsistent merges (approx.)

Sampled EDSM

- Replace every time value *t* in a timed string by *t*/*r* special time tick symbols, where *r* is a given constant
- Run the original EDSM algorithm on the transformed input, also using the red-blue framework
- The result is a DFA representation of the DRTA

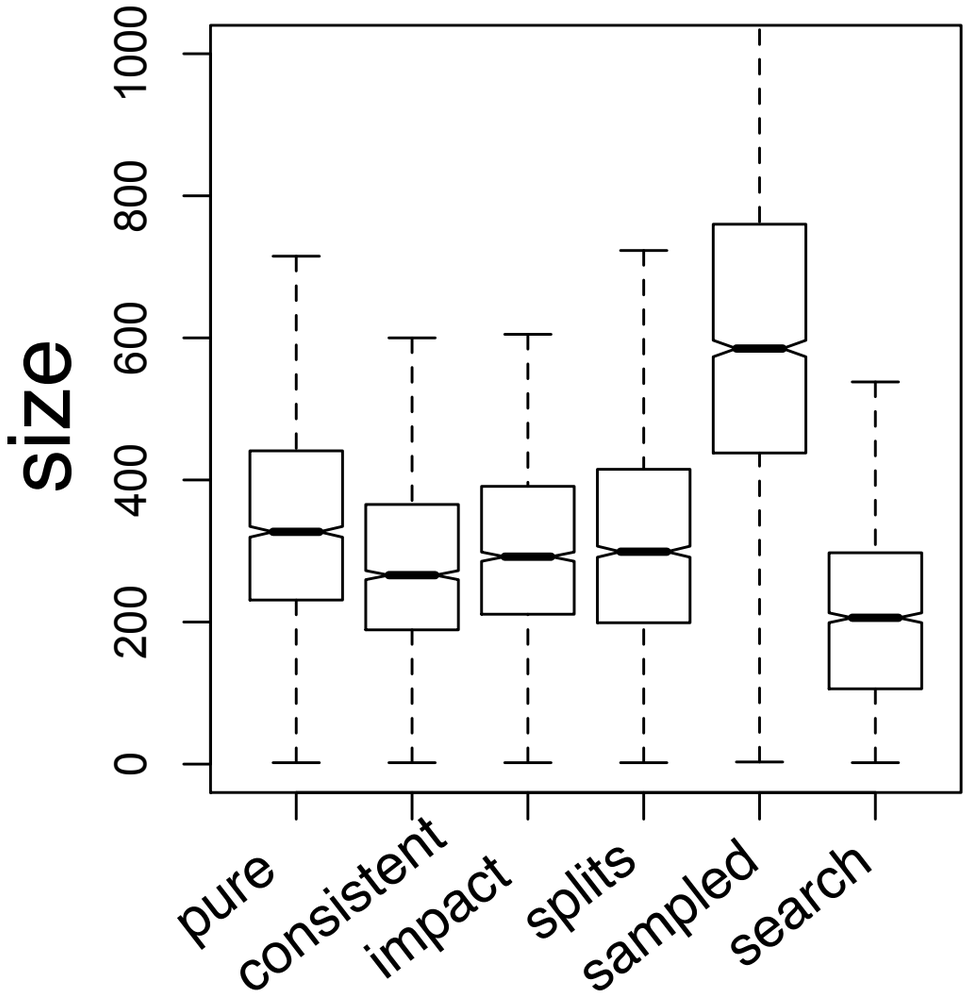
Labeled search

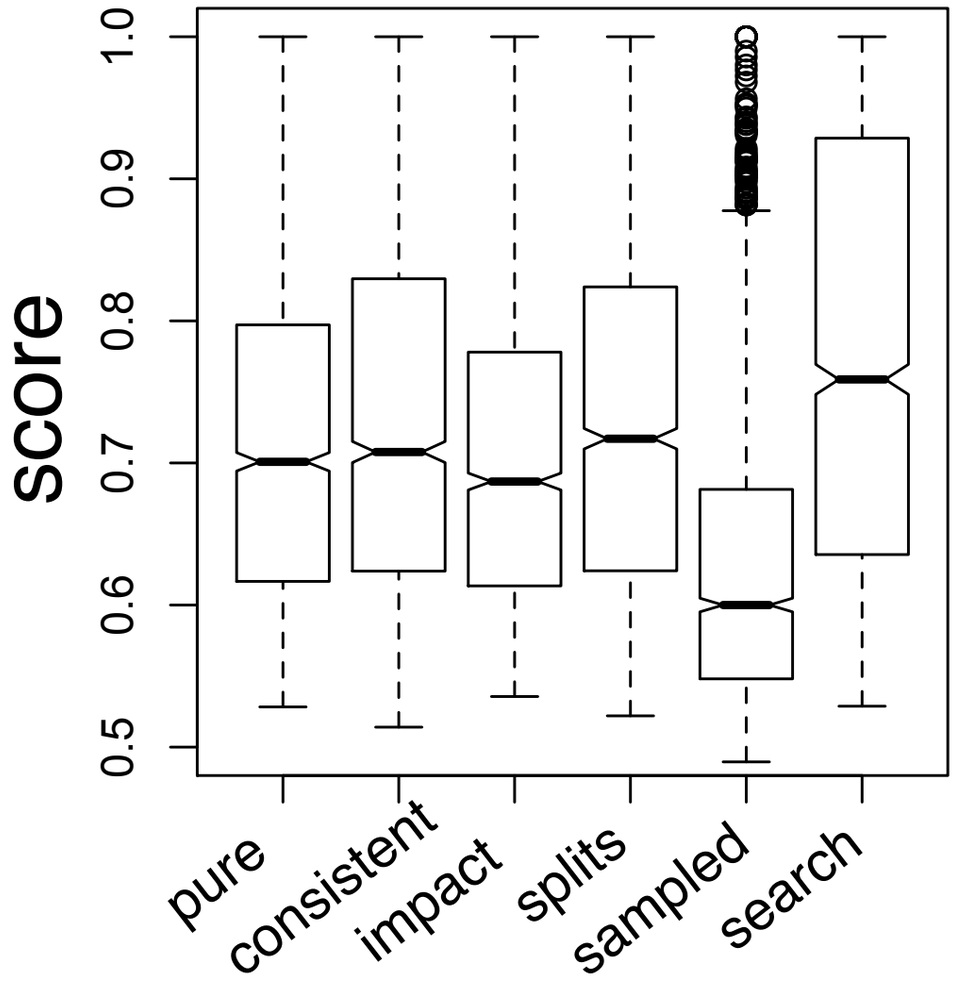


Choose the split/merge that in the end (after a greedy run) results in the smallest PDRTA

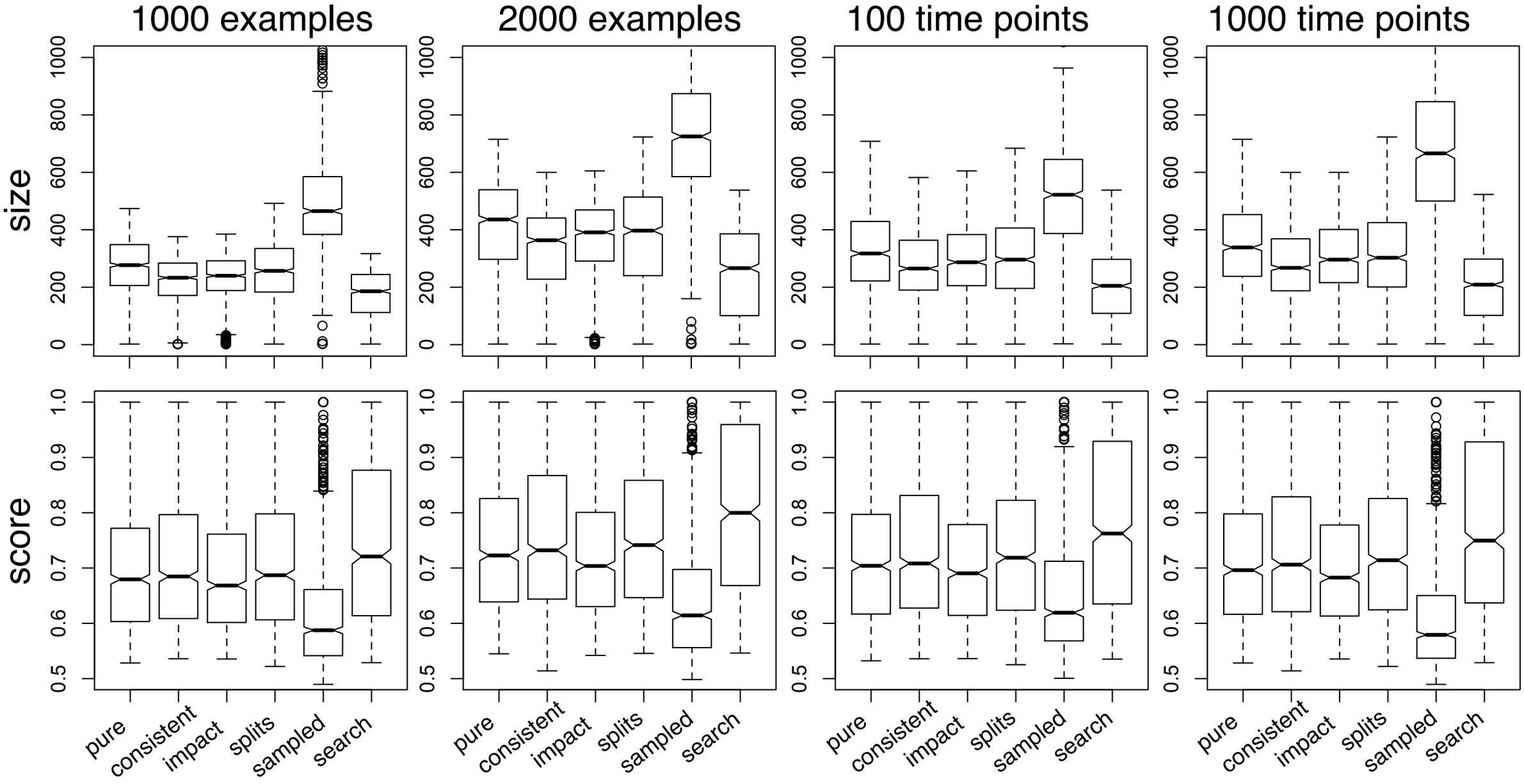
TU Delft

Results


size
0
200
400
600
800
1000
pure
consistent
impact
splits
sampled
search


score
0.5
0.6
0.7
0.8
0.9
1.0
pure
consistent
impact
splits
sampled
search

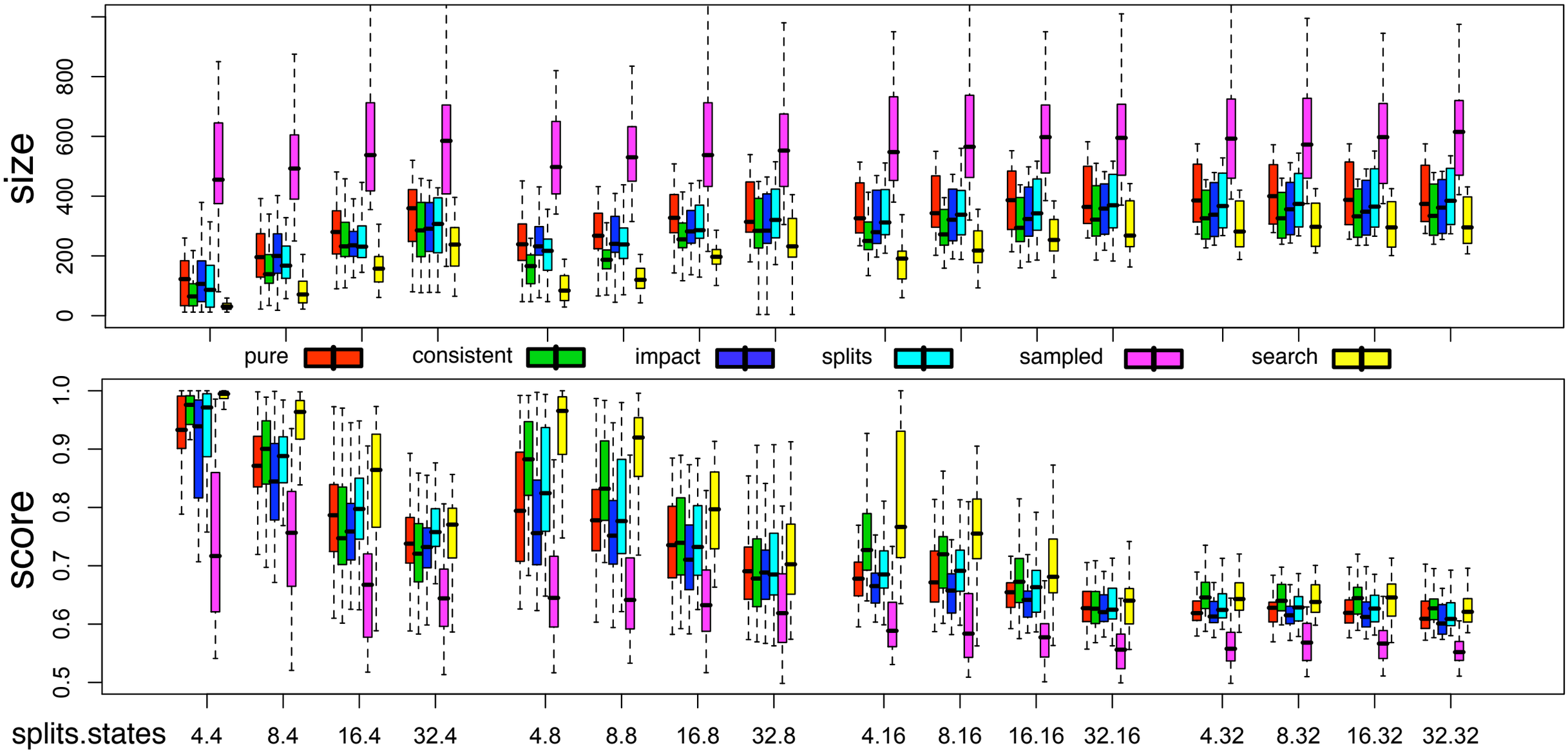
Results



TU Delft

Results

alphabet size 4



TU Delft

Overview

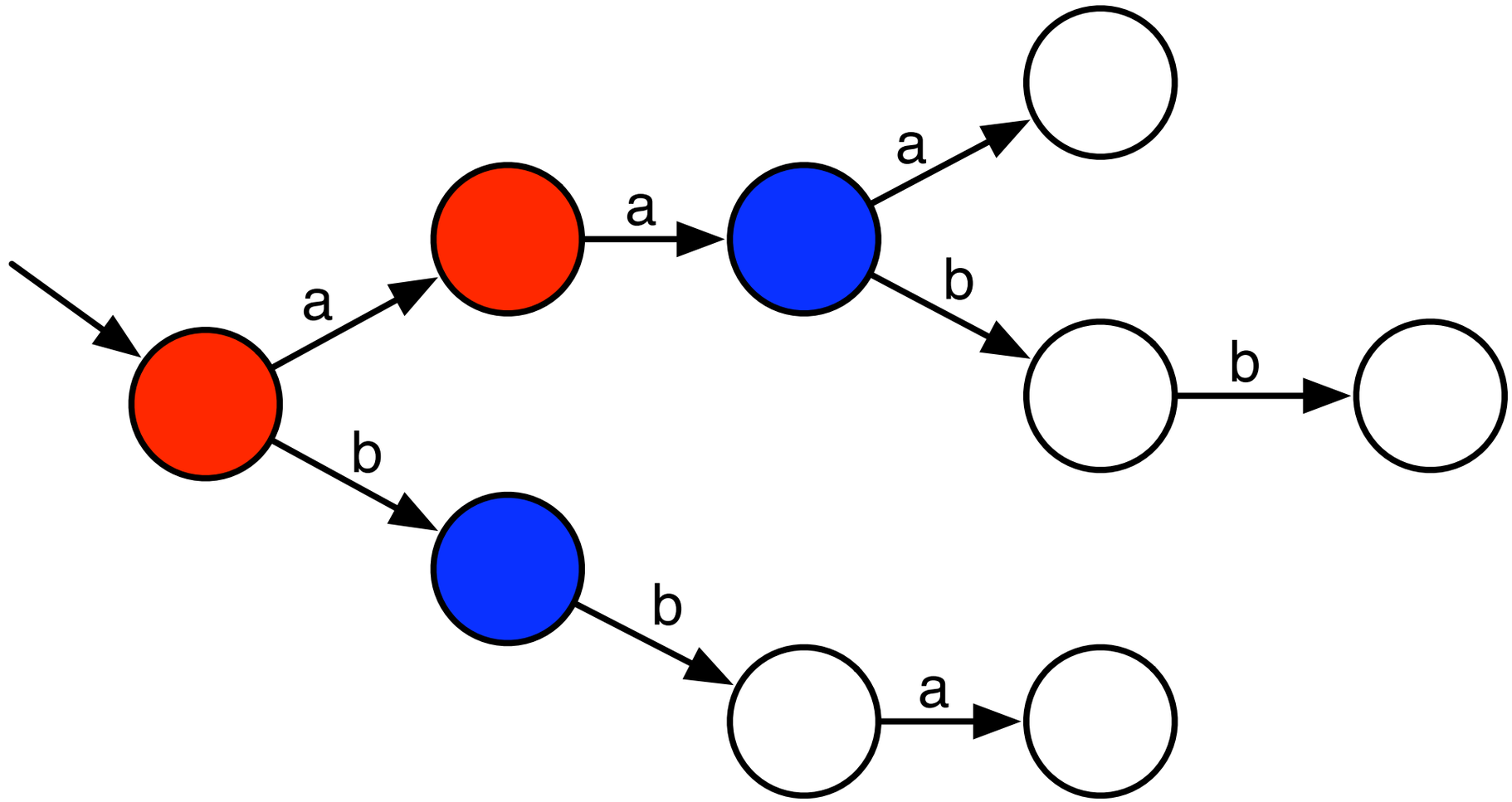
- Automata & how to learn them
- Real-time automata
- Learning real-time automata from labeled data
  - Results
- Learning real-time automata from unlabeled data
  - Results
- Conclusions

TU Delft

Unlabeled data

- Input: one sets S_+ of positive timed strings
- Occurs more often in practice
- Use the state merging and transition splitting algorithm, but
  - Maintain a core of inferred states using the red-blue framework
  - Only perform splits/merges of the transition/state that is visited by the most examples from S_+

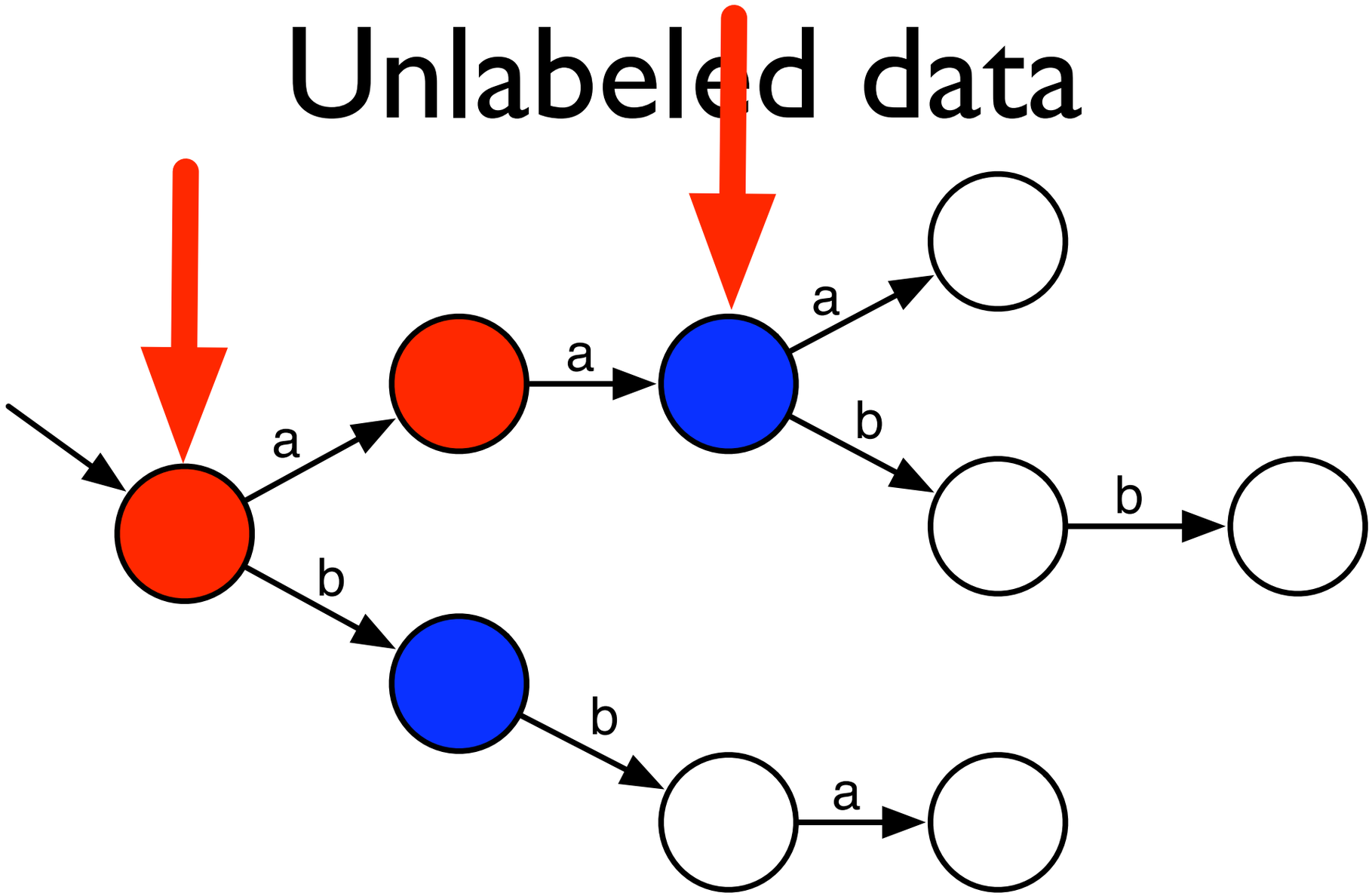
Unlabeled data



There are only positive states

TU Delft

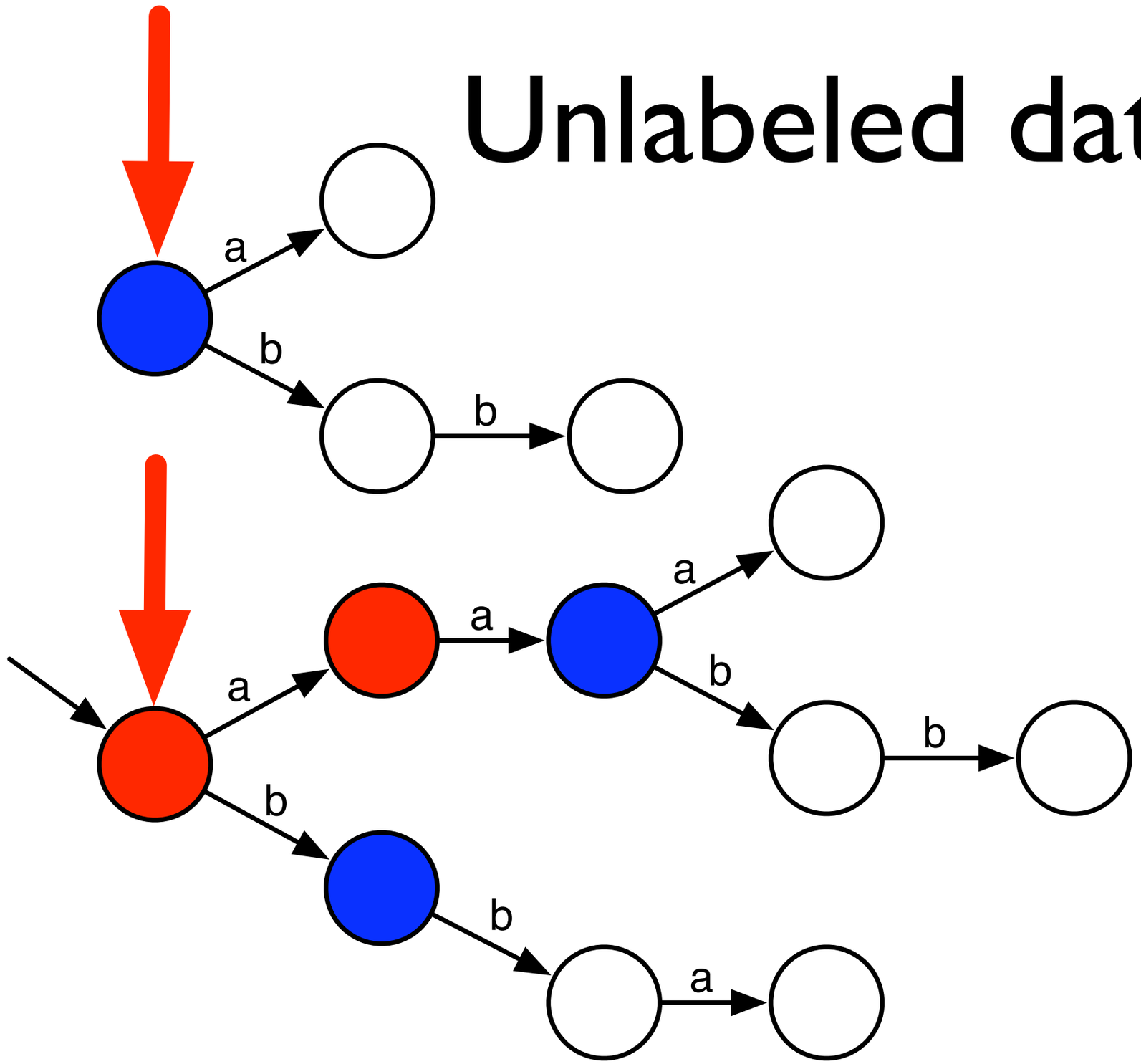
Unlabeled data



States should be merged if their future behavior is similar

TU Delft

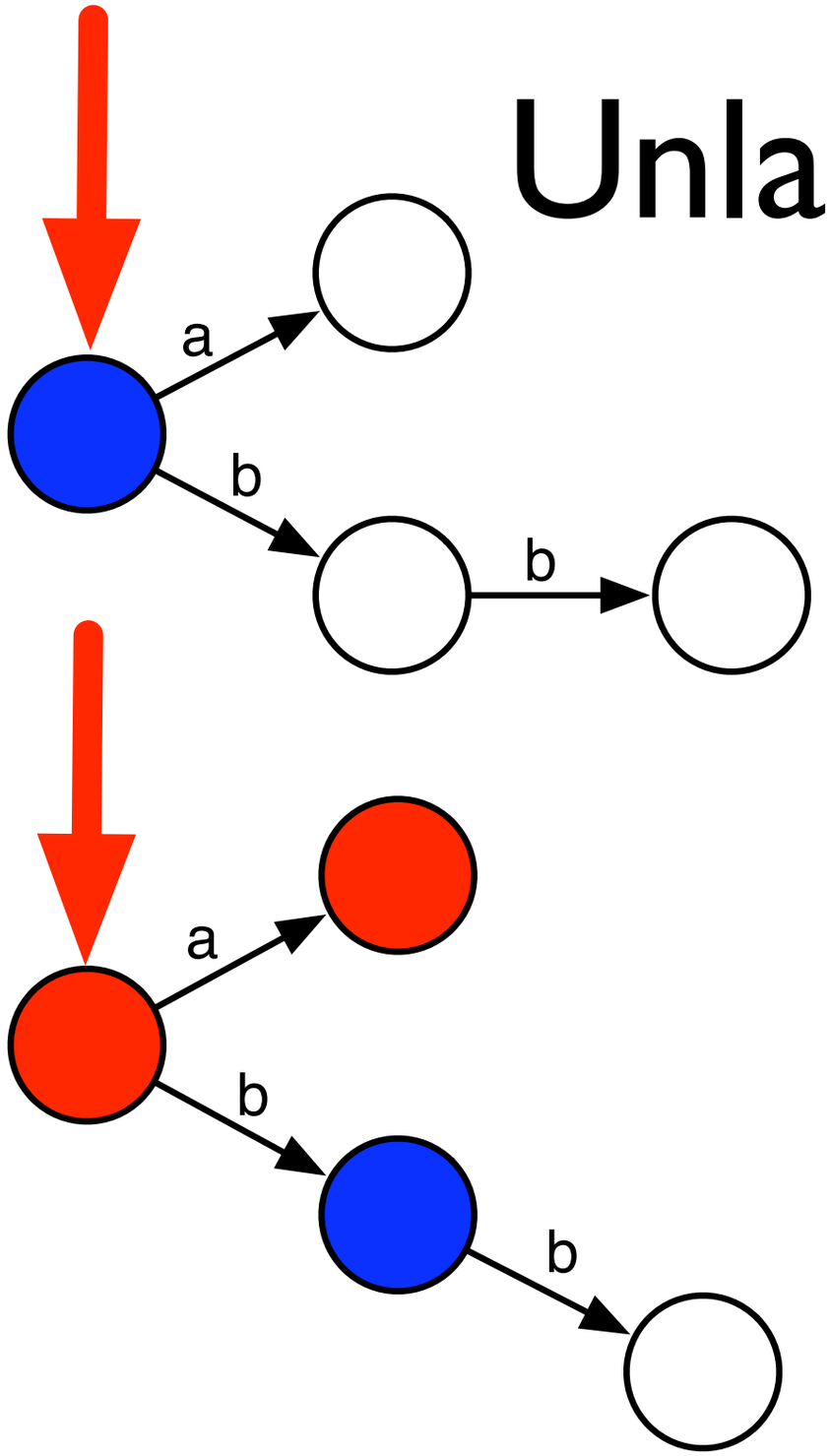
Unlabeled data



This behavior is determined by the trees rooted by the two states

TU Delft

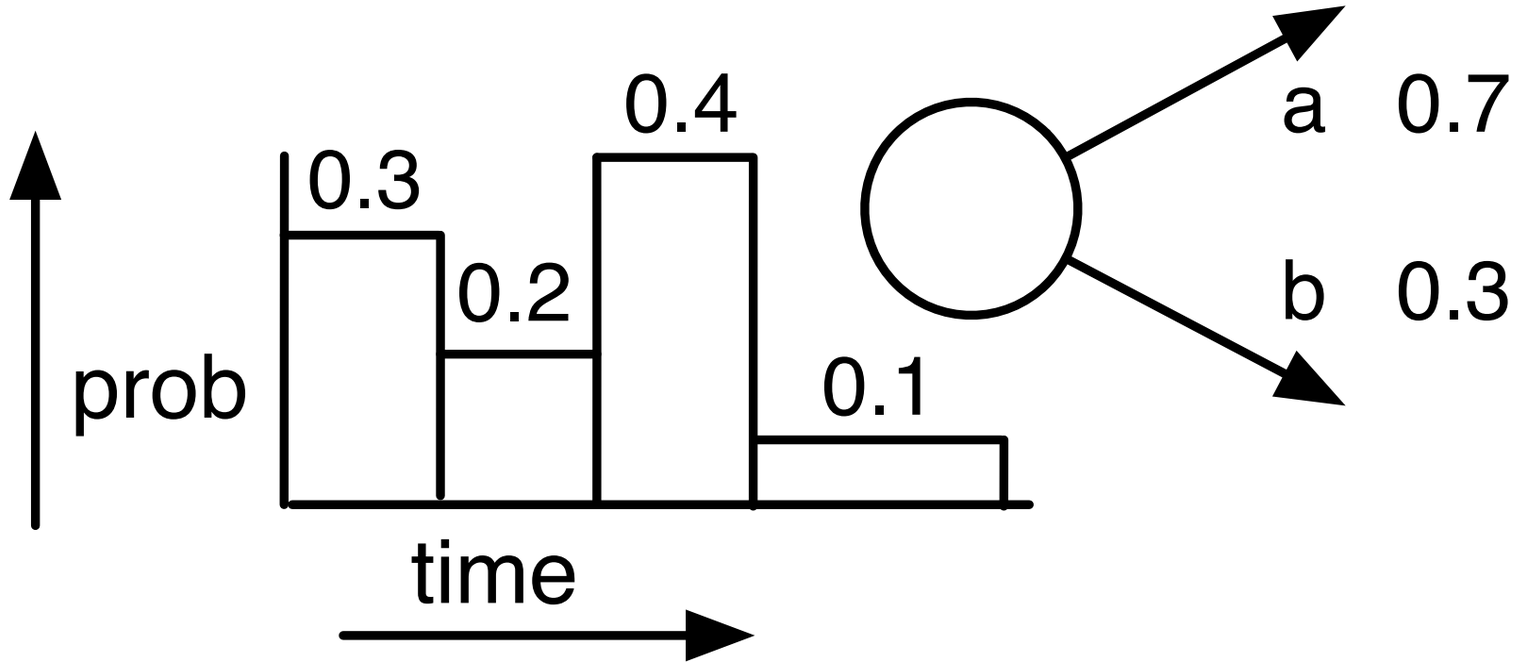
Unlabeled data



We are only interested in the overlapping behavior, i.e., the behavior for which we have data

TU Delft

DRTA Distributions



By counting the occurrences of symbols, and bins of time points (forming a histogram) in S_+

Every state defines a distribution over timed symbols

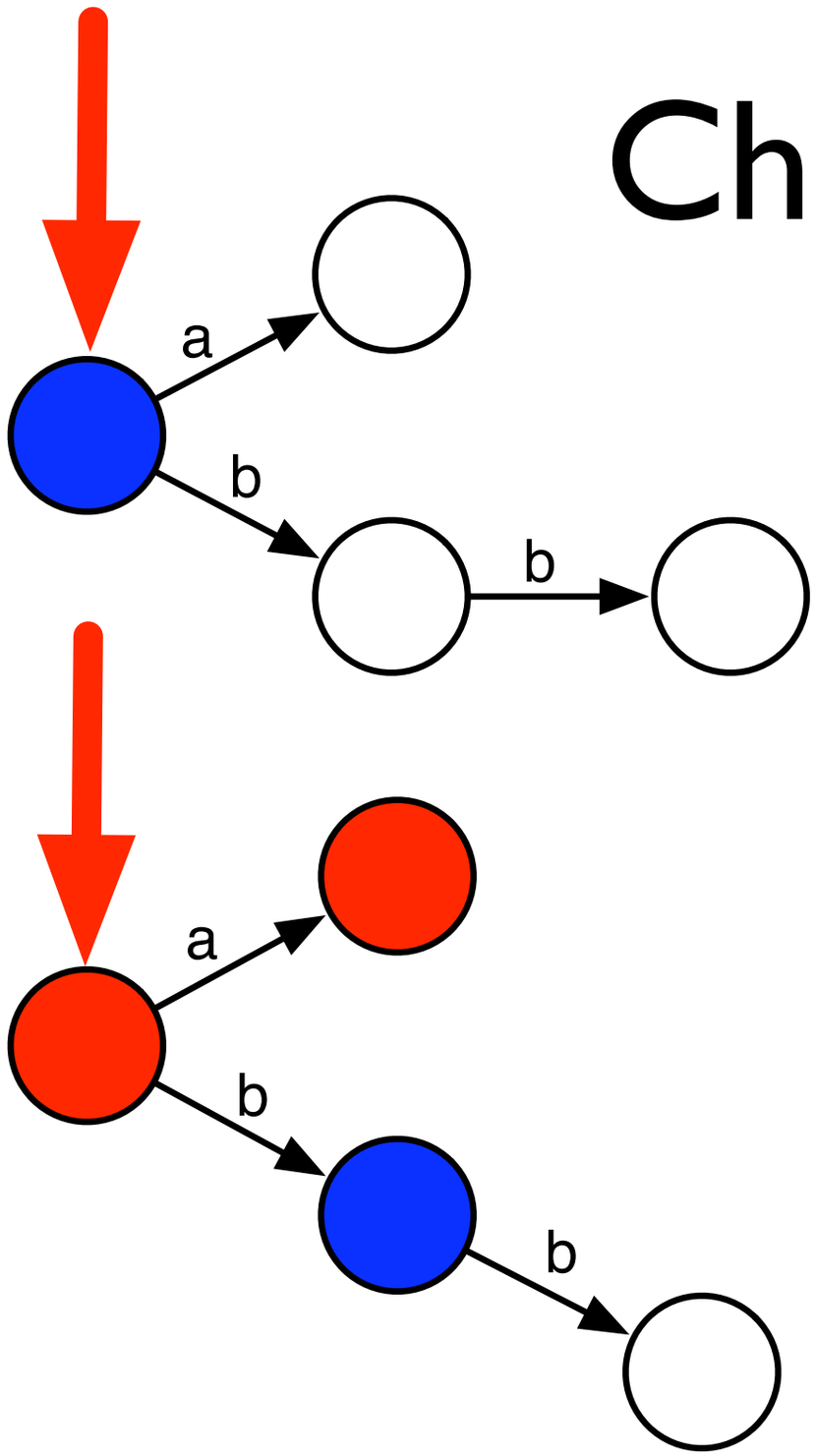
TU Delft

Three similarity values

- Two states and their future states define a distribution over timed strings
- Two states are similar if:

1. Chi squared, for S_+ their distributions are not statistically dissimilar
2. Likelihood ratio, merging them results in a significantly smaller model with respect to the likelihood of S_+
3. Akaike information criterium (AIC), merging them results in a smaller AIC (model selection criterium) for S_+

Chi squared



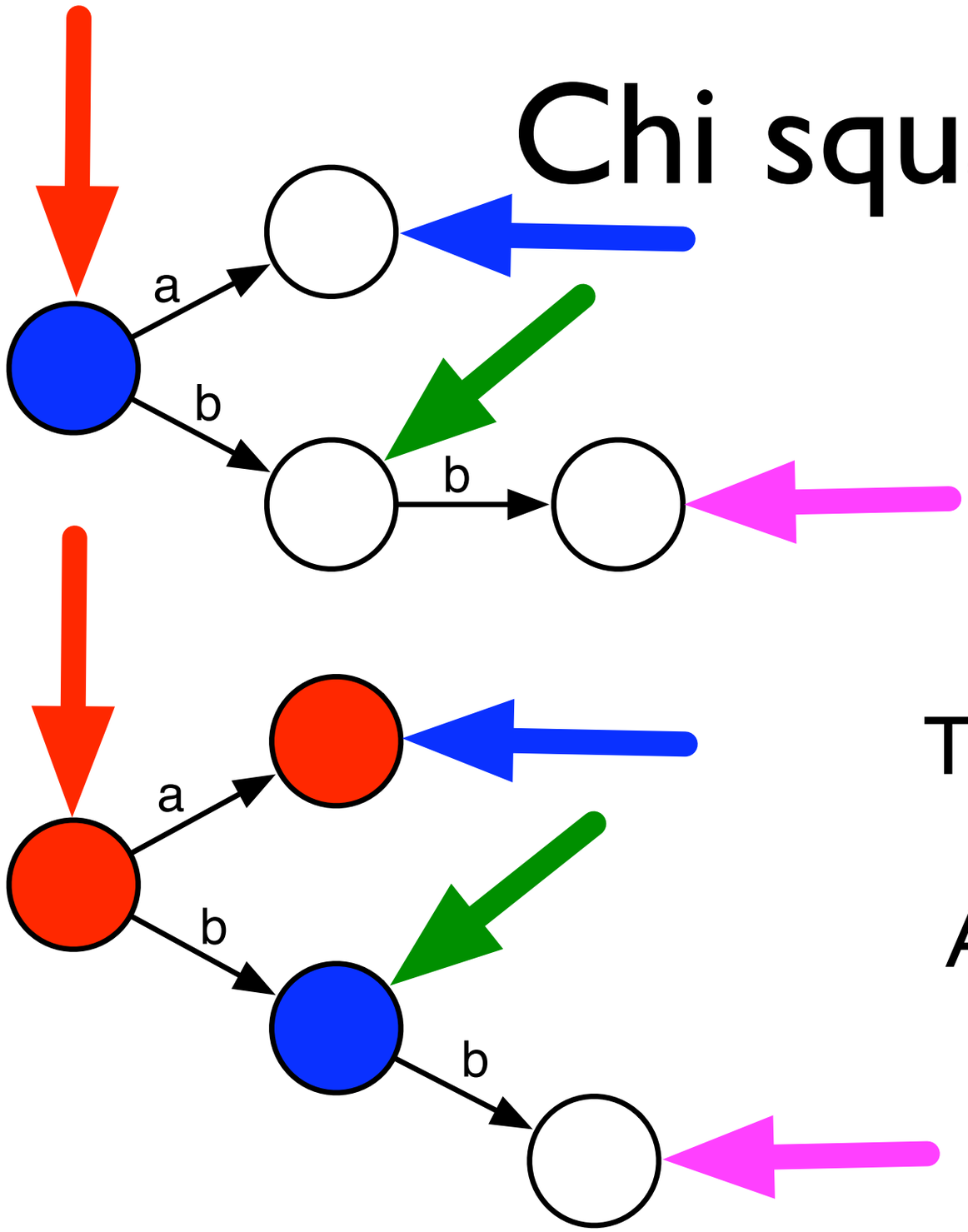
In the two states:

Compute a Chi-squared test for independence of the distributions over time values and symbols

This results in two p-values

TU Delft

Chi squared



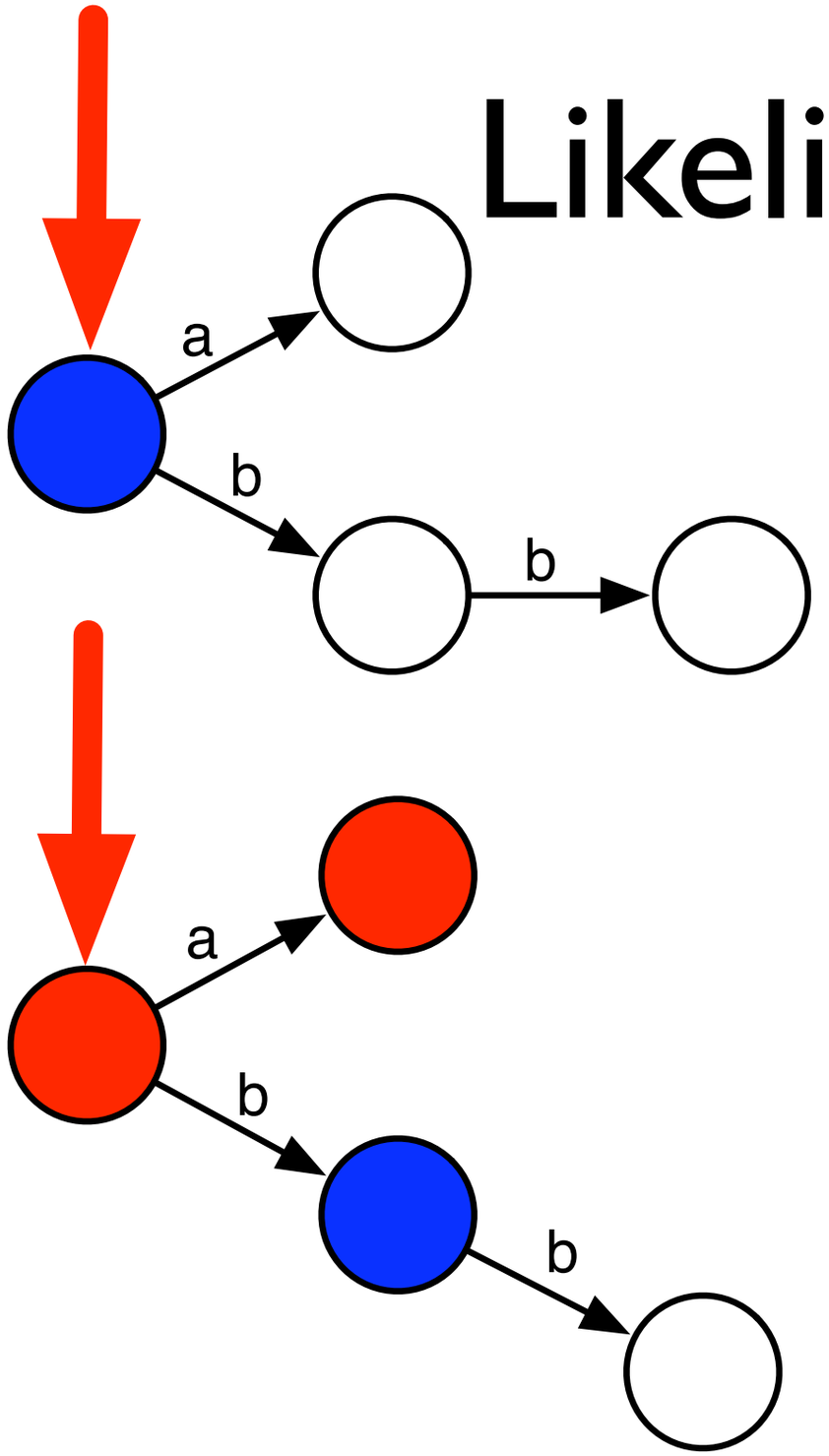
Do the same for every other overlapping future states

The result is many p-values

All of these are combined into a single test using a consensus test

TU Delft

Likelihood ratio

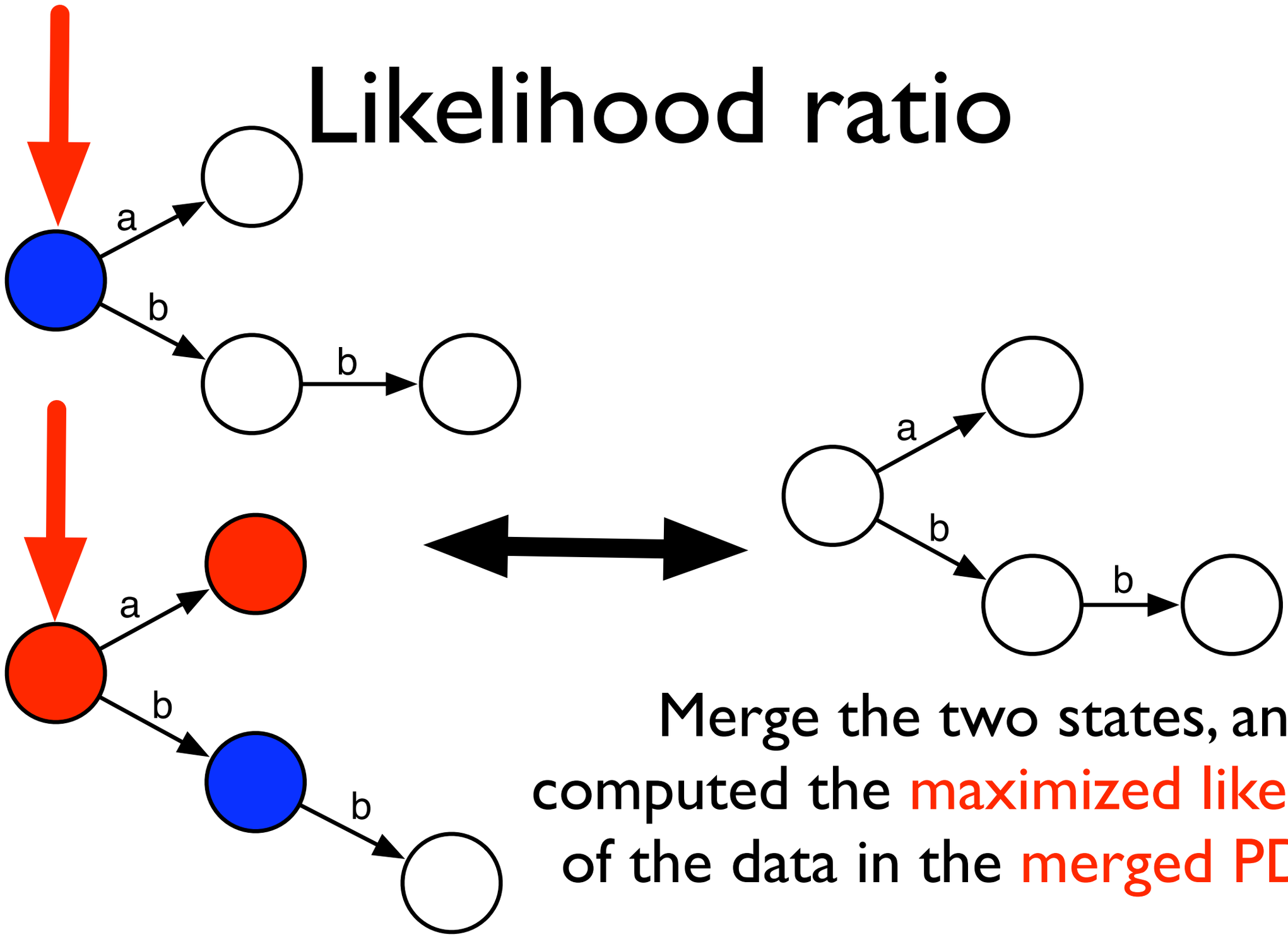


In the two states and the future states:

Determine the maximized likelihood of the data

TU Delft

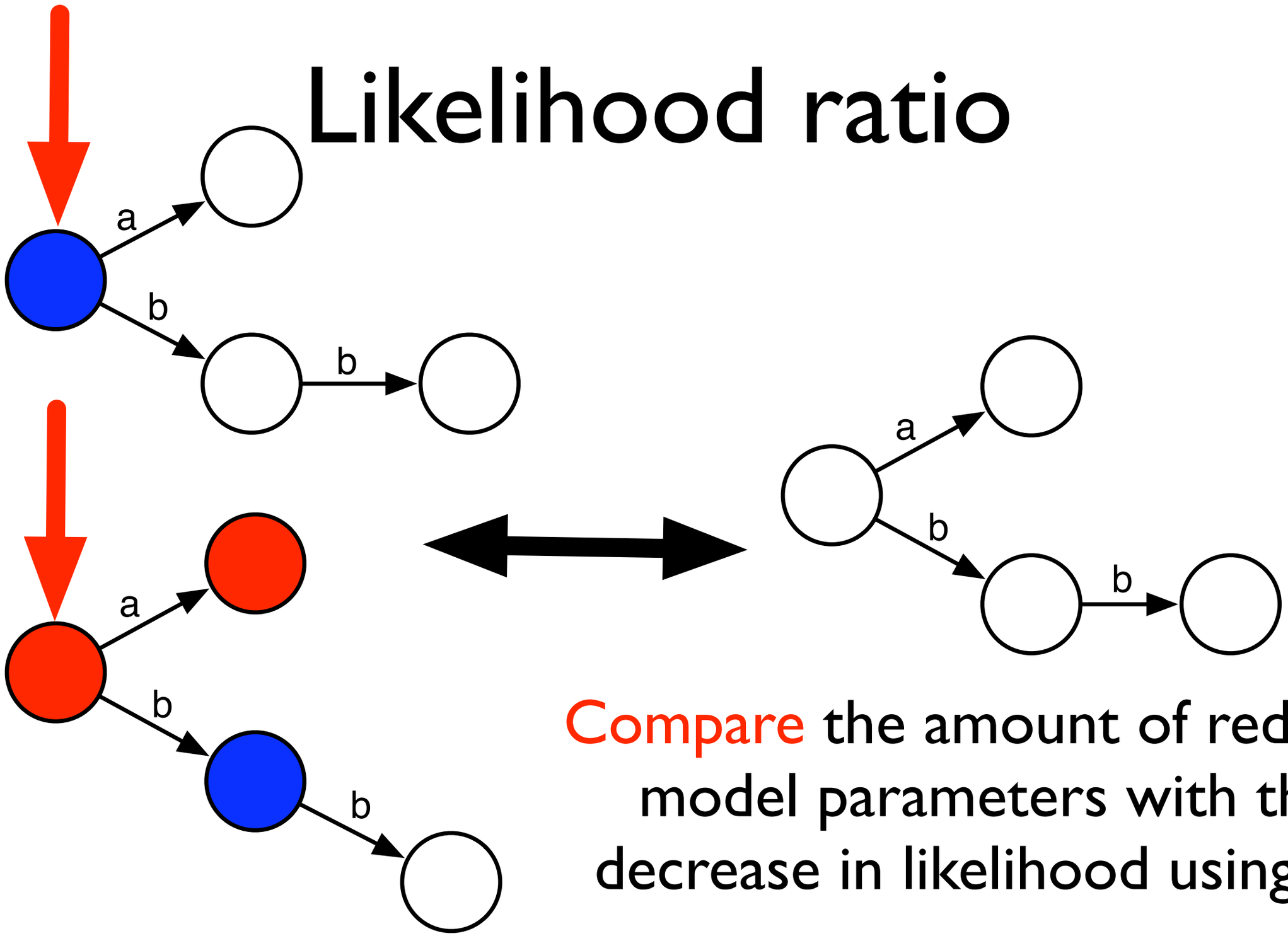
Likelihood ratio



Merge the two states, and computed the maximized likelihood of the data in the merged PDRTA

TU Delft

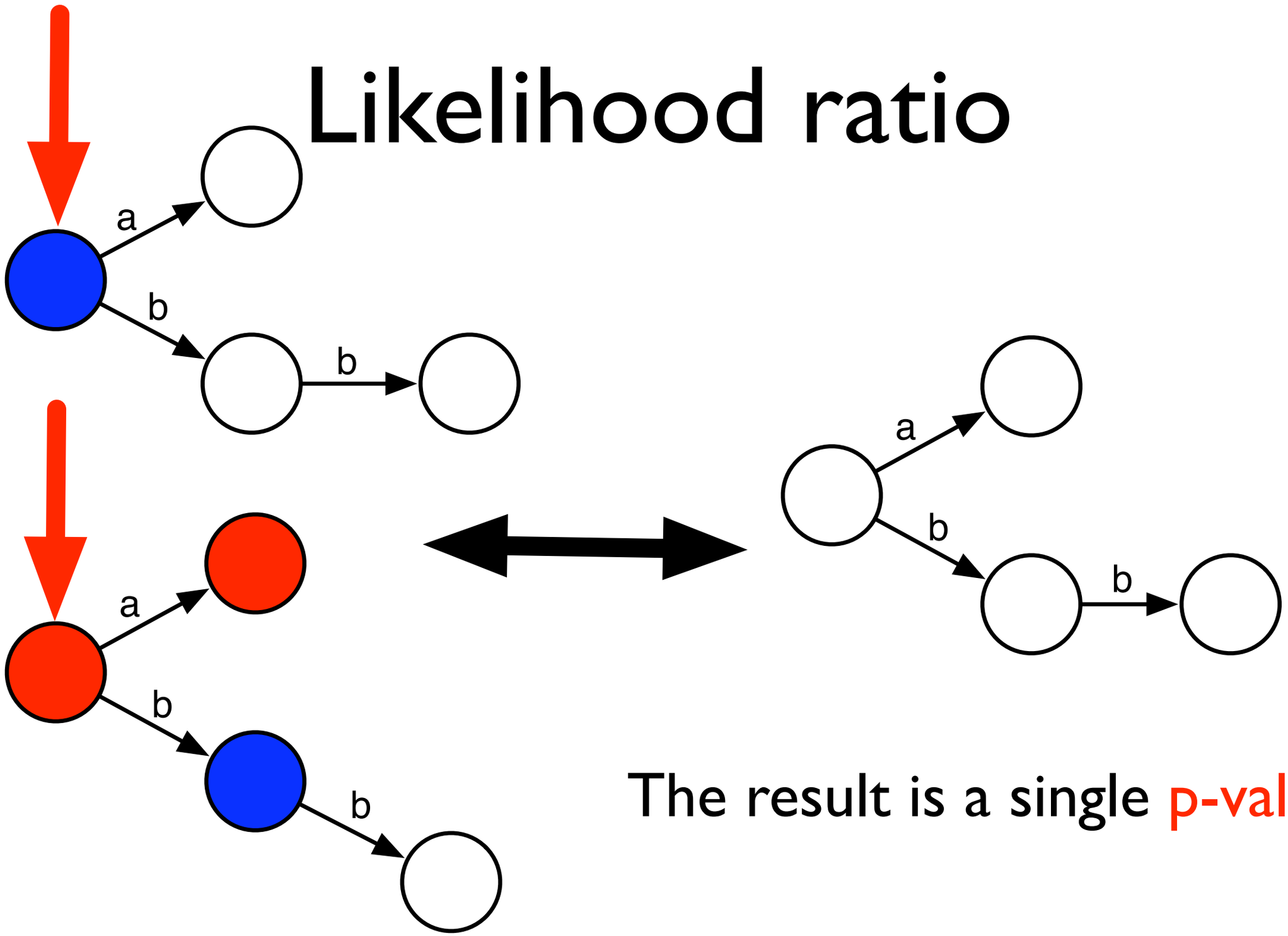
Likelihood ratio



Compare the amount of reduced model parameters with the decrease in likelihood using LR

TU Delft

Likelihood ratio



The result is a single p-value

TU Delft

Using similarity values

- Try all possible merges and splits of the transition to a blue state, compute their similarity p-values
- If the smallest split p-value is less than 0.05:
  - perform the split with the smallest p-value
- Else if the largest merge p-value is greater then 0.05:
  - perform the merge with the largest p-value
- Else color the blue state red

Using the AIC

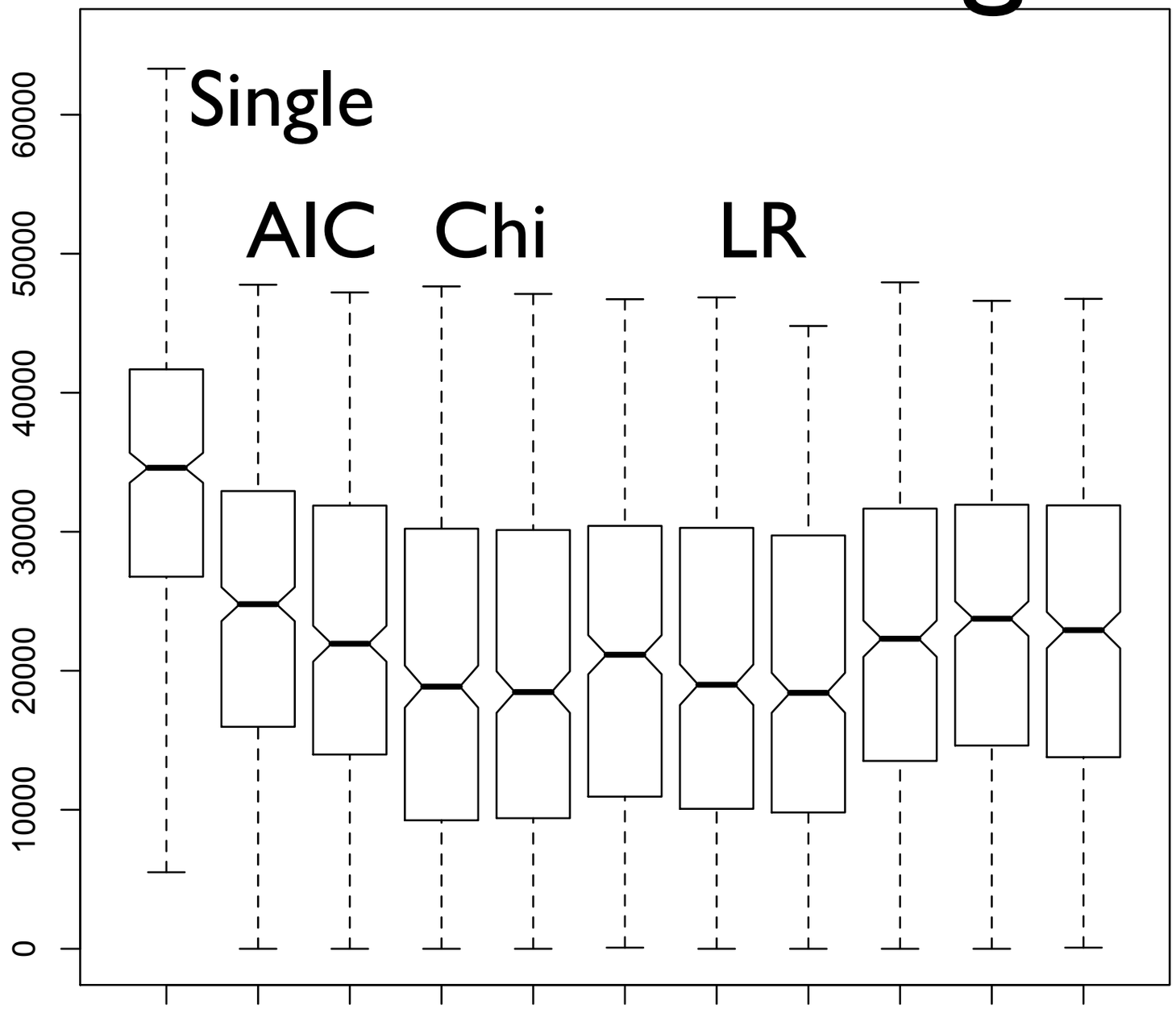
- Try all possible merges and splits of the transition to a blue state, compute their AIC values
- If the AIC of a DRTA resulting from a merge or split is less than the AIC of the current DRTA:
  - Perform this merge or split
- Else
  - Color the blue state red

Results 1000 strings

AIC result

-

AIC optimal



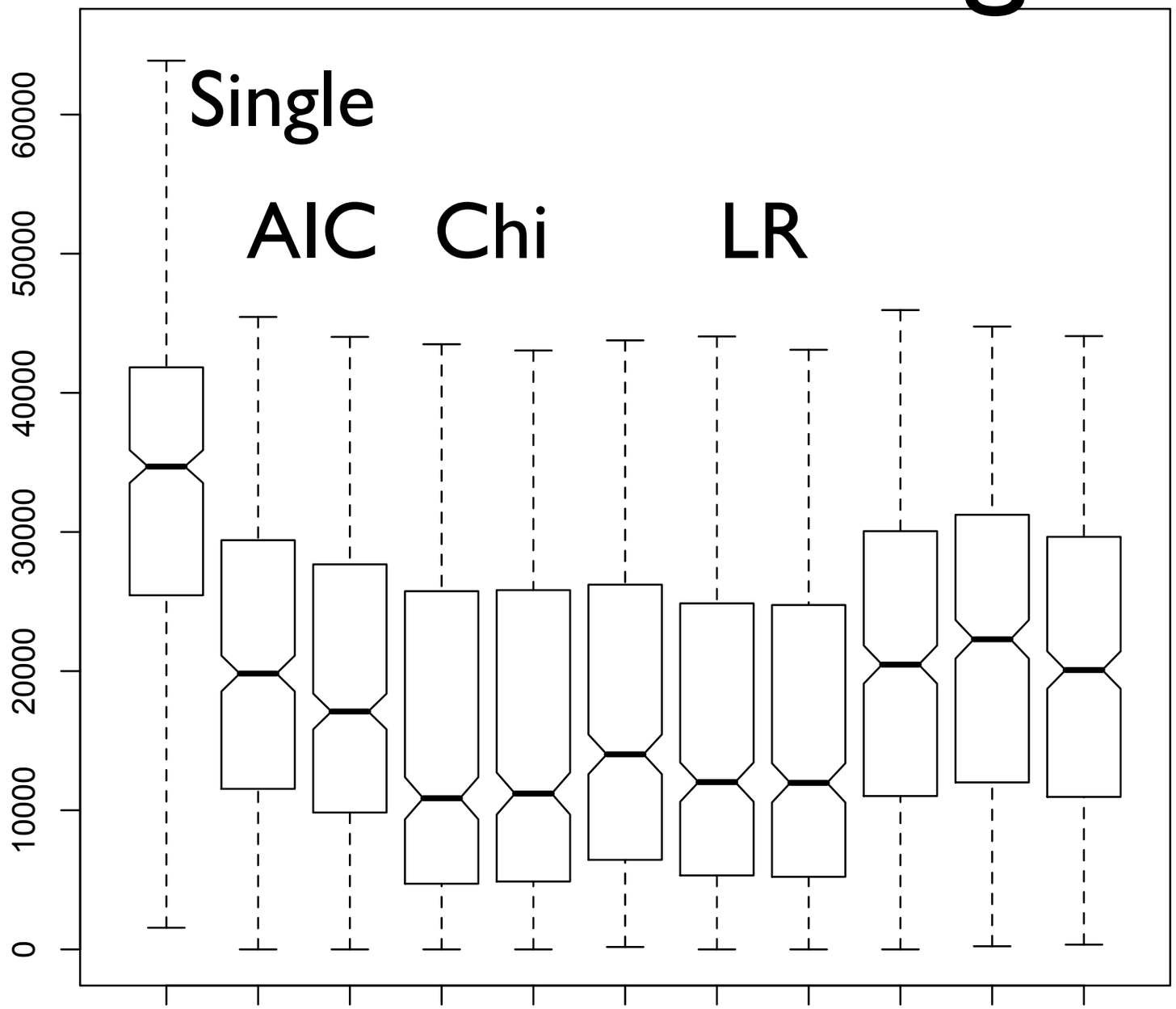
TU Delft

Results 2000 strings

AIC result

-

AIC optimal



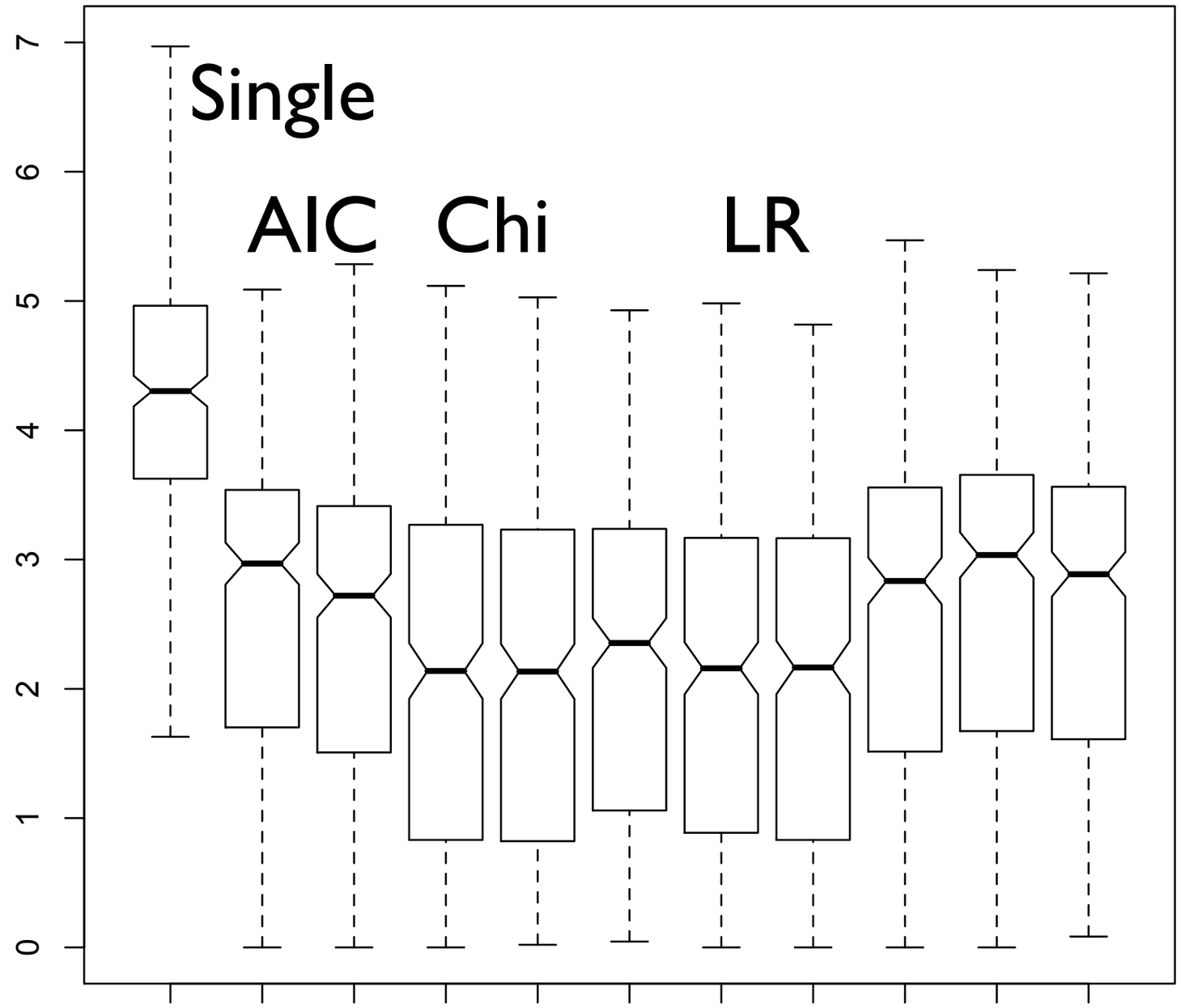
Results

Alphabet size = 4

Perplexity result

-

Perplexity optimal

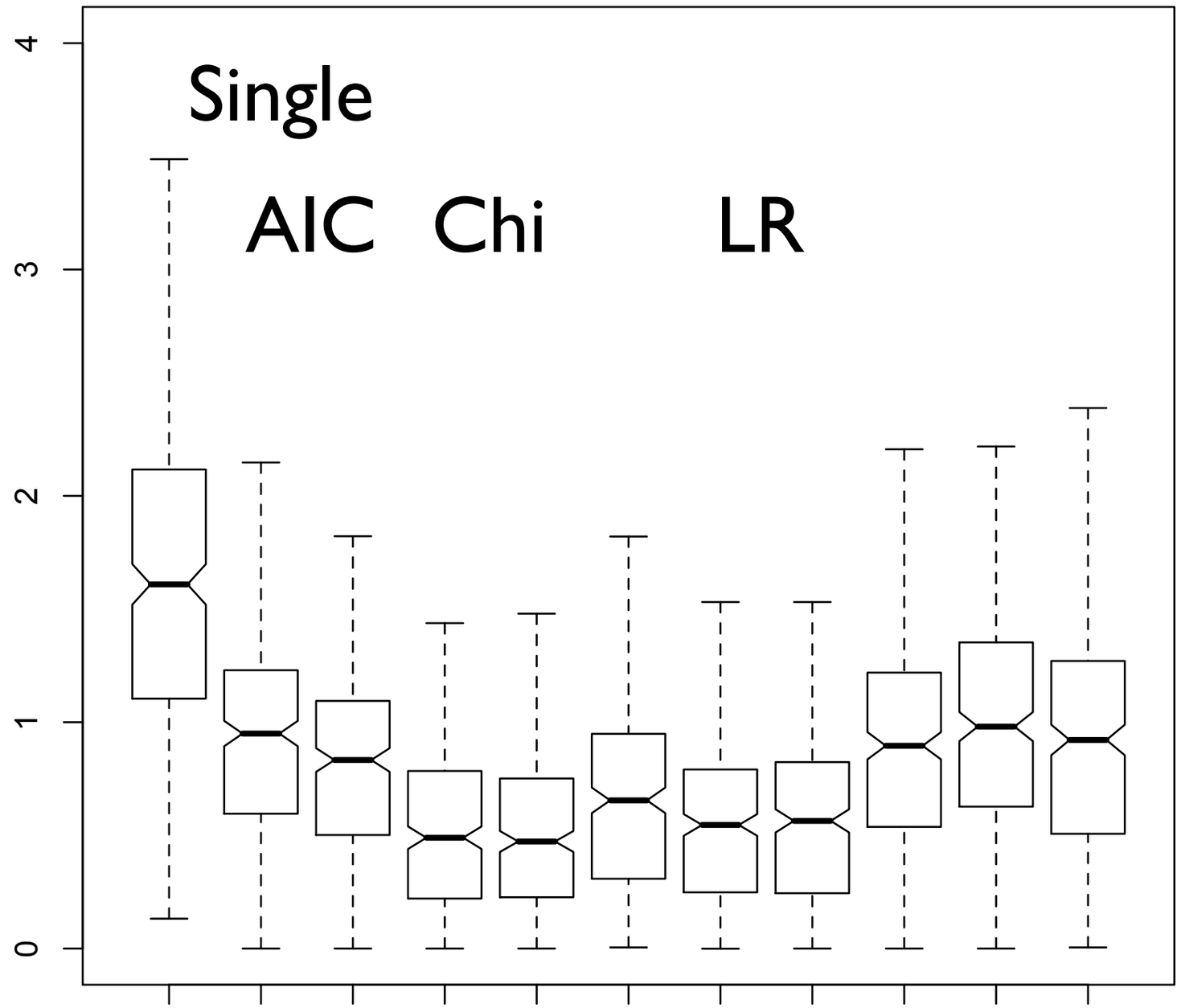


TU Delft

Results

Alphabet size = 2

Perplexity result - Perplexity optimal



TU Delft

Conclusions

- We have an efficient algorithm for learning deterministic real-time automata (DRTA)s
- The algorithm works both on labeled and unlabeled data
- The algorithm is based on the best known algorithm for learning DFAs, state-merging
- It shows promising results on artificial data

Contact

- http://www.st.ewi.tudelft.nl/~sicco/
- s.e.verwer@tudelft.nl

- DFA learning references, see:
  - Colin de la Higuera, A bibliographical study of grammatical inference, Pattern Recognition, Volume 38, Issue 9, Pages 1332 1348
- DRTA (and TA) learning references, see my homepage