Exact DFA identification using SAT solvers

Marijn Heule[1], Sicco Verwer[2]
ICGI 2010

1. Delft University of Technology
2. Eindhoven University of Technology

Overview

- Why use satisfiability to solve DFA learning?
- Learning DFAs as graph coloring
- Encoding DFA learning into satisfiability
- Results
- Conclusions

Learning DFA

- Given data (a set of positive and negative strings)
- Find the smallest DFA that is consistent with the data (that can produce these positive and negative stings)

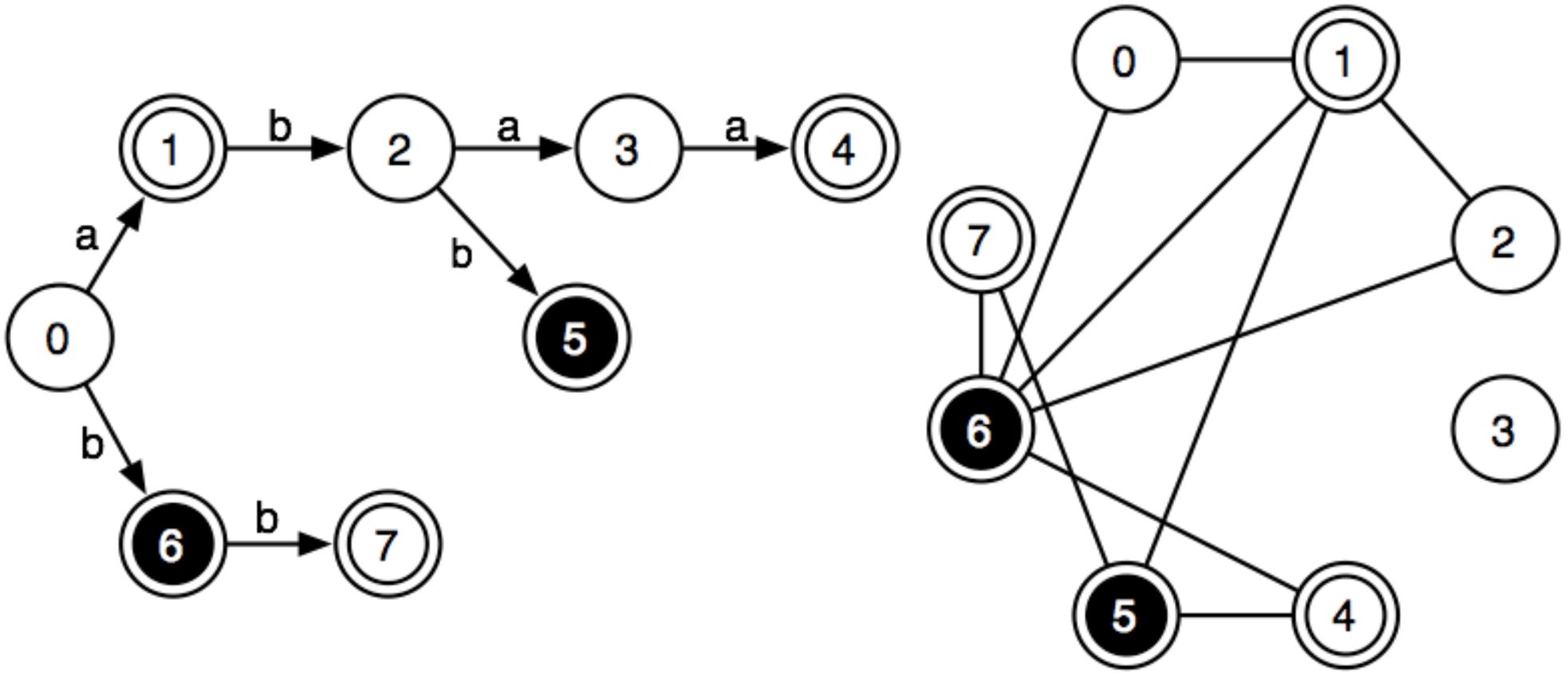
Why use satisfiability?

- Satisfiability is one of the best studied problems in computer science
- SAT-solvers are highly optimized and use advanced search techniques
- We can make direct use of these advanced techniques by encoding DFA learning to satisfiability and running a SAT-solver
- Such an approach has been shown to be competitive for several well-known problems

Graph coloring

- Given a graph G = (V, E) and a value k
- Is G colorable using k colors such that no two adjacent nodes have the same color?

from DFAs to GC



Start with a prefix tree

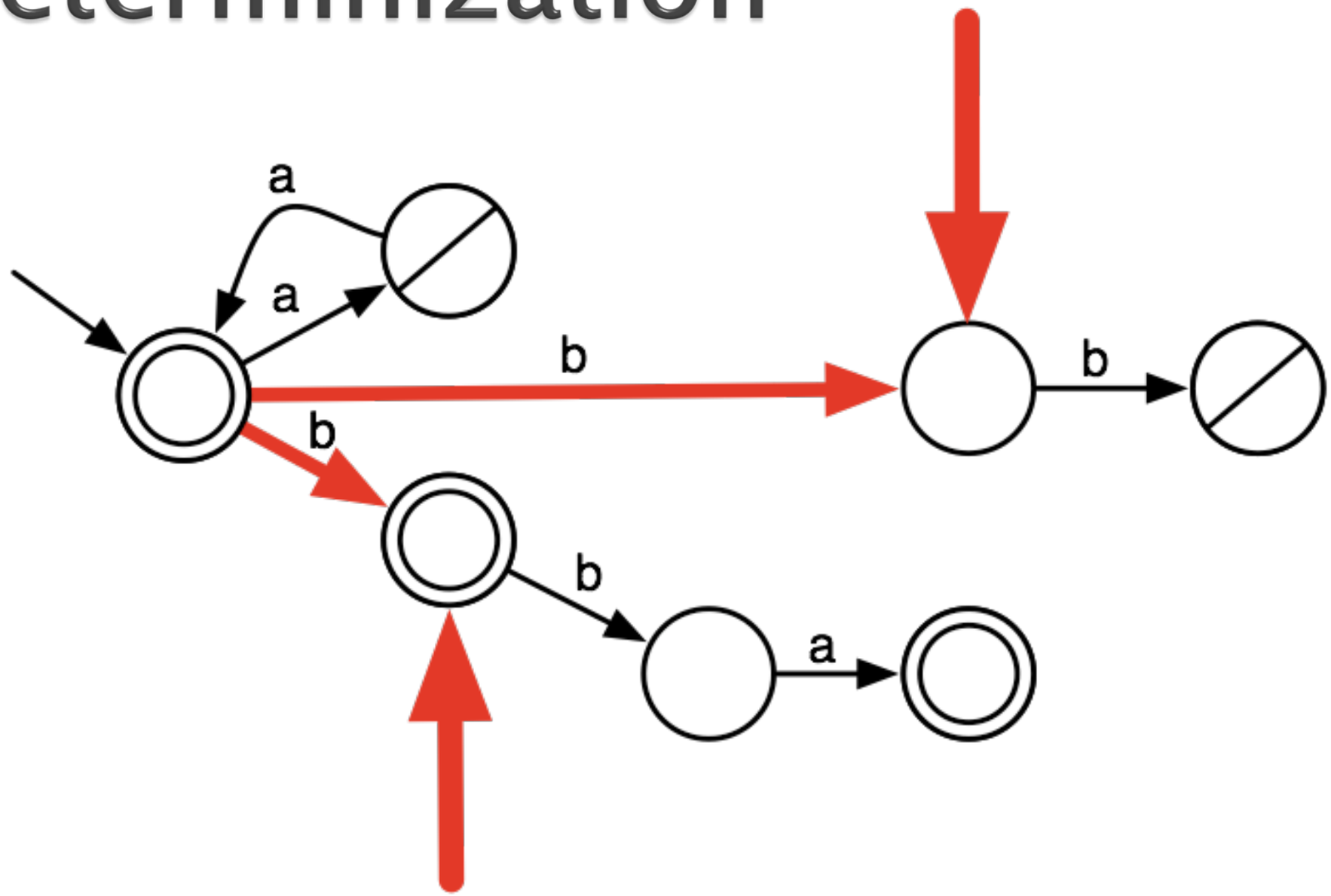
Create one node for every state

connect two nodes if the states cannot be merged

from DFAs to GC

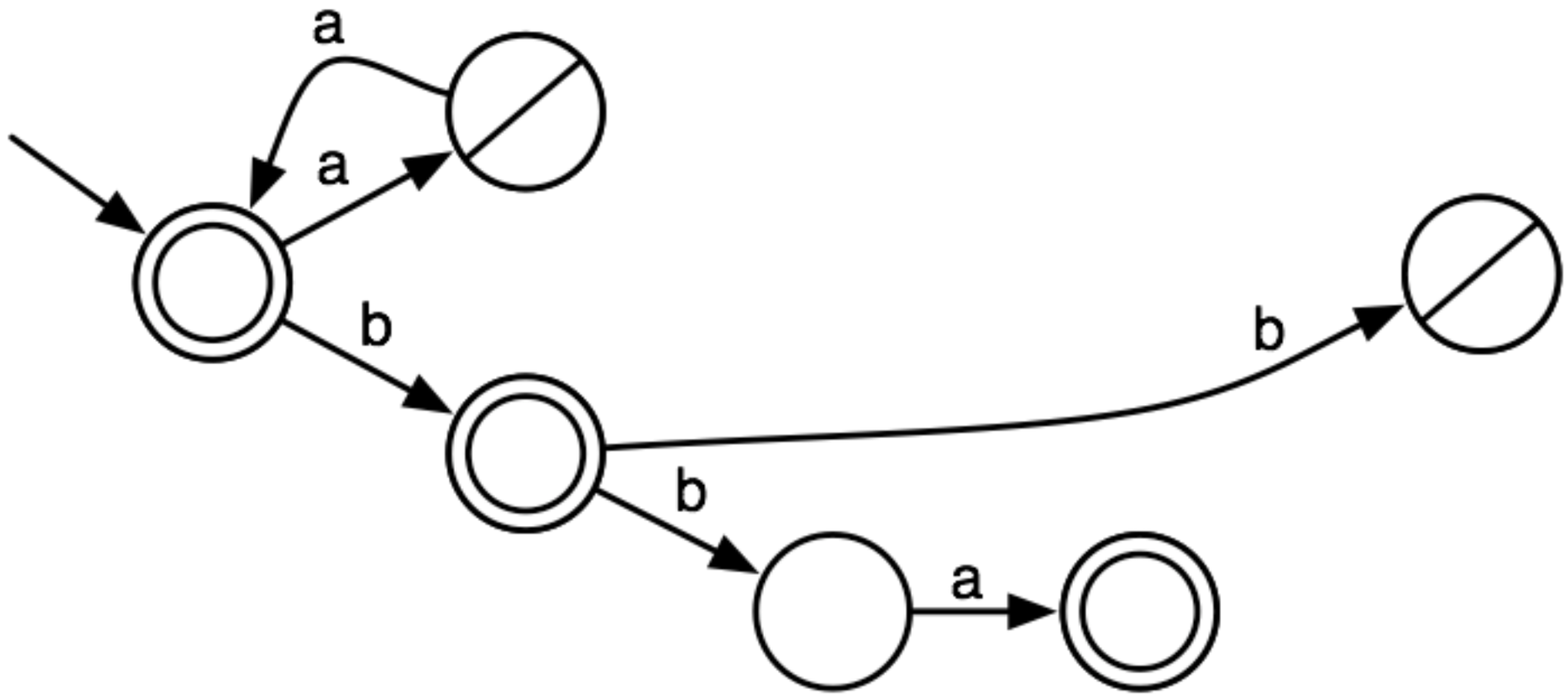
- This encodes inequality constraints:
  - restricts when two nodes cannot have the same color
- We also require equality constraints:
  - restrictions when two nodes can have the same color
- Equality constraints encode the determinization procedure....

Determinization



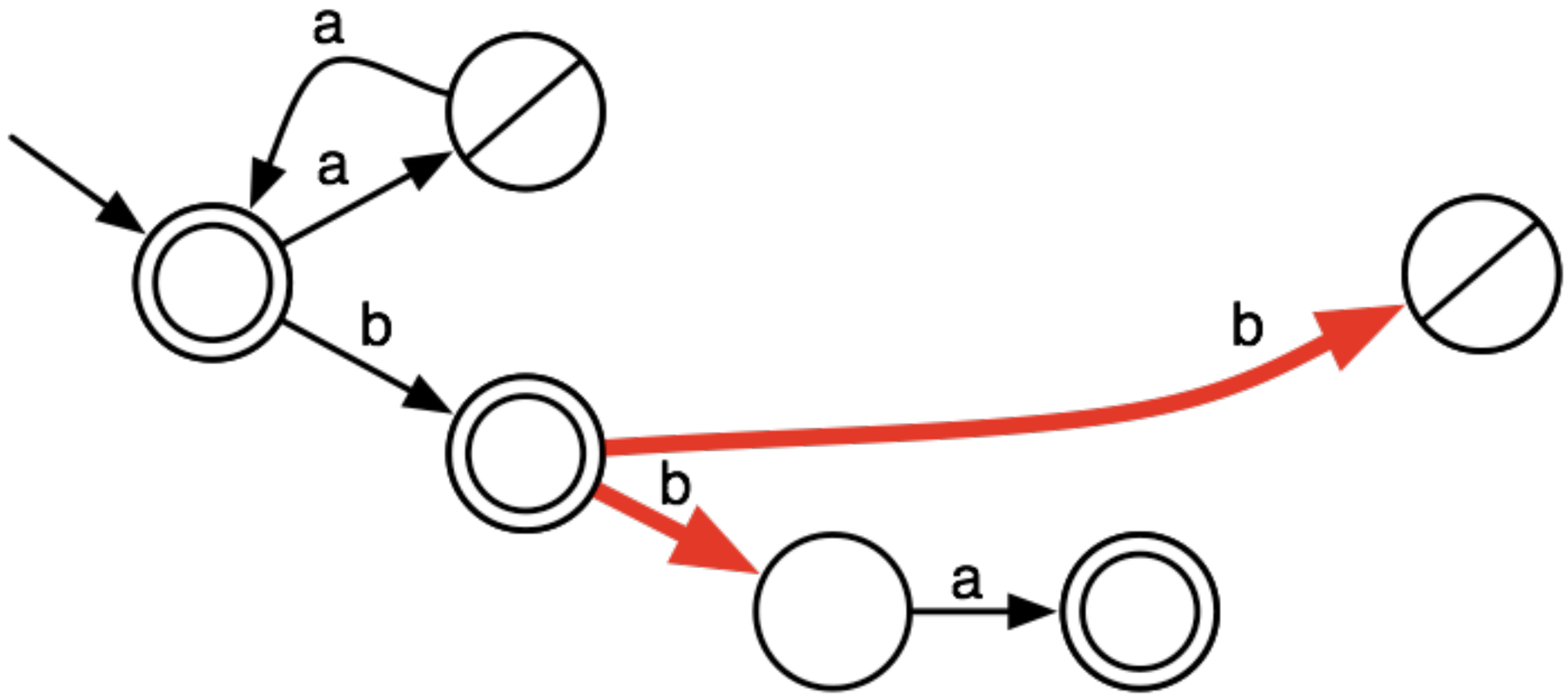
If two states are merged

Determinization



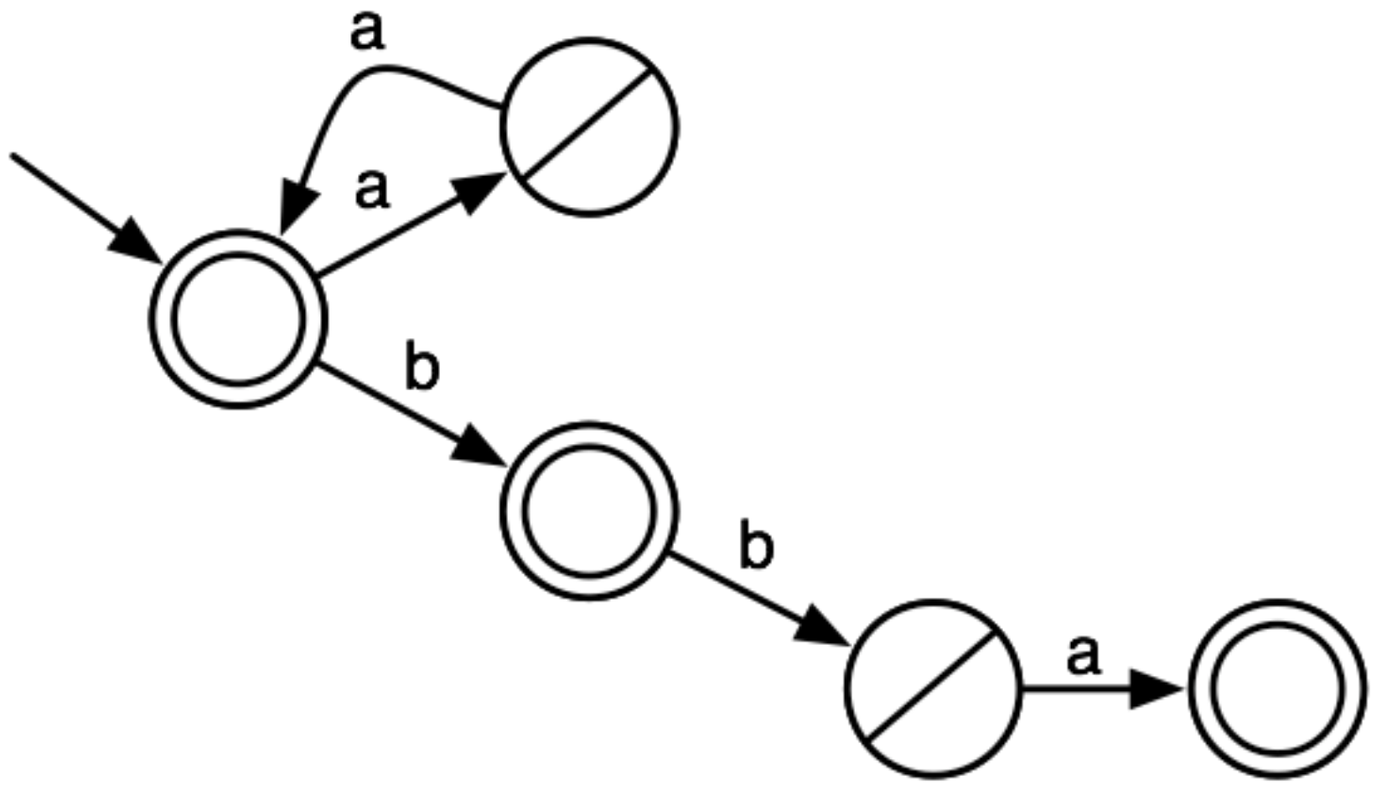
If two nodes have the same color

Determinization



And they have children reached by the same label

Determinization



Then these states are merged too

Then these nodes have the same color too

from DFAs to GC

- The equality constraints encode that:
  - two parent nodes can be colored with the same color
  - only if their children reached by the same label have the same color
- Such a constraint is difficult to implement in graph coloring
- We therefore encode it in satisfiability instead!

Satisfiability

- Given a formula *F* in propositional logic
- Does there exist an assignment of truth values to the variables of *F* such that each clause in *F* is satisfied?

Standard GC to SAT

- Variable $x_{v,i}$ denotes that vertex v has color i
- The following then encodes graph coloring:

$$(\neg x_{v,i} \vee \neg x_{w,i}) \quad \text{for each edge}$$

$$(x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,n}) \quad \text{for each node}$$

- Where v and w are adjacent vertexes, i.e., states that cannot be merged, this encodes inequality
- Requires $O(|C| \times |E|)$ clauses

Simple equality in SAT

- If *v* and *w* have the same incoming label this then encodes the equality constraints:

$$x_{p(v)} = x_{p(w)} = i \rightarrow x_v = x_w = j$$

  - $(\neg x_{p(v),i} \vee \neg x_{p(w),i} \vee x_{v,j} \vee \neg x_{w,j}) \wedge$
  - $(\neg x_{p(v),i} \vee \neg x_{p(w),i} \vee \neg x_{v,j} \vee x_{w,j})$

- Requires $O(|C|^2 \times |V|^2)$ clauses
- Unfortunately, this is too large for state-of-the-art SAT solvers

More efficient equality

- Additional variables:
  - $y_{a,i,j}$ denotes that for all vertexes with color *i*, the child reached by label *a* has color *j*
- This then encodes the equality constraints:
  - $(\neg x_{p(v),i} \vee \neg x_{v,j} \vee y_{l(v),i,j})$
  - $(\neg y_{a,i,j} \vee \neg y_{a,i,k})$
- Requires $O(|C|^2 \times |V|)$ clauses

More efficient inequality

- Additional variables:
  - z_i denotes that color i is used for accepting states
- This then encodes the inequality constraints:
  - $(\neg x_{v,i} \vee z_i) \wedge (\neg x_{w,i} \vee \neg z_i)$
  - Where node v is accepting and node w is rejecting

- Requires $O(|C| \times |V|)$ clauses
- The original GC constraints are now redundant

Redundant clauses

- Explicitly add all GC conflicts (edges)
  - $(\neg x_{v,i} \vee \neg x_{w,i})$
  - Where v and w cannot be merged
- A node cannot have multiple colors
  - $(\neg x_{v,i} \vee \neg x_{v,j})$
- Add more knowledge about the y variables
  - $(y_{a,i,1} \vee y_{a,i,2} \vee \ldots \vee y_{a,i,|C|})$
  - $(\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j})$

Learning automata by SAT

1. Generate the prefix-tree T
2. Initialize the set of colors L
3. Construct a SAT formula F using T and L
4. Solve F using a SAT solver
5. If F is unsatisfiable add a color to L, goto 3
6. Return the DFA found in step 4

Symmetry breaking

- When a SAT solver is applied to a graph coloring instance with *k* colors, it can try all *k!* color permutations
- These permutations are equivalent in terms of meaning and satisfiability
- We can avoid (part) of this redundant work by adding symmetry breaking predicates…

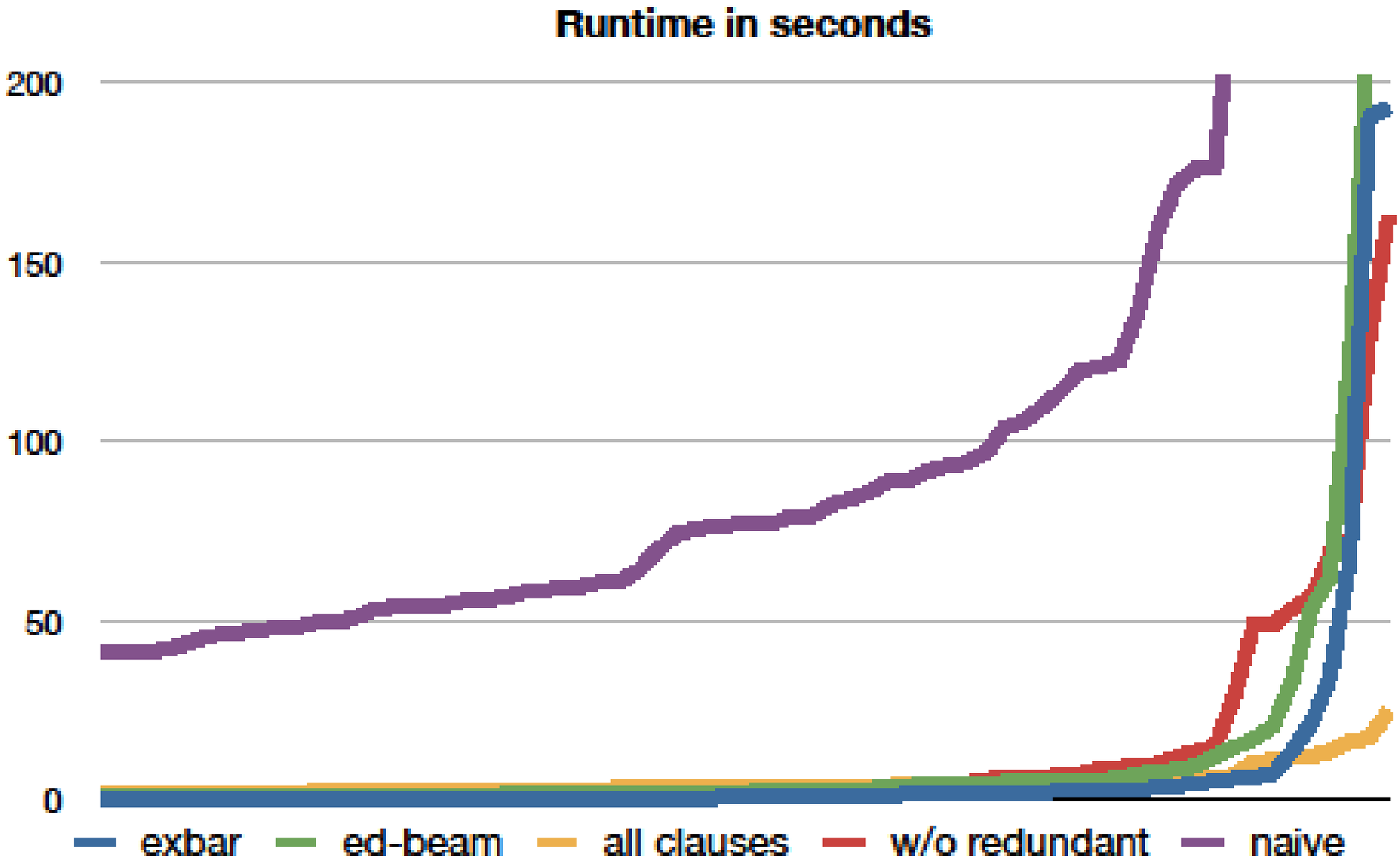
Symmetry breaking

- All vertices in a clique have to be colored differently
- We force this by first finding a large clique, and then pre-assigning a distinct color to each of the nodes in this clique
- Although finding the largest clique is an NP-complete problem, finding a large clique can be done efficiently

Learning automata by SAT

1. Generate the prefix-tree T
2. Compute a large clique C in T
3. Initialize the set of color L, where $|L| = |C|$
4. Construct a SAT formula F using T, C and L
5. Solve F using a SAT solver
6. If F is unsatisfiable add a color to L, goto 4
7. Return the DFA found in step 5

Results


Runtime in seconds
200
150
100
50
0
exbar
ed-beam
all clauses
w/o redundant
naive

Using a SAT subroutine

- Our translation does not require a prefix tree as initial DFA
- Any partially constructed DFA can be used
- Thus we can:
  - Apply a few steps of EDSM
  - Apply our translation to SAT

- Using this approach we were able to solve the smallest of the Abbadingo problems

Conclusions

- We have constructed an efficient translation from DFA learning to SAT
- The translation is based on graph coloring
- We include symmetry breaking predicates and redundant clauses to help the SAT solver
- The result is a simple algorithm for DFA learning that makes use of advanced SAT solving techniques
- This algorithm is competitive with the current state-of-the-art

Future work

- Over 90% of the time is lost due to reading/writing the SAT formula from/to disk
  - We need a tighter coupling between our algorithm and the SAT solver
- Investigate whether it is beneficial to use cliques instead of red states in state-merging
- Try it on the new competition, since alphabet size does not matter much for the size of the translation