Finite automata algorithms:

bisimulations up to congruence, and binary decision diagrams

Damien Pous

Plume group, CNRS, ENS Lyon

Journées SDA2, Marne la Vallée, o8.o4.2o15

Language equivalence of automata

- Model checking
- Program equivalence (cf. D'Antoni's work)
- SDN analysis (cf. NetKAT)
- Formal proof automation (Relational methods, Kleene algebra)

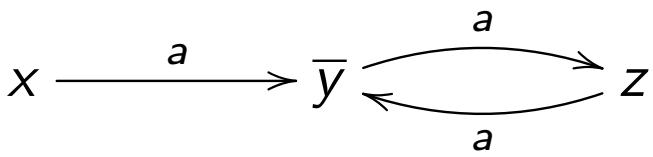
Outline

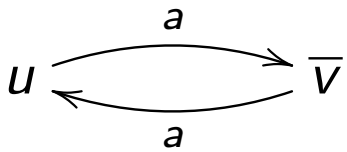
Many states

Many letters

Checking language equivalence of finite automata

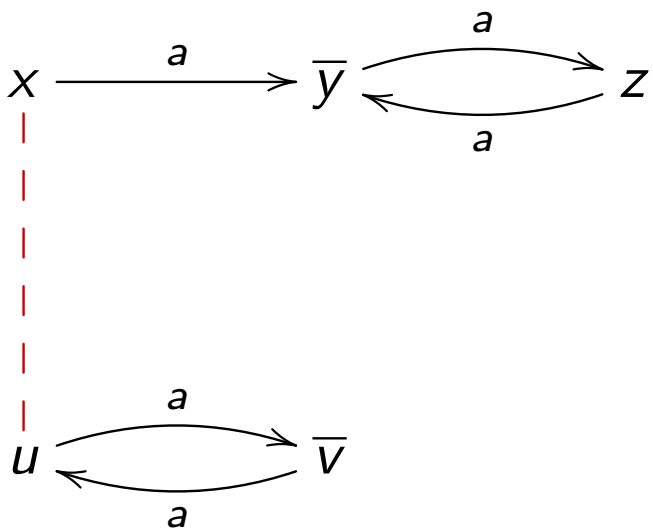
Deterministic case, naive algorithm:





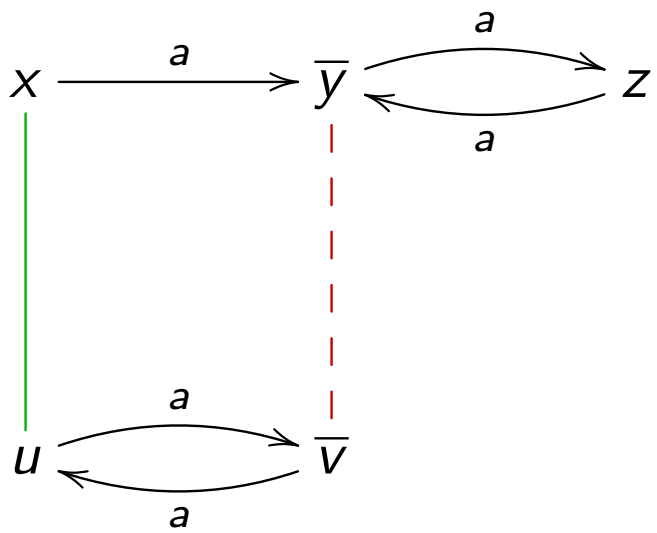
Checking language equivalence of finite automata

Deterministic case, naive algorithm:



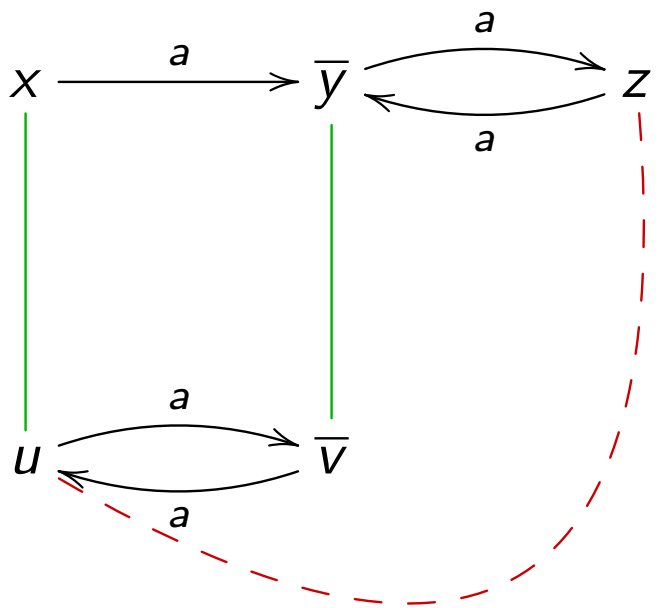
Checking language equivalence of finite automata

Deterministic case, naive algorithm:



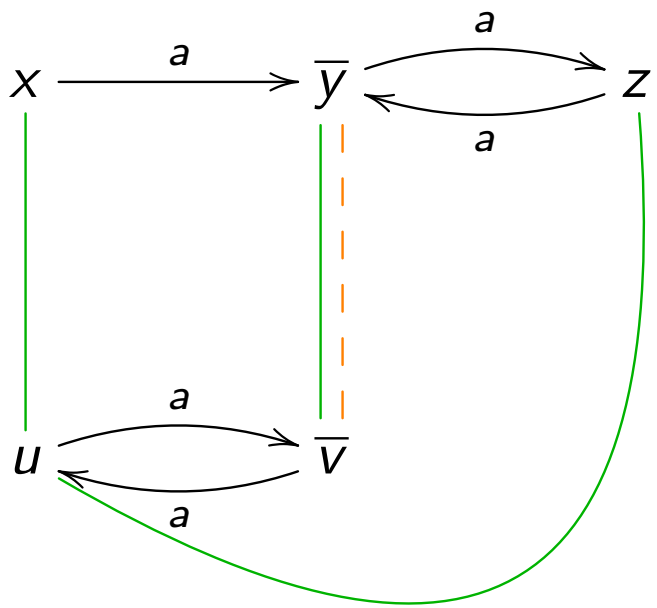
Checking language equivalence of finite automata

Deterministic case, naive algorithm:



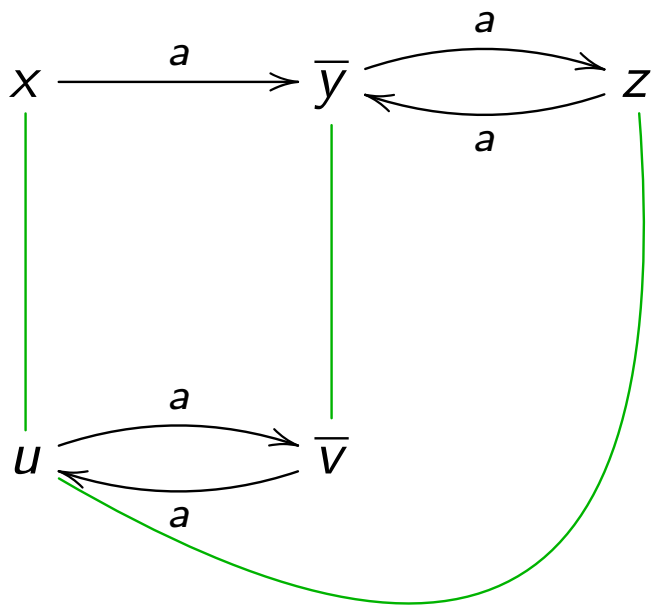
Checking language equivalence of finite automata

Deterministic case, naive algorithm:



Checking language equivalence of finite automata

Deterministic case, naive algorithm:



Checking language equivalence

Deterministic case, naive algorithm, correctness:

bisimulation

Theorem: $x \sim y$ iff there exists a bisimulation R with $x \, R \, y$

Checking language equivalence

Deterministic case, naive algorithm, correctness:

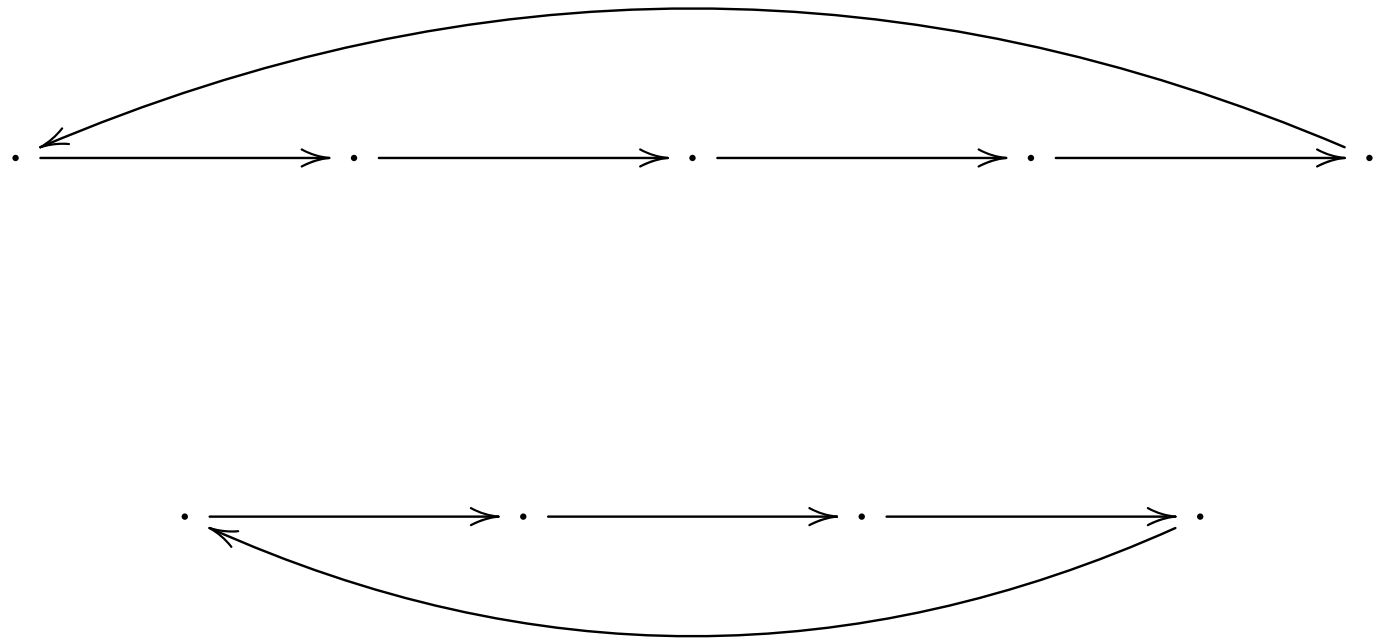
bisimulation

Theorem: $x \sim y$ iff there exists a bisimulation R with $x \, R \, y$

The previous algorithm attempts to construct a bisimulation

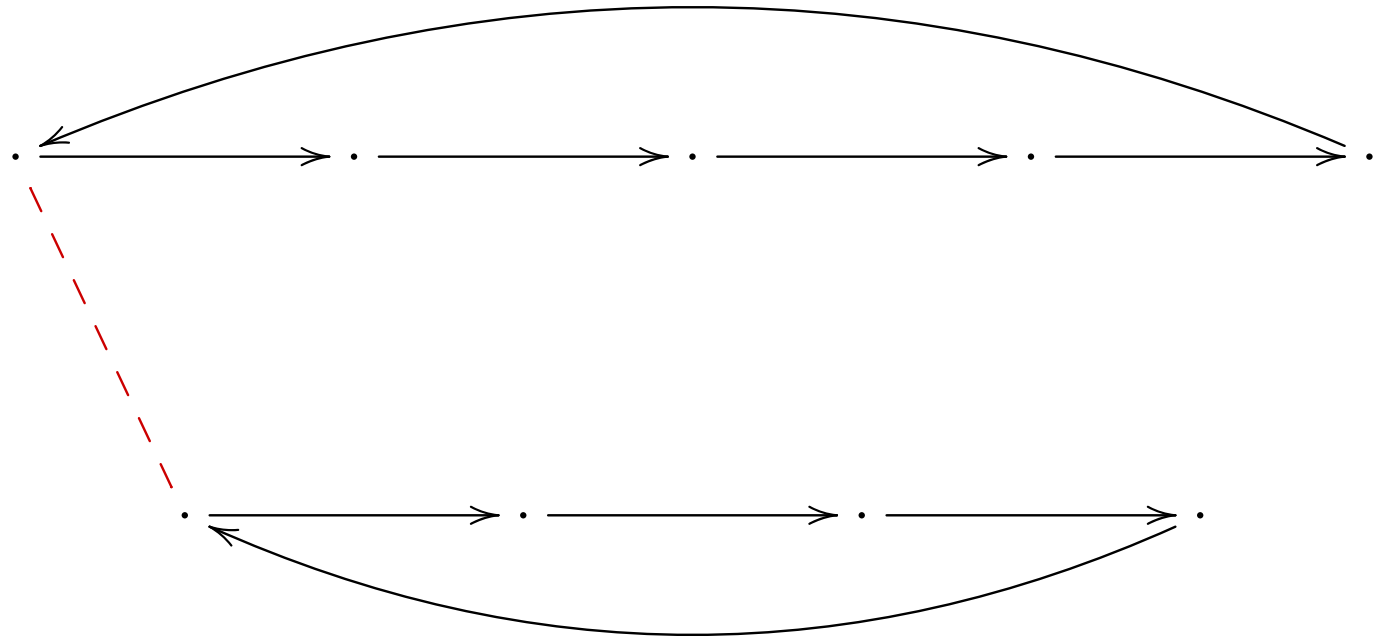
Checking language equivalence

Deterministic case, naive algorithm: quadratic complexity



Checking language equivalence

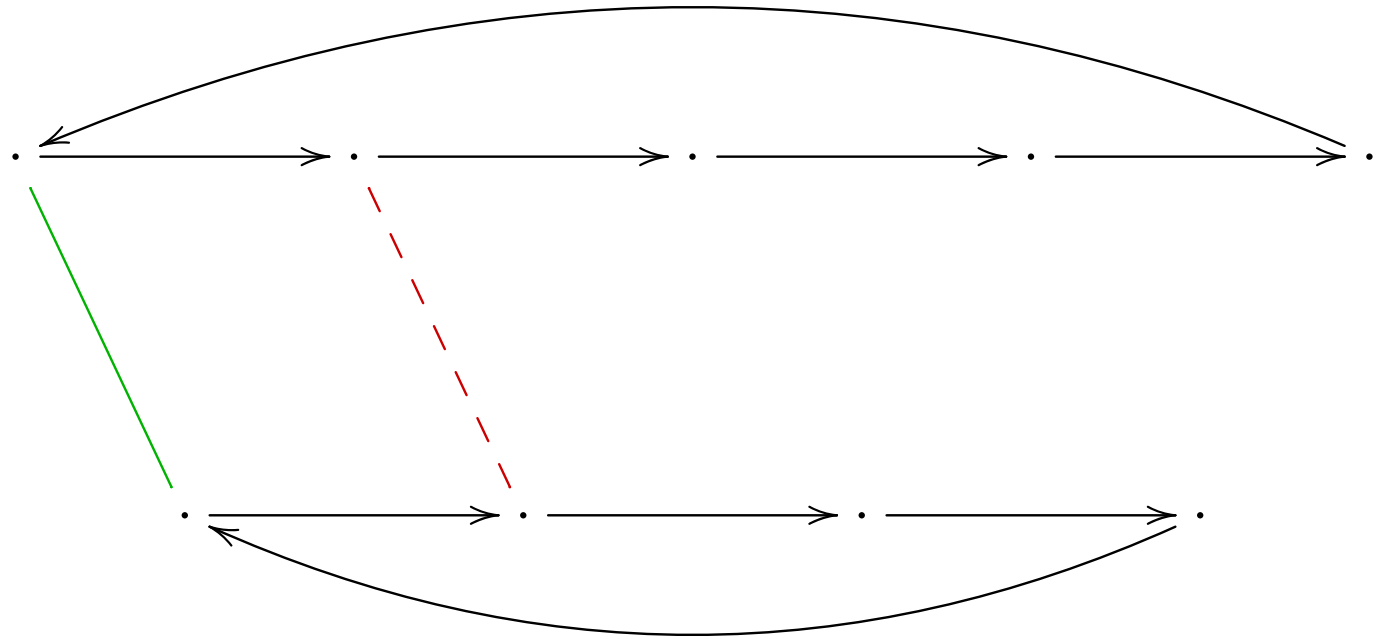
Deterministic case, naive algorithm: quadratic complexity



1 pairs

Checking language equivalence

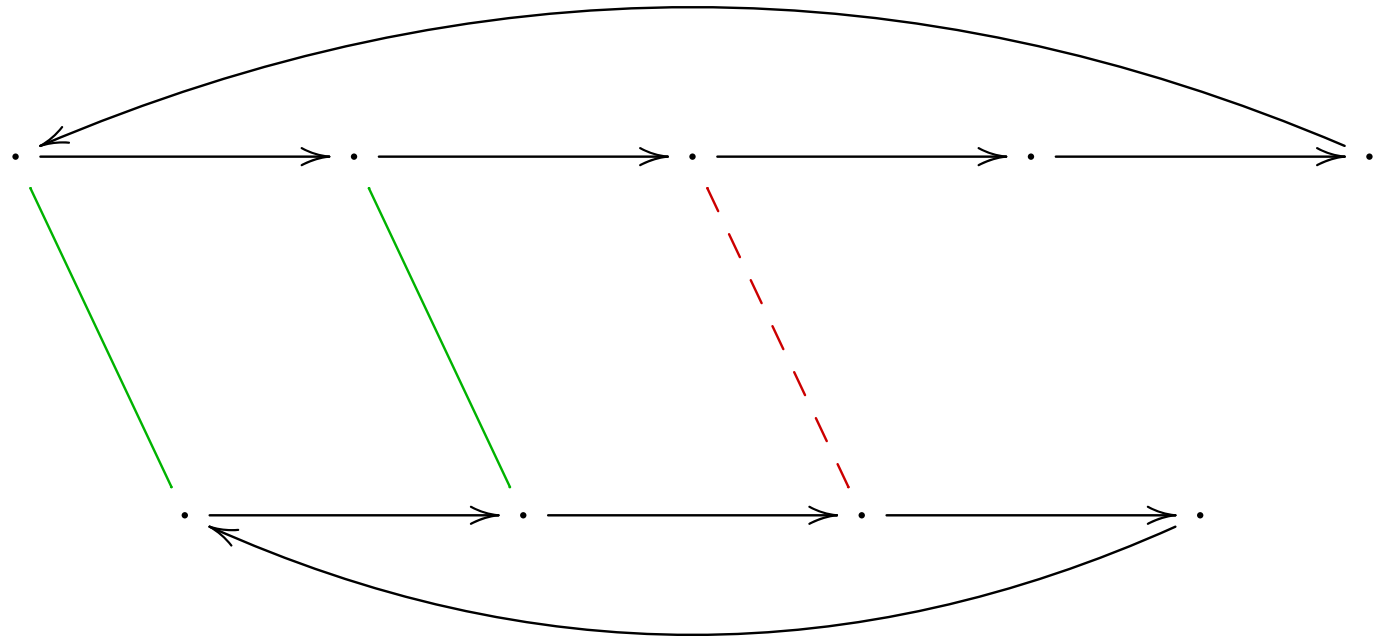
Deterministic case, naive algorithm: quadratic complexity



2 pairs

Checking language equivalence

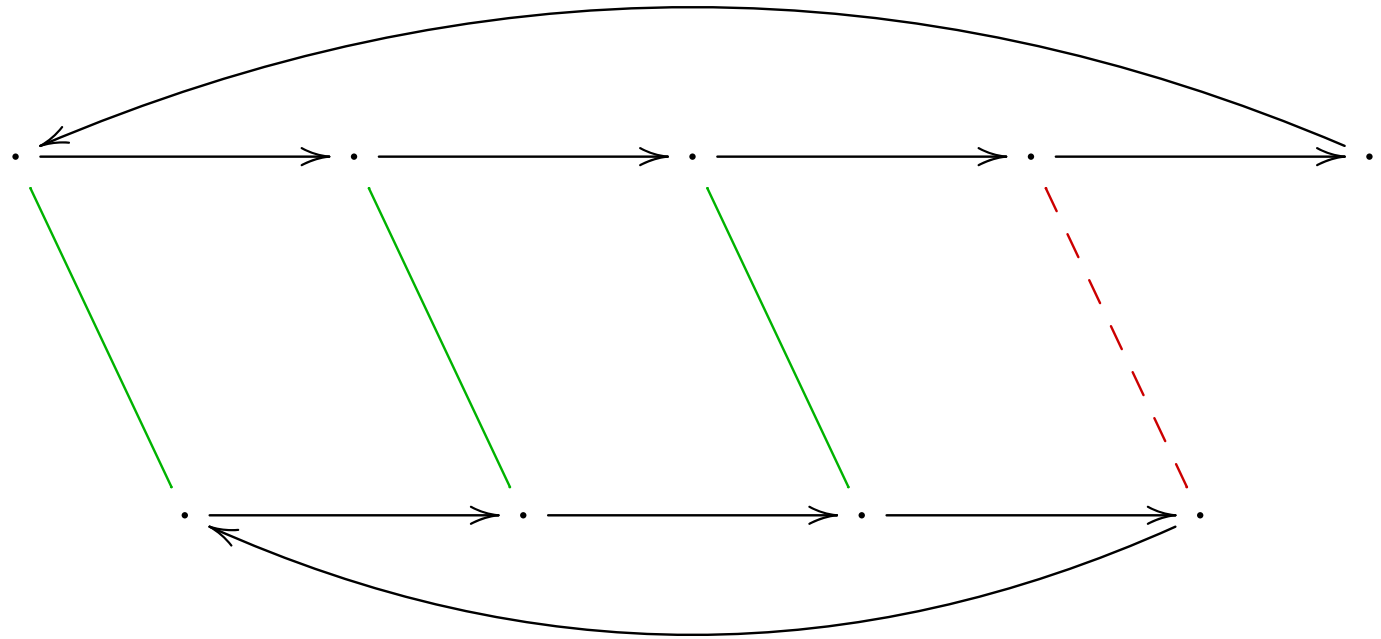
Deterministic case, naive algorithm: quadratic complexity



3 pairs

Checking language equivalence

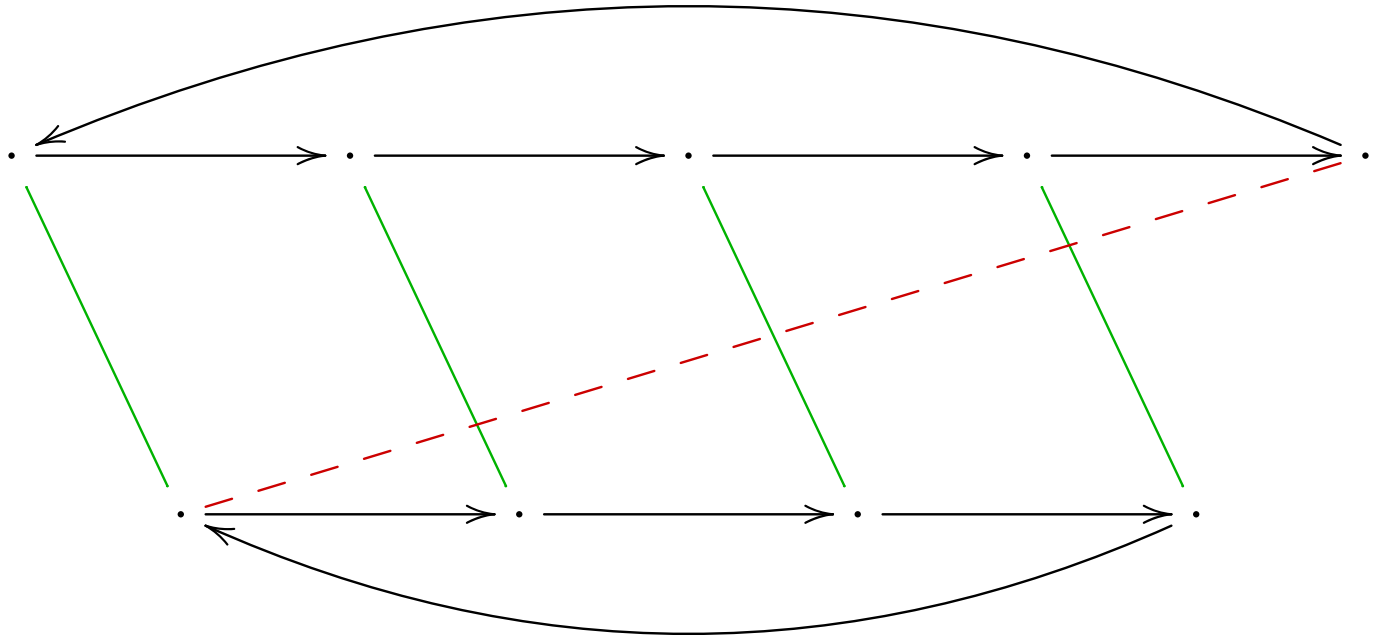
Deterministic case, naive algorithm: quadratic complexity



4 pairs

Checking language equivalence

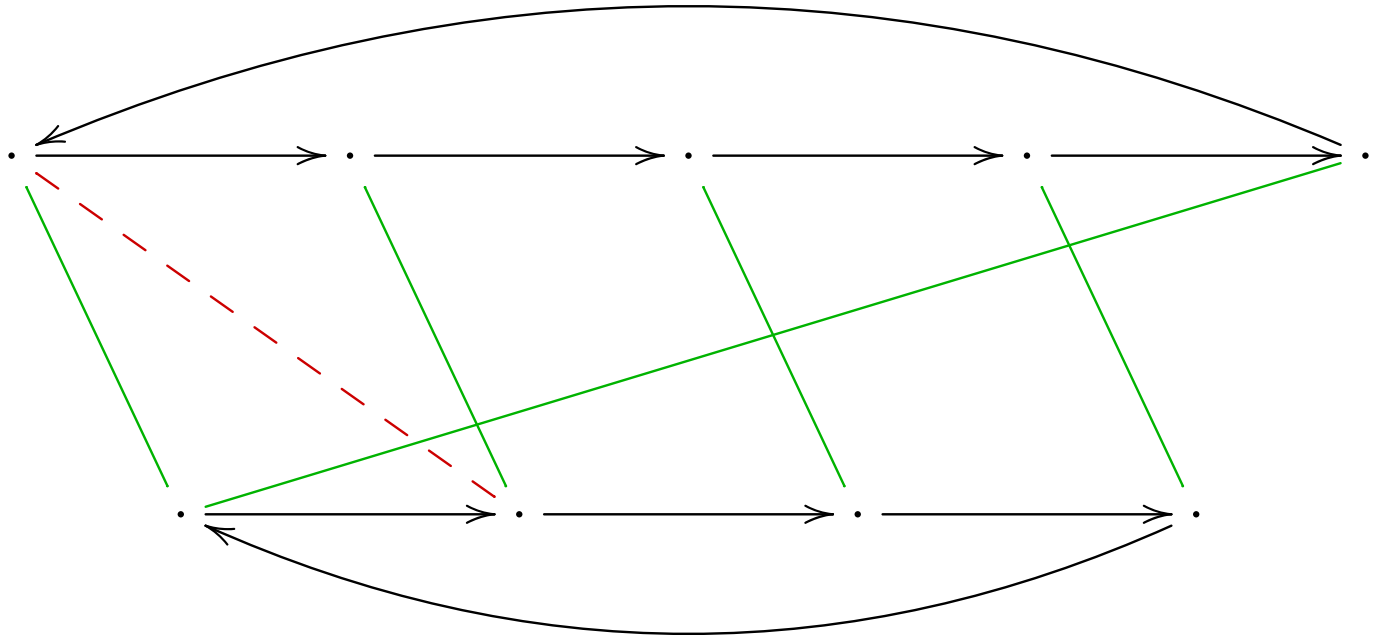
Deterministic case, naive algorithm: quadratic complexity



5 pairs

Checking language equivalence

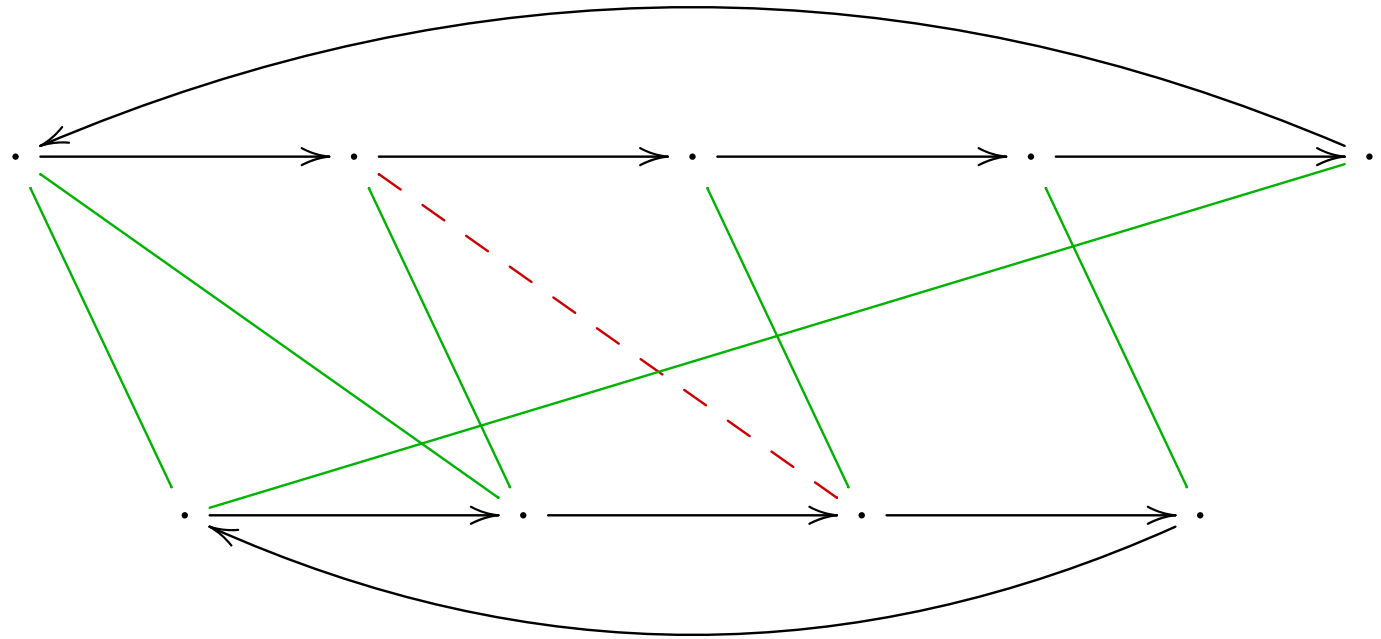
Deterministic case, naive algorithm: quadratic complexity



6 pairs

Checking language equivalence

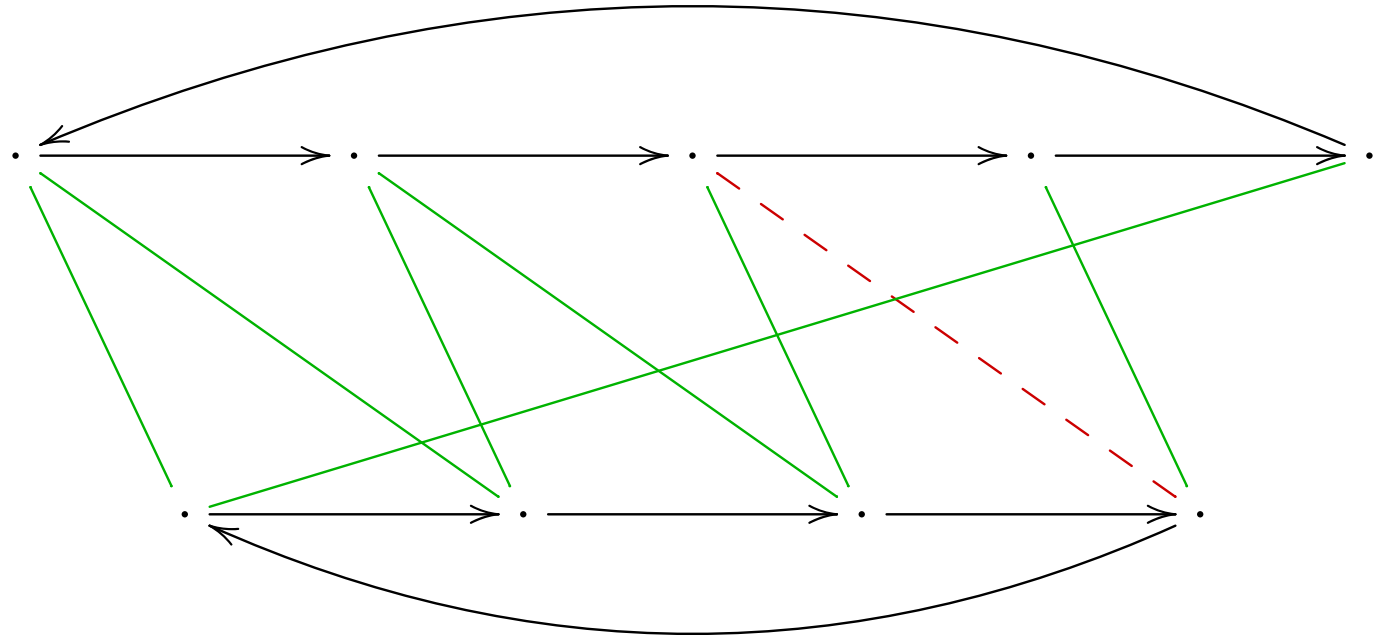
Deterministic case, naive algorithm: quadratic complexity



7 pairs

Checking language equivalence

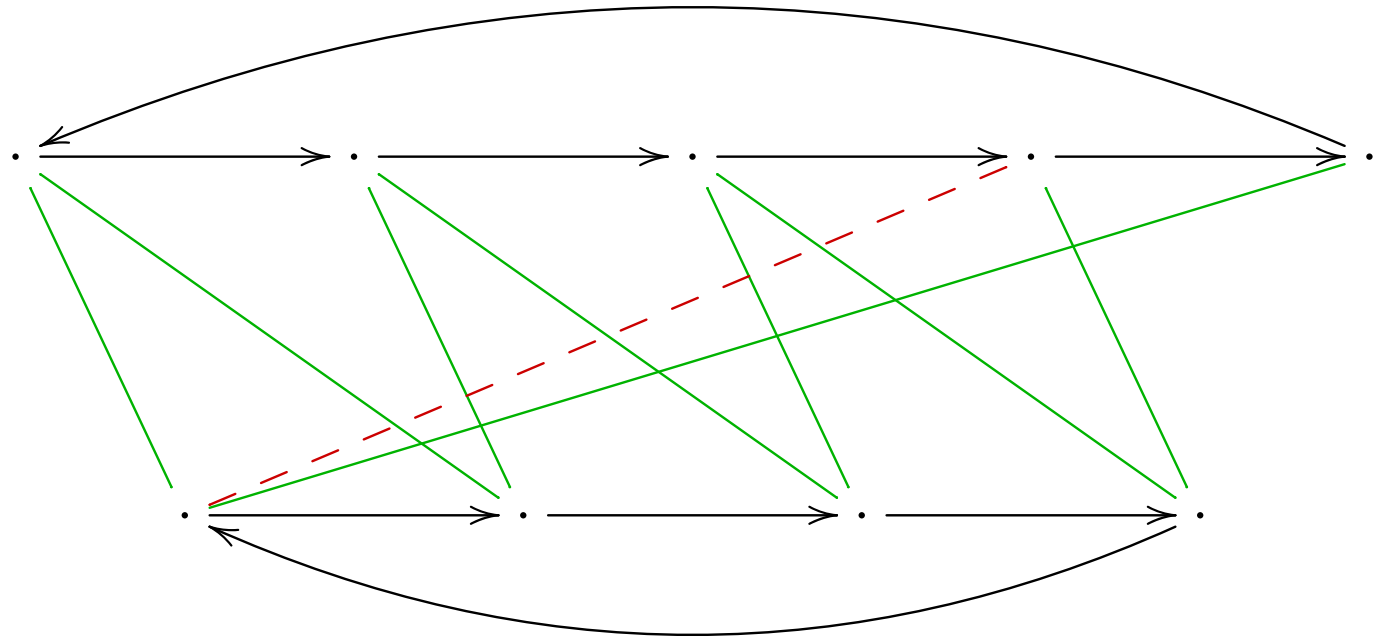
Deterministic case, naive algorithm: quadratic complexity



8 pairs

Checking language equivalence

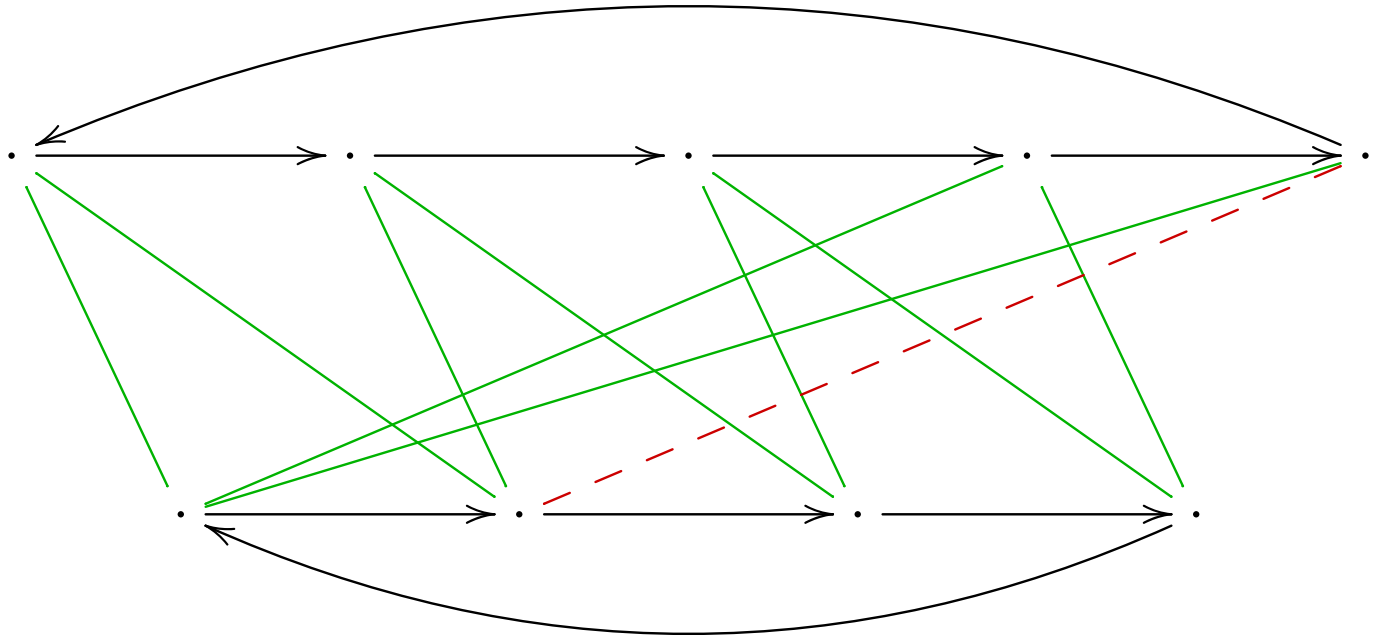
Deterministic case, naive algorithm: quadratic complexity



9 pairs

Checking language equivalence

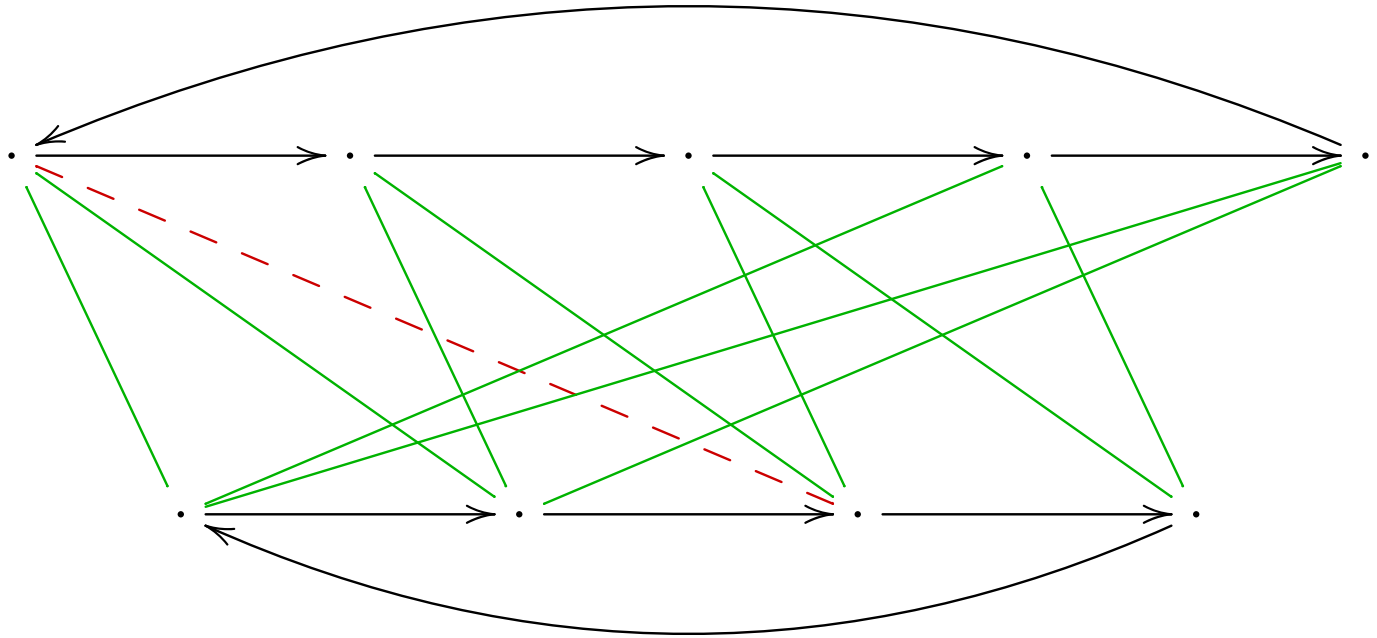
Deterministic case, naive algorithm: quadratic complexity



10 pairs

Checking language equivalence

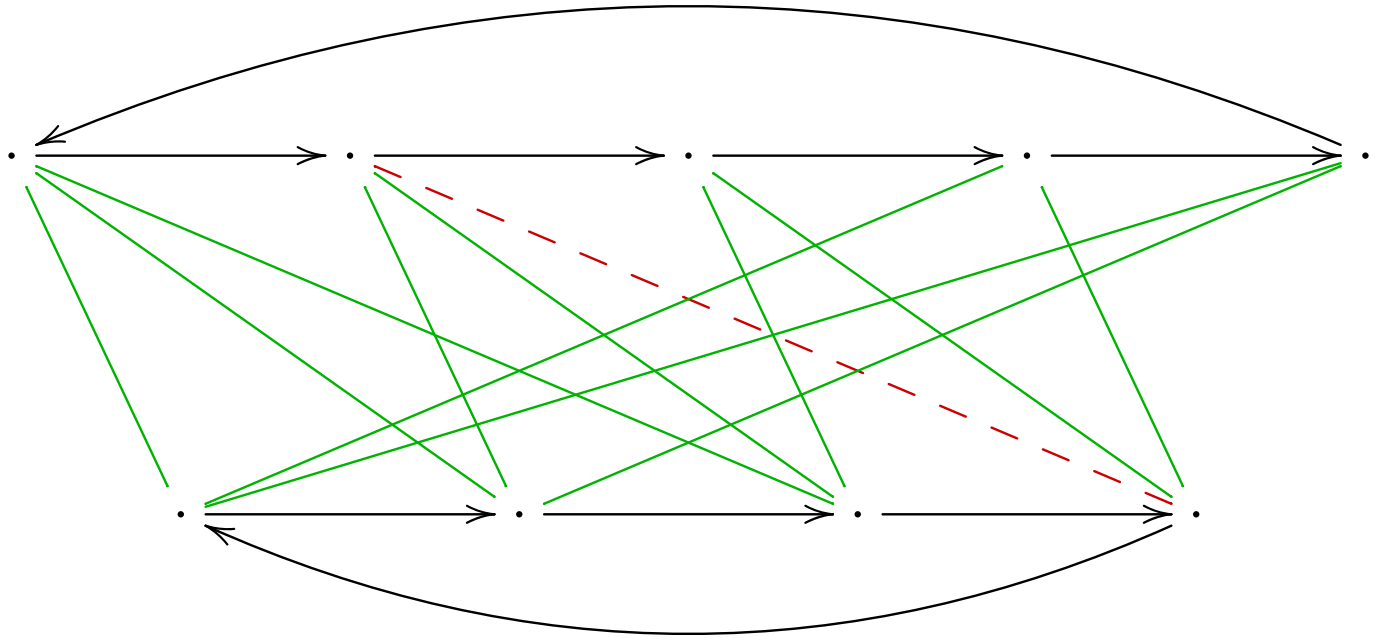
Deterministic case, naive algorithm: quadratic complexity



11 pairs

Checking language equivalence

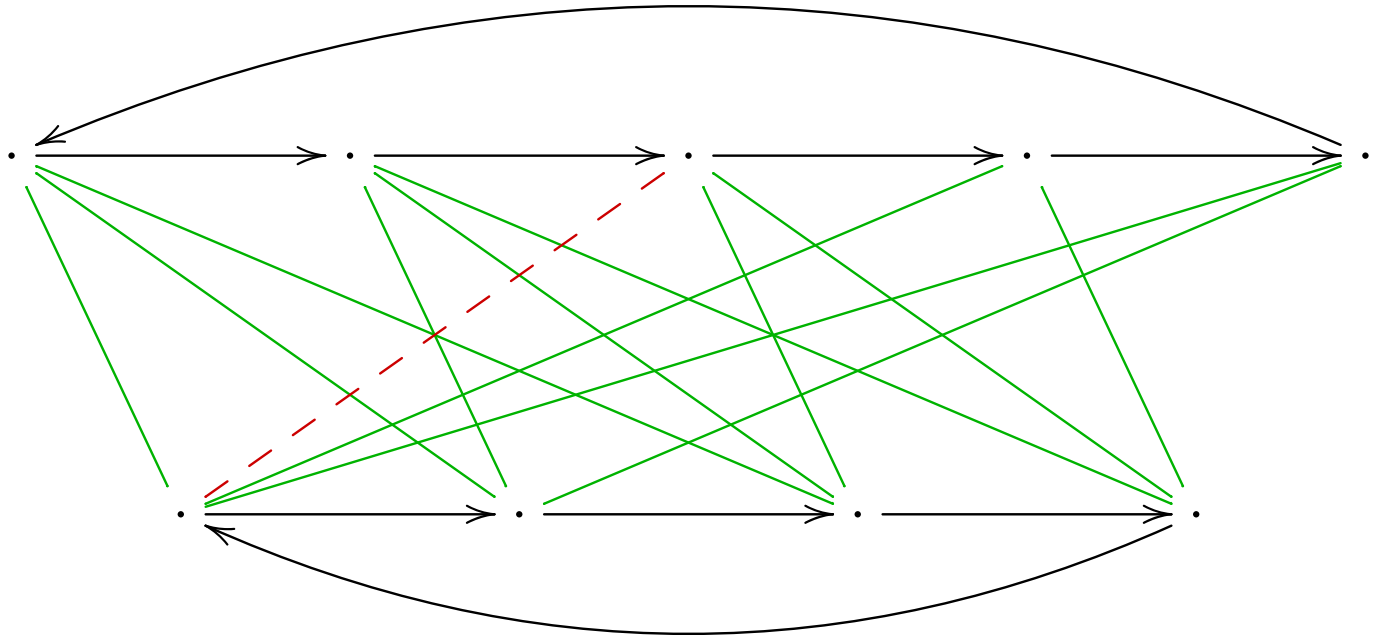
Deterministic case, naive algorithm: quadratic complexity



12 pairs

Checking language equivalence

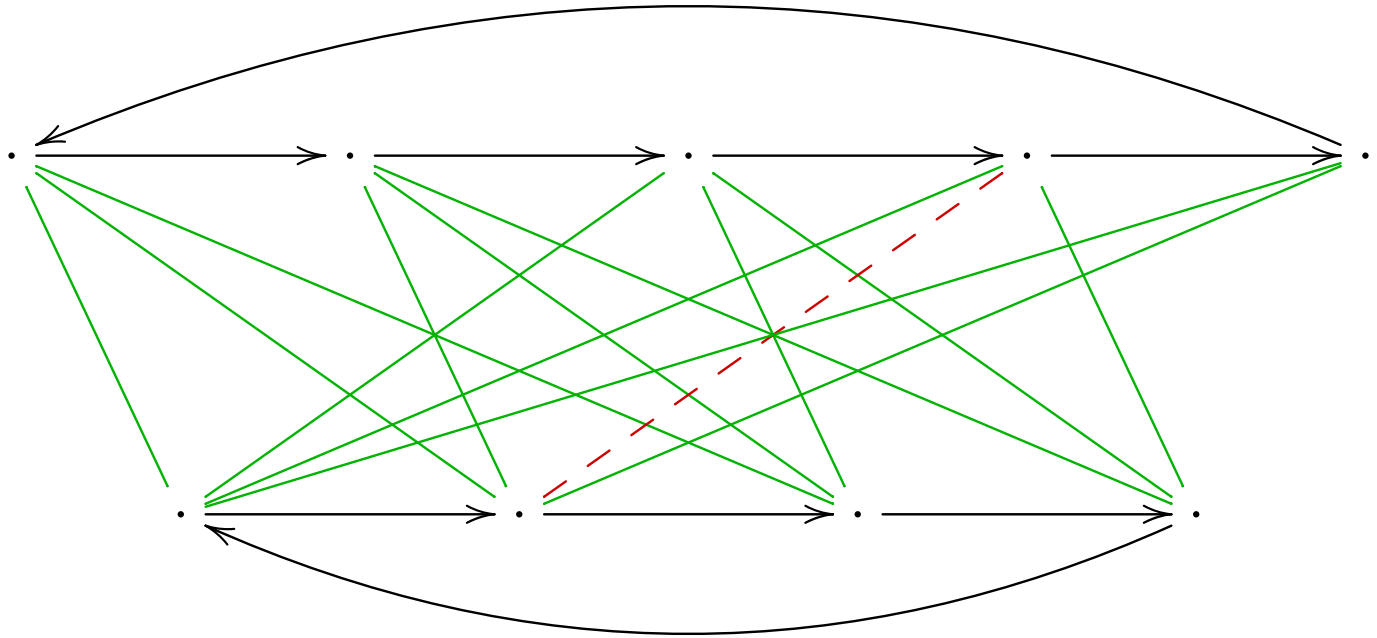
Deterministic case, naive algorithm: quadratic complexity



13 pairs

Checking language equivalence

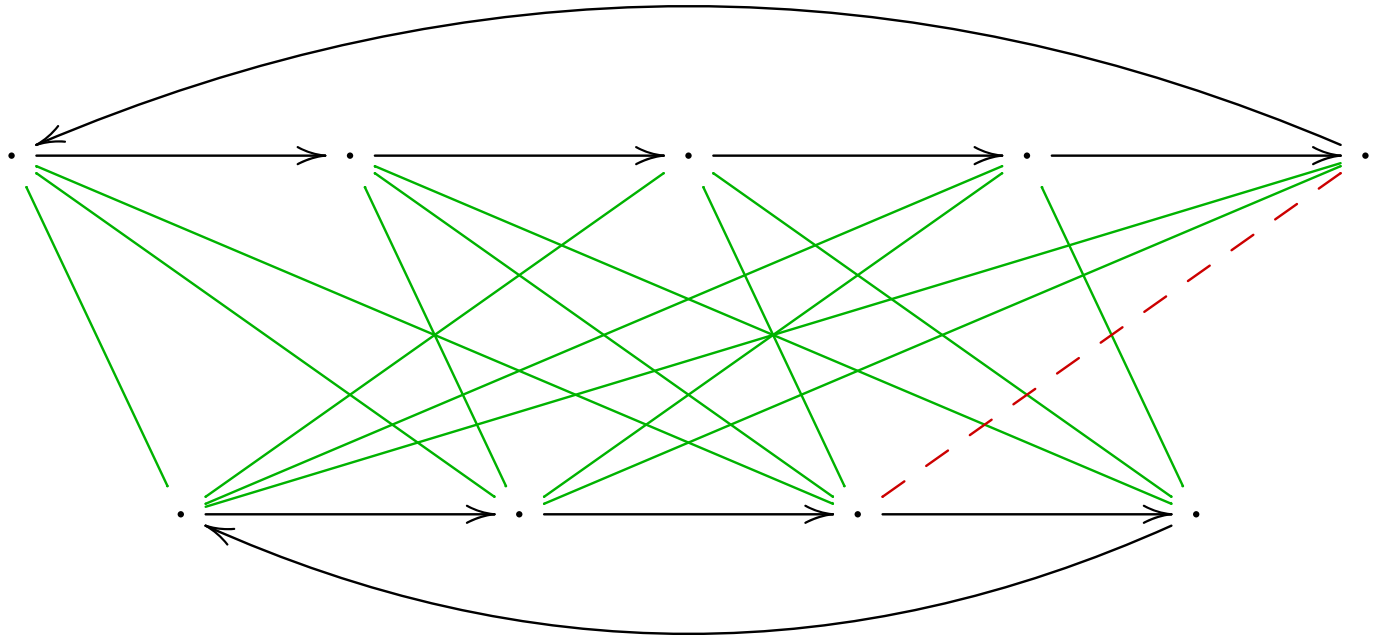
Deterministic case, naive algorithm: quadratic complexity



14 pairs

Checking language equivalence

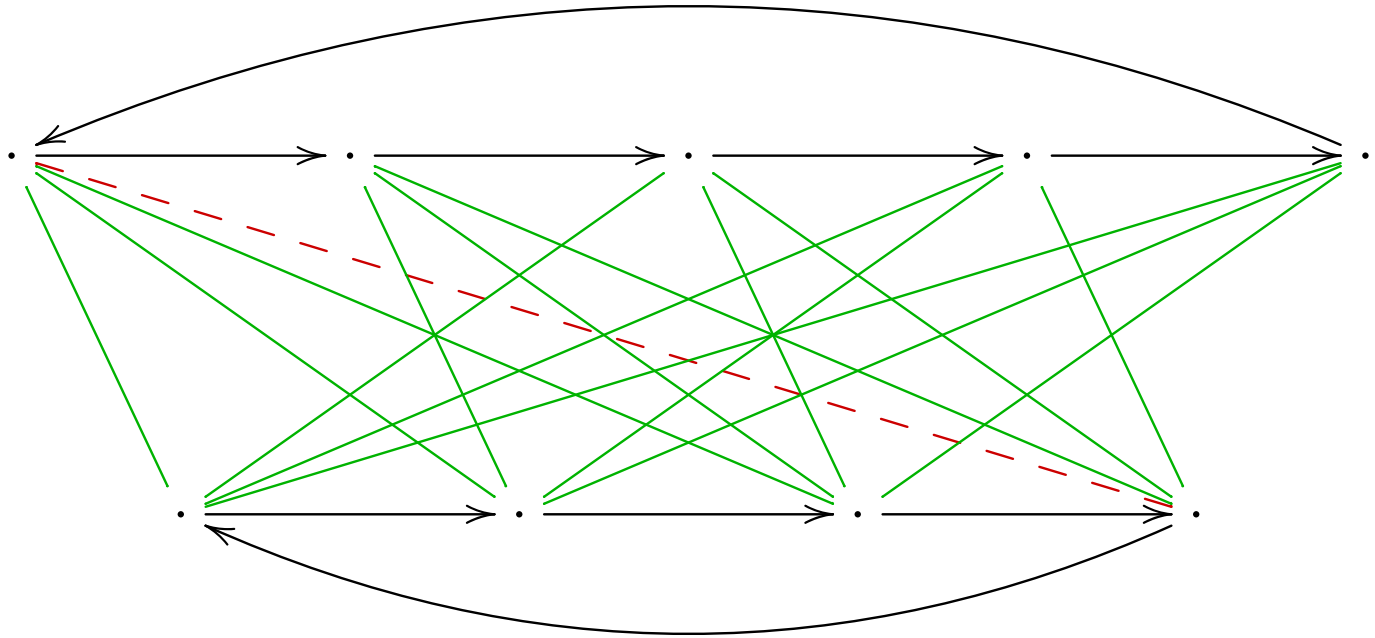
Deterministic case, naive algorithm: quadratic complexity



15 pairs

Checking language equivalence

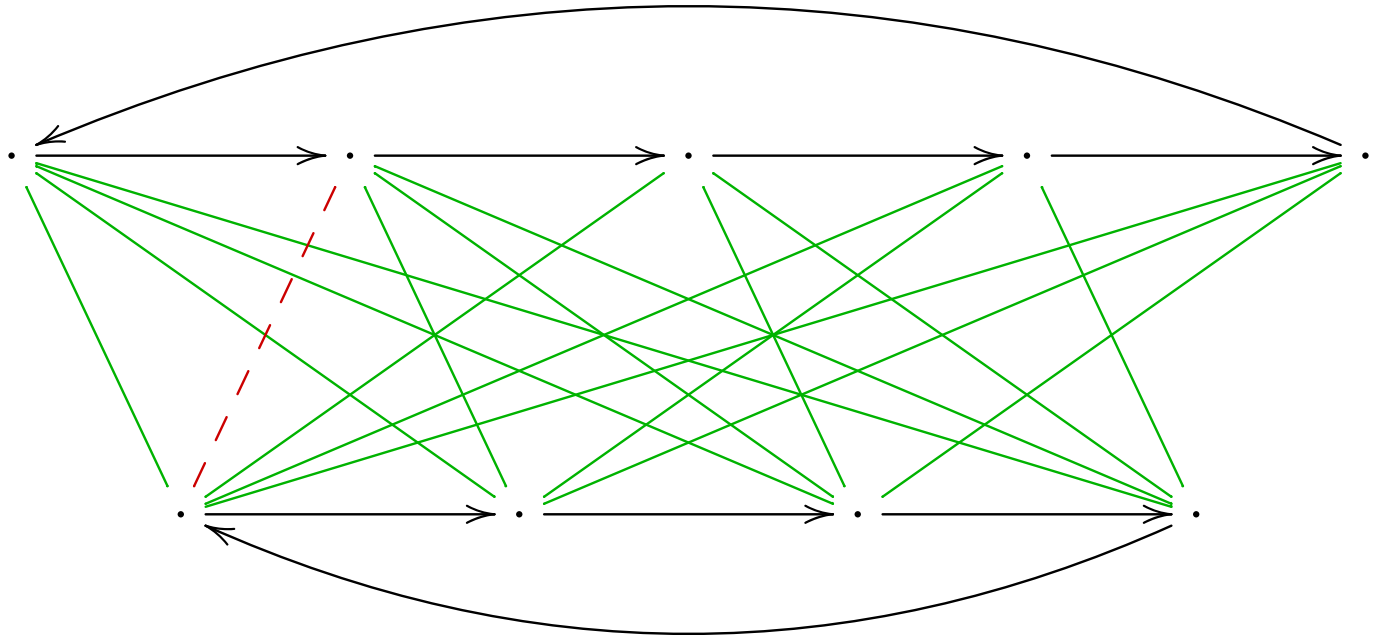
Deterministic case, naive algorithm: quadratic complexity



16 pairs

Checking language equivalence

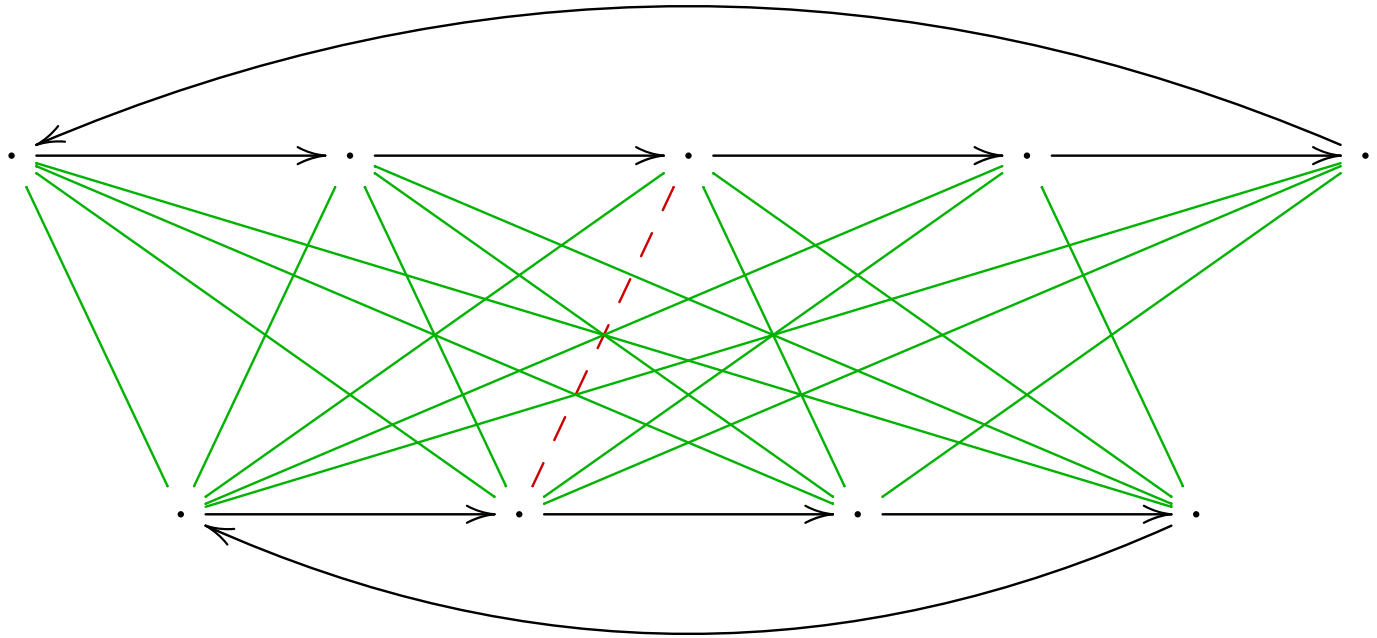
Deterministic case, naive algorithm: quadratic complexity



17 pairs

Checking language equivalence

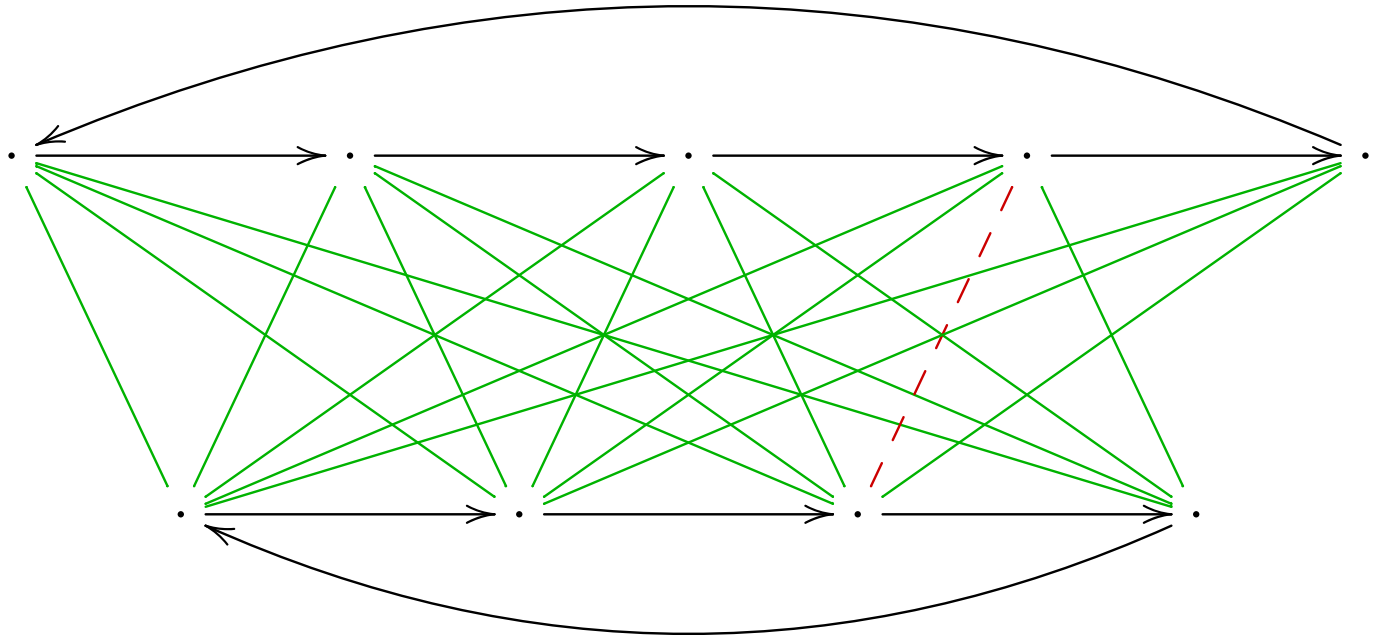
Deterministic case, naive algorithm: quadratic complexity



18 pairs

Checking language equivalence

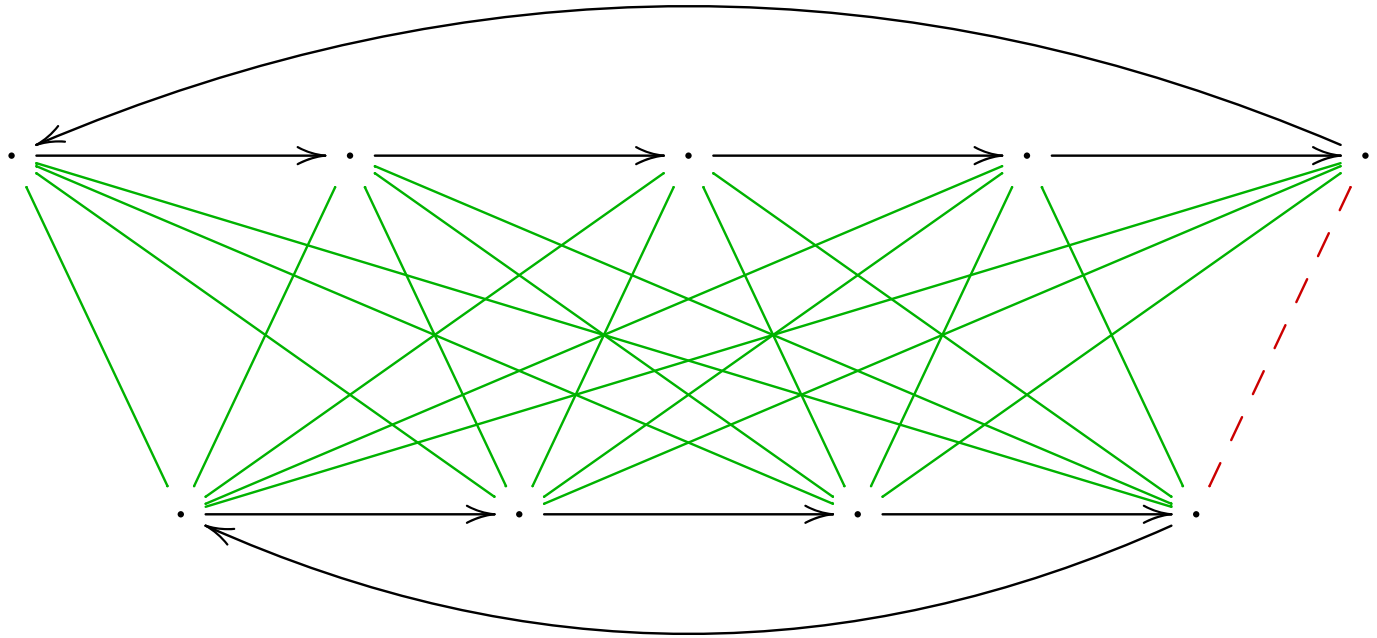
Deterministic case, naive algorithm: quadratic complexity



19 pairs

Checking language equivalence

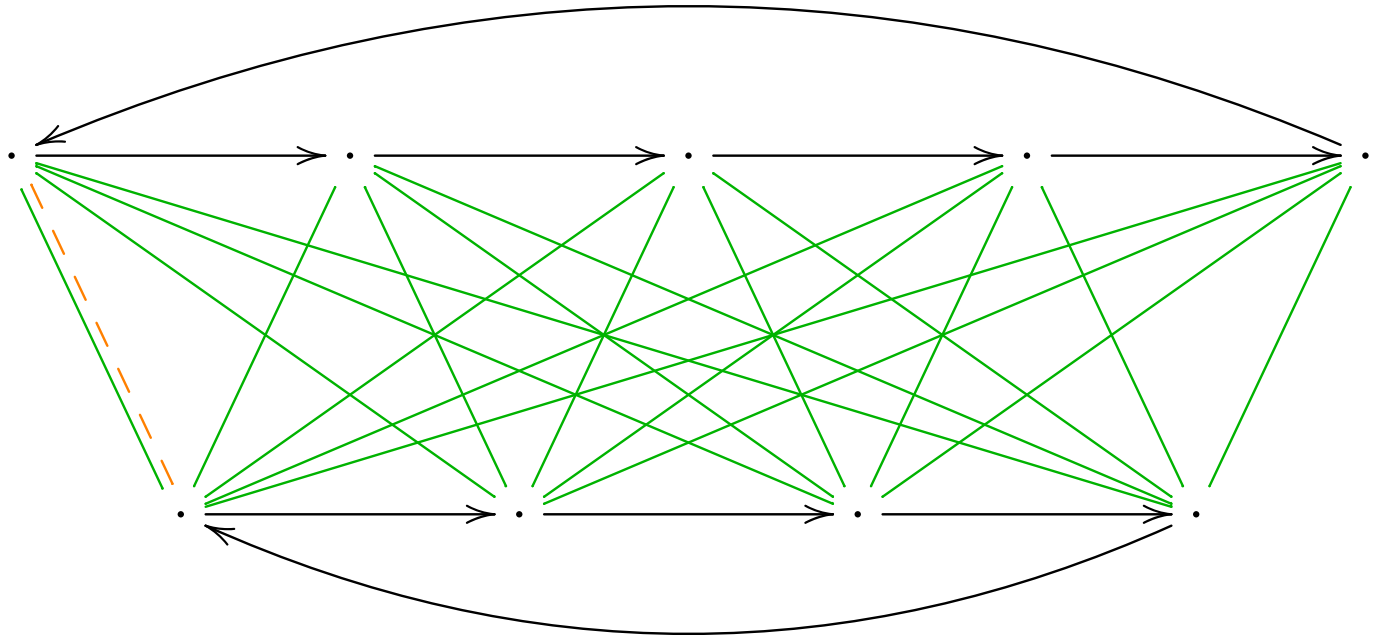
Deterministic case, naive algorithm: quadratic complexity



20 pairs

Checking language equivalence

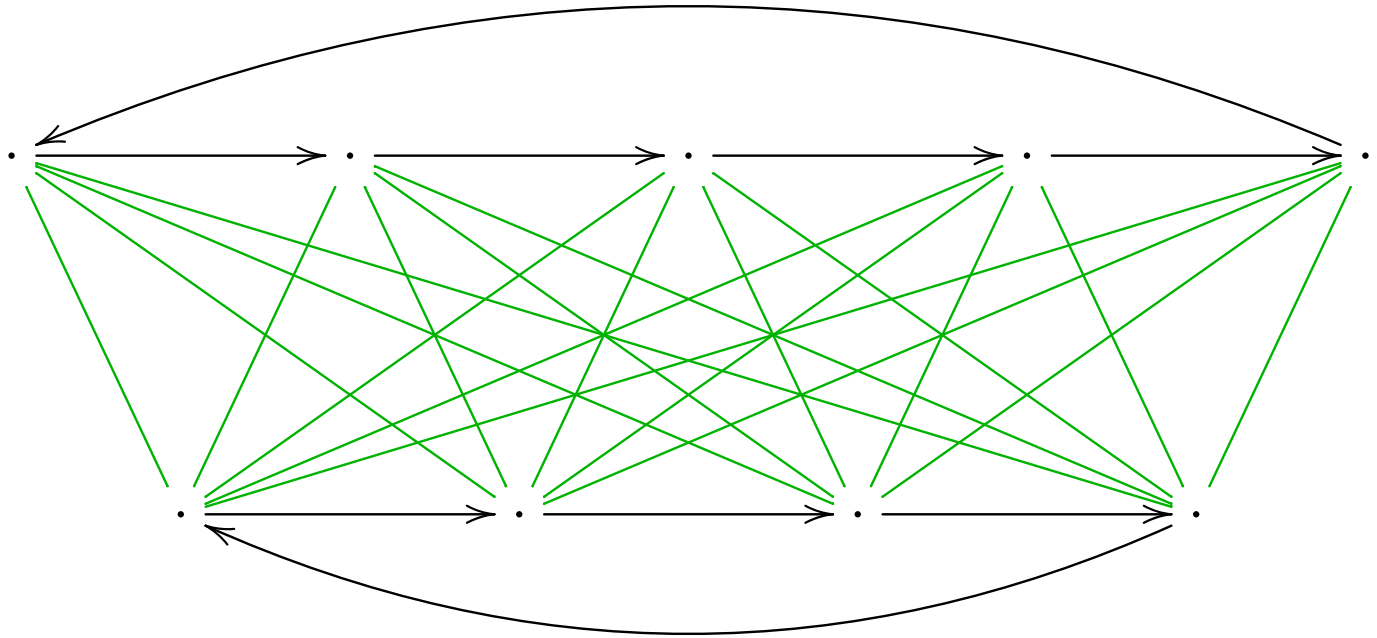
Deterministic case, naive algorithm: quadratic complexity



20 pairs

Checking language equivalence

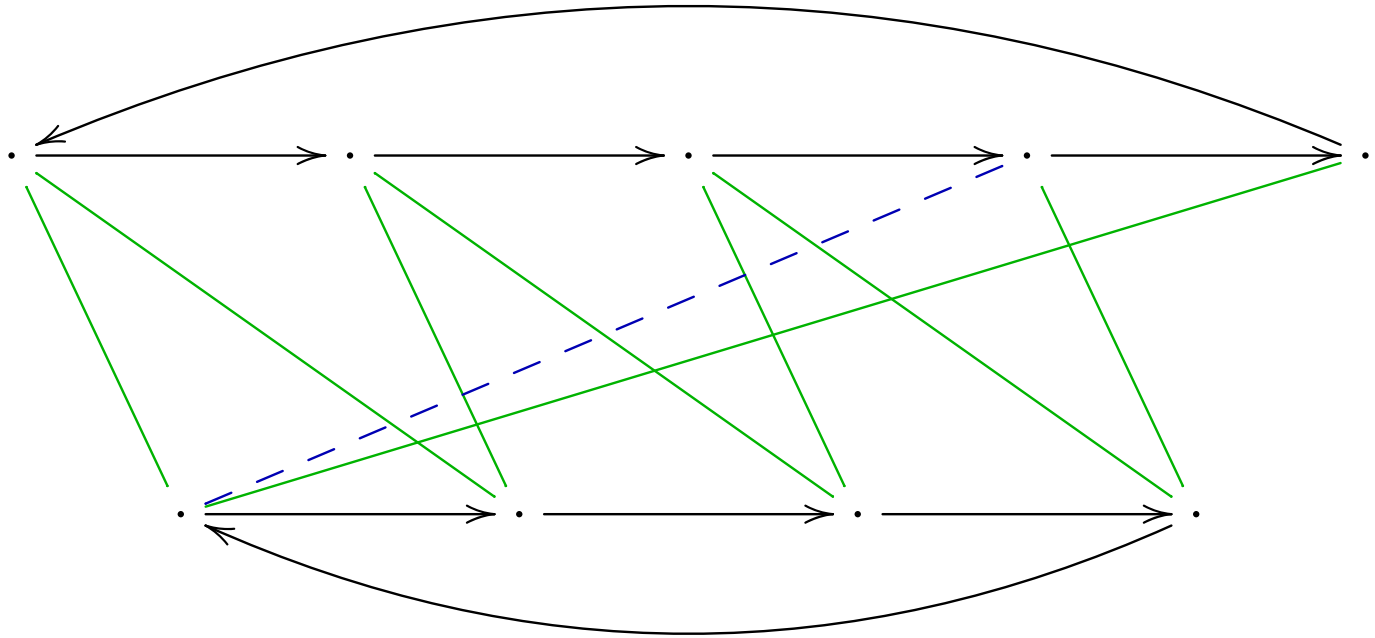
Deterministic case, naive algorithm: quadratic complexity



20 pairs

Checking language equivalence

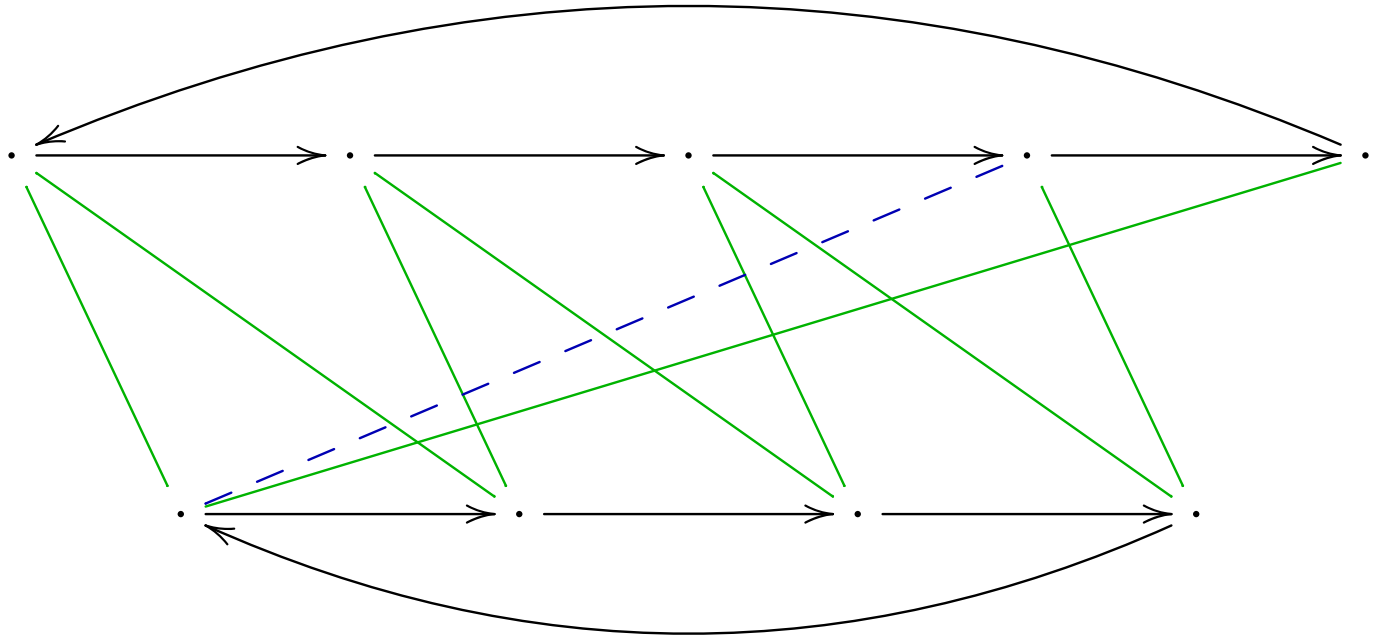
One can stop much earlier



~~20~~ 8 pairs

Checking language equivalence

One can stop much earlier



[Hopcroft and Karp '71]
[Tarjan '75]

Complexity: almost linear

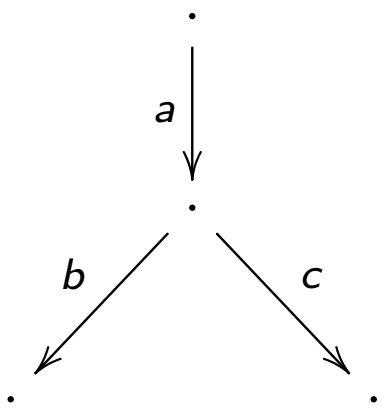
Checking language equivalence

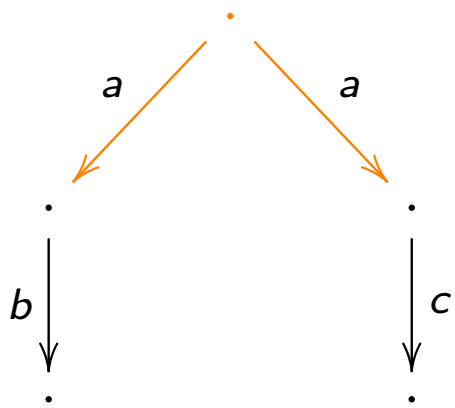
Correctness of HK algorithm, revisited:

bisimulation up to equivalence

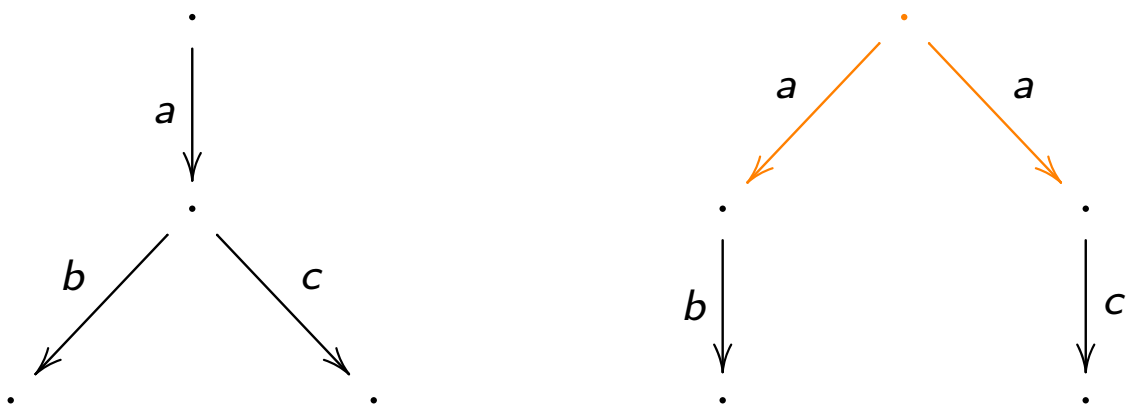
Lemma: if R is a bisimulation up to equivalence,
then R^e is a bisimulation

Non-Deterministic Automata





Non-Deterministic Automata

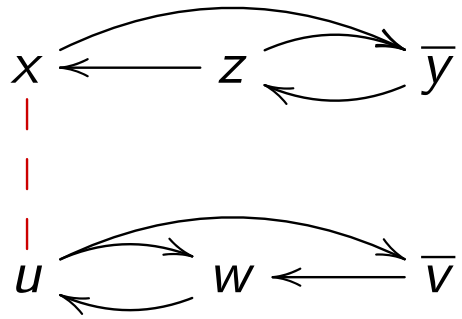


Reduction to the deterministic case:

- "powerset construction": $(S, t, o) \mapsto (\mathcal{P}(S), t^{\#}, o^{\#})$
- from states $(x, y, \dots)$ to sets of states $(X, Y, \dots)$

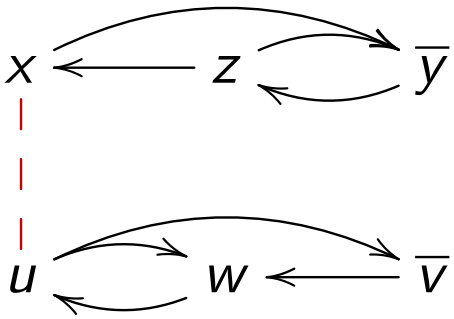
Checking language equivalence

Non-deterministic case: use Hopcroft and Karp on the fly:



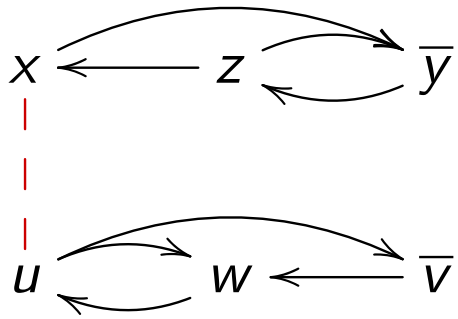
Checking language equivalence

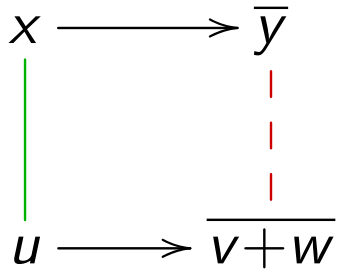
Non-deterministic case: use Hopcroft and Karp on the fly:



Checking language equivalence

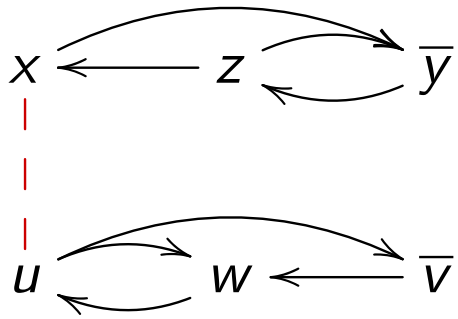
Non-deterministic case: use Hopcroft and Karp on the fly:

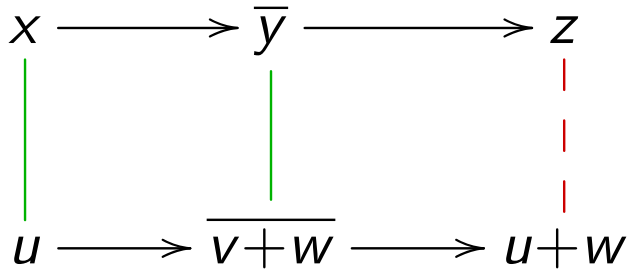




Checking language equivalence

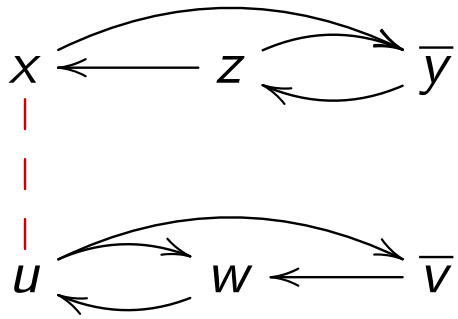
Non-deterministic case: use Hopcroft and Karp on the fly:

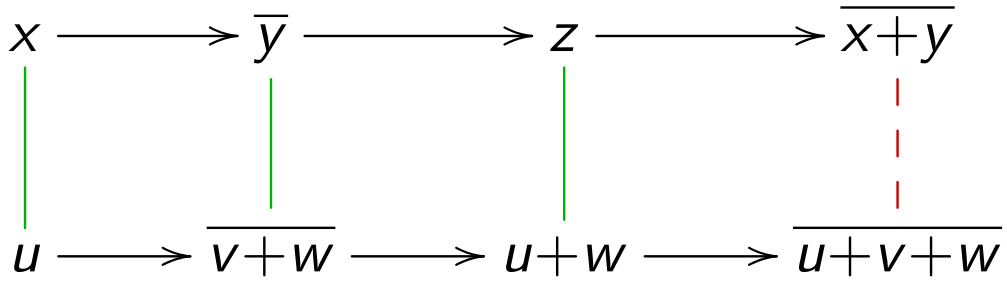




Checking language equivalence

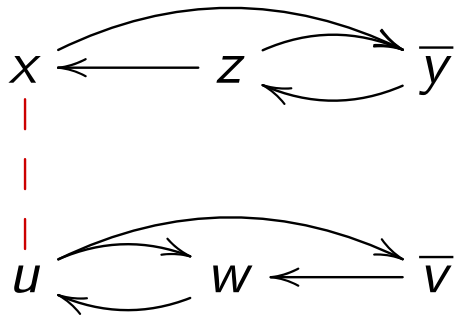
Non-deterministic case: use Hopcroft and Karp on the fly:

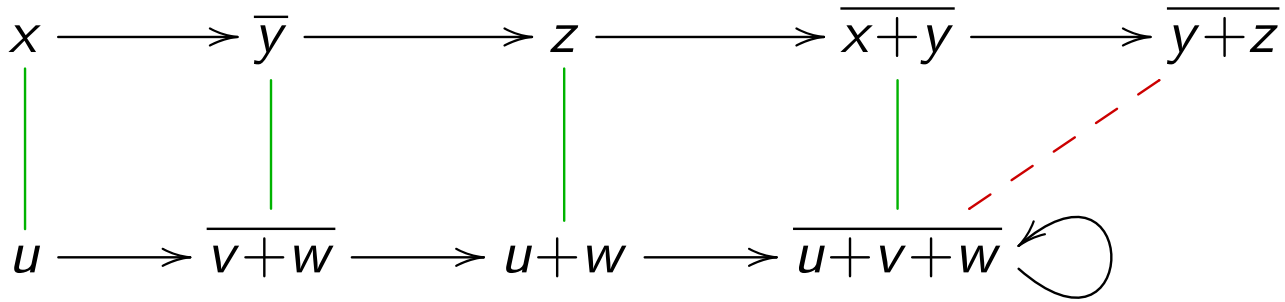




Checking language equivalence

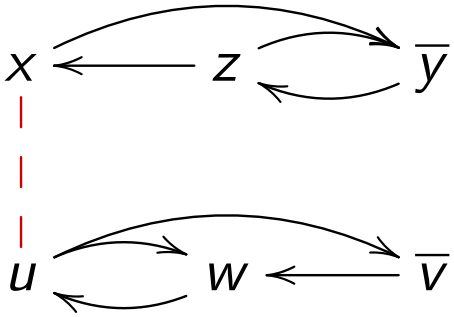
Non-deterministic case: use Hopcroft and Karp on the fly:

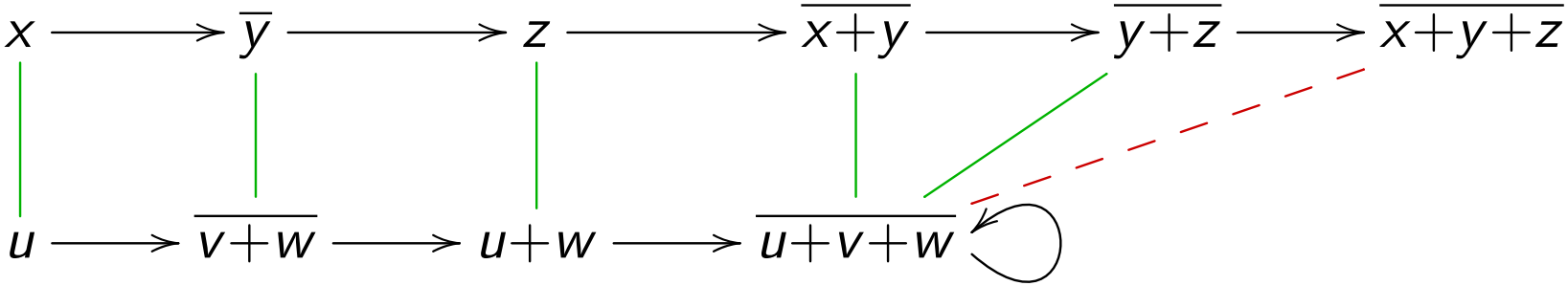




Checking language equivalence

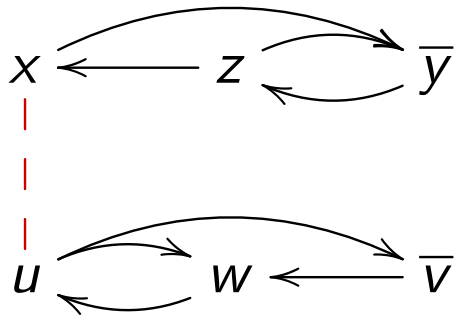
Non-deterministic case: use Hopcroft and Karp on the fly:

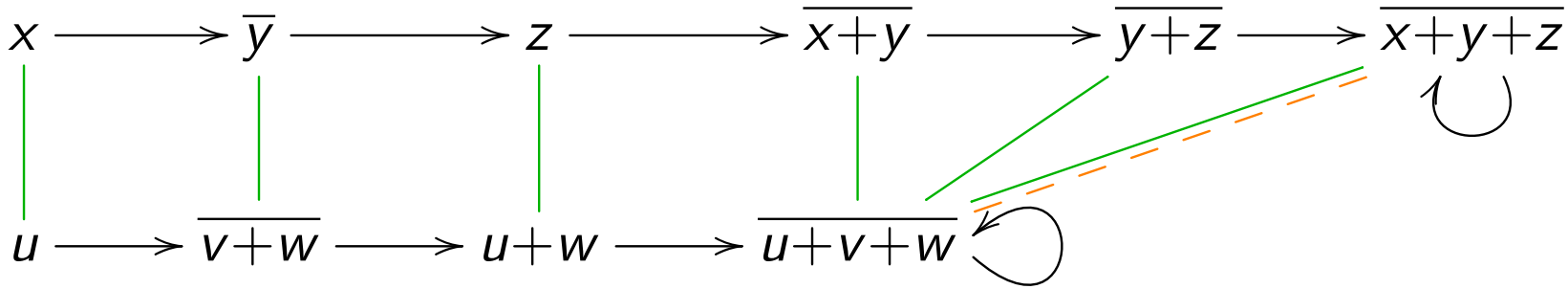




Checking language equivalence

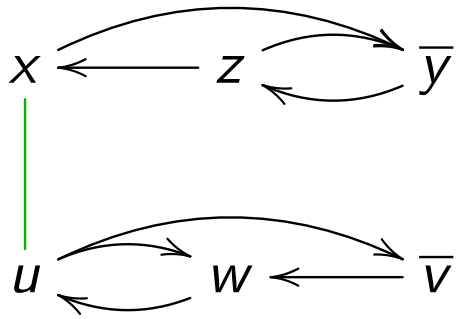
Non-deterministic case: use Hopcroft and Karp on the fly:

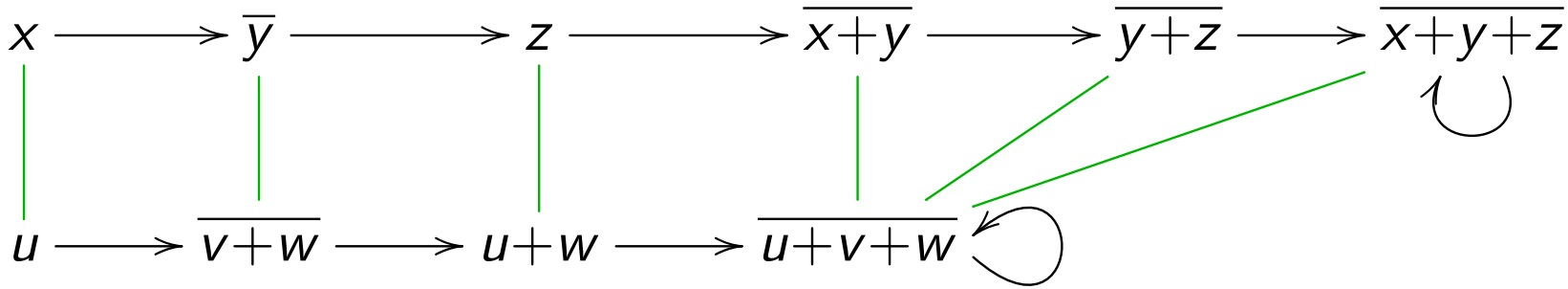




Checking language equivalence

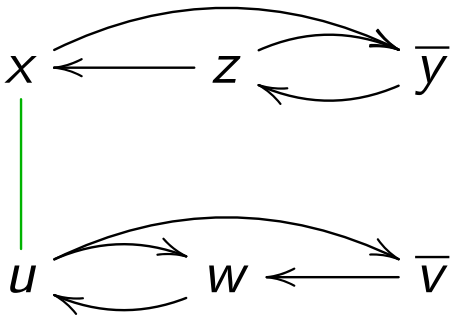
Non-deterministic case: use Hopcroft and Karp on the fly:

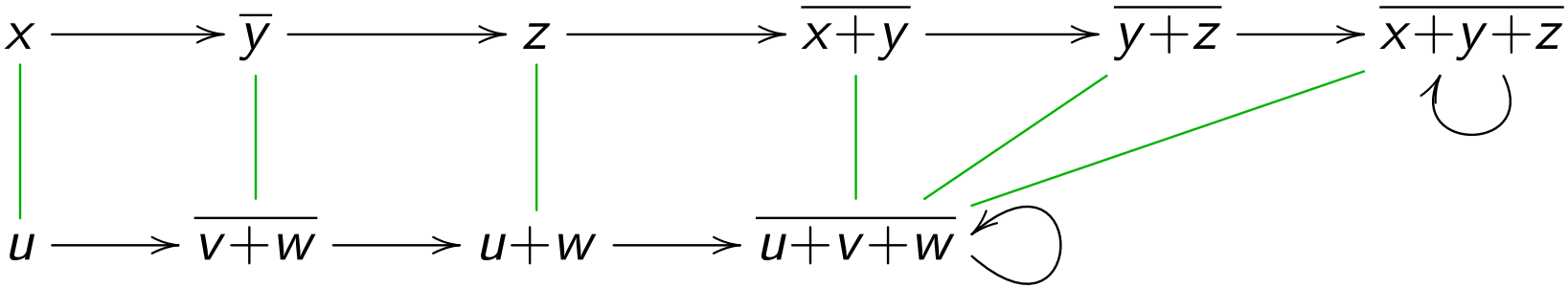




Checking language equivalence

Non-deterministic case: use Hopcroft and Karp on the fly:

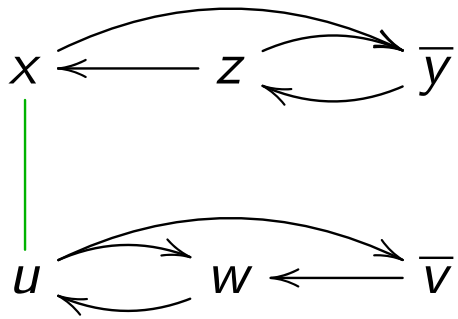


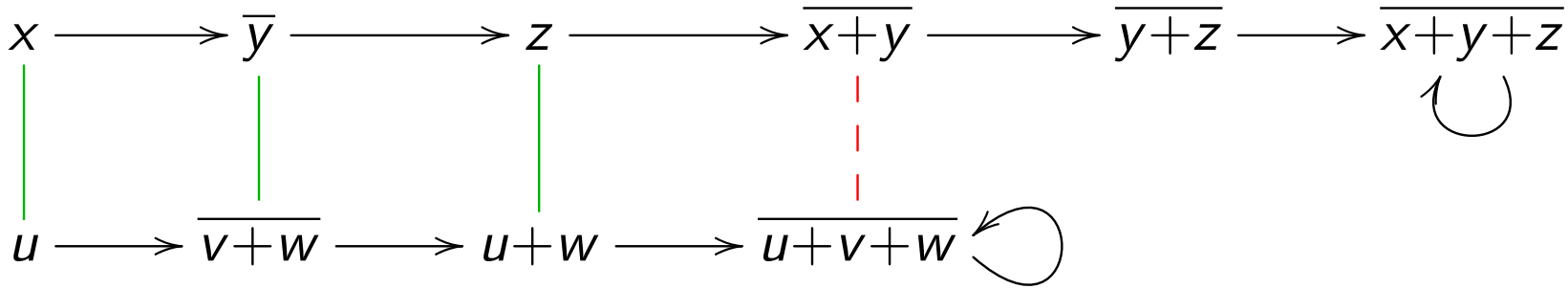


(correctness comes for free)

Checking language equivalence

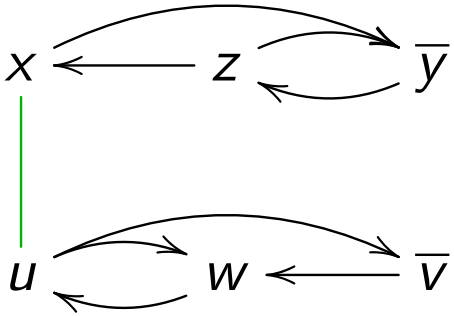
One can do better:



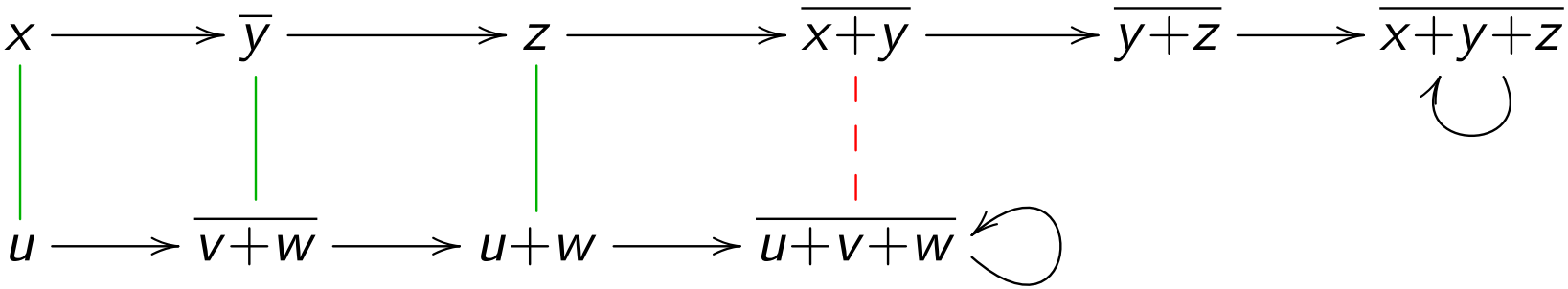


Checking language equivalence

One can do better:

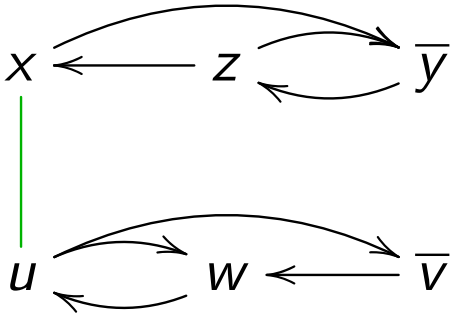


$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v+w) \\ \hline = & (x+y,\ u+v+w) \end{array}$$

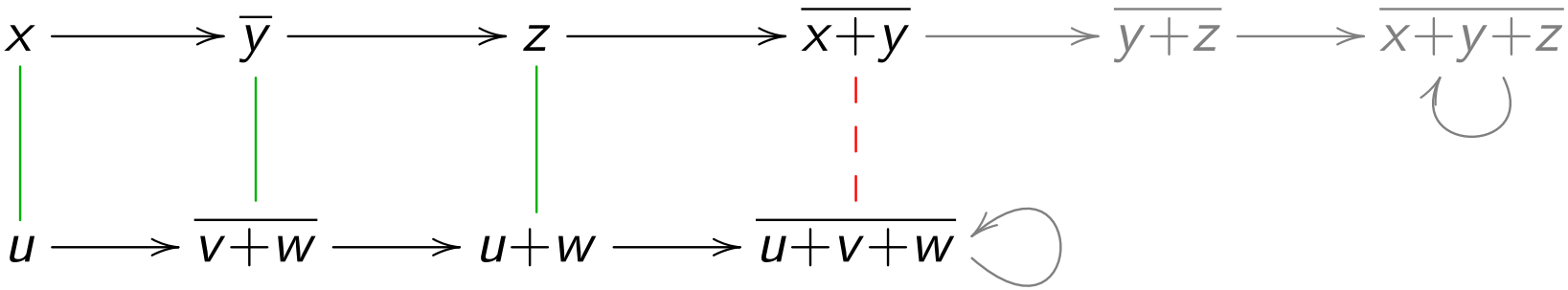


Checking language equivalence

One can do better:



$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v+w) \\ \hline = & (x+y,\ u+v+w) \end{array}$$



parts of the accessible subsets need not be explored

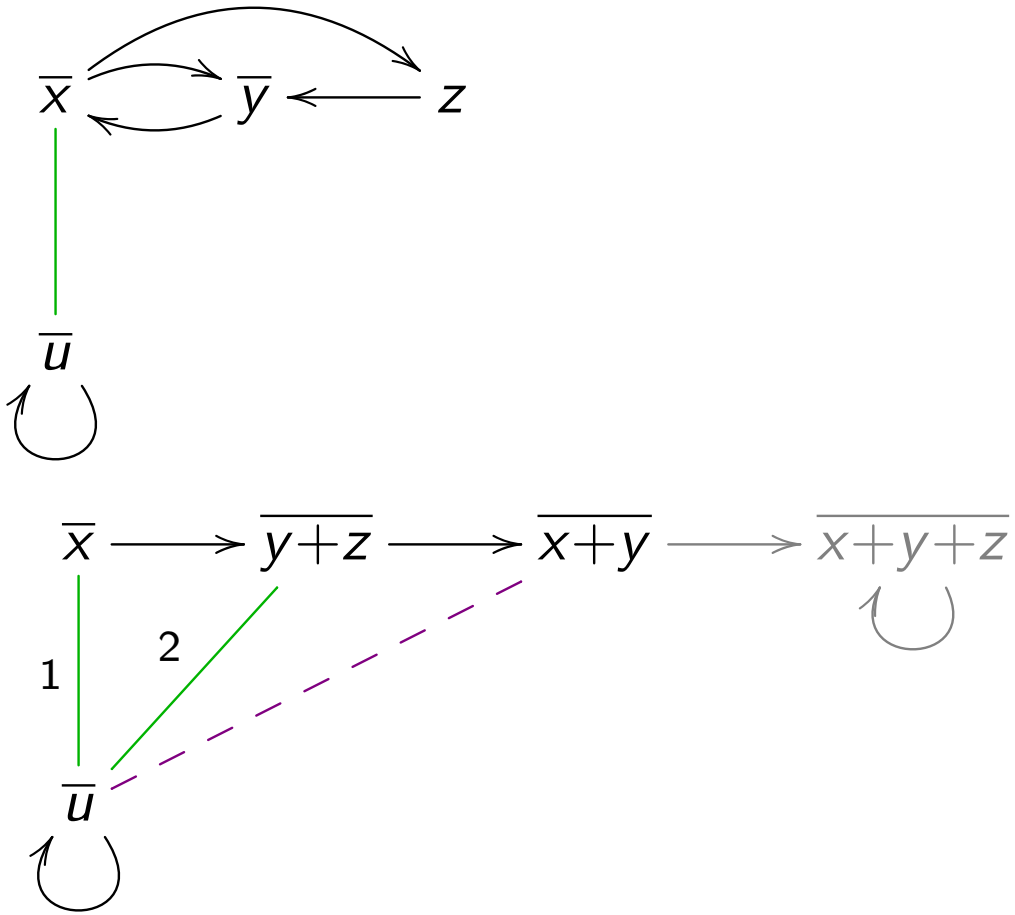
Correctness

bisimulation up to context

Lemma: if R is a bisimulation up to context,
then R^u is a bisimulation

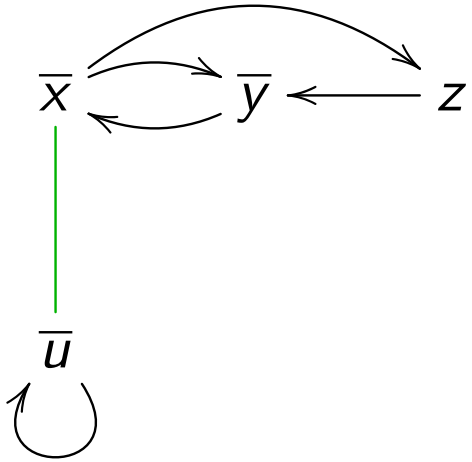
Checking language equivalence

One can do even better:

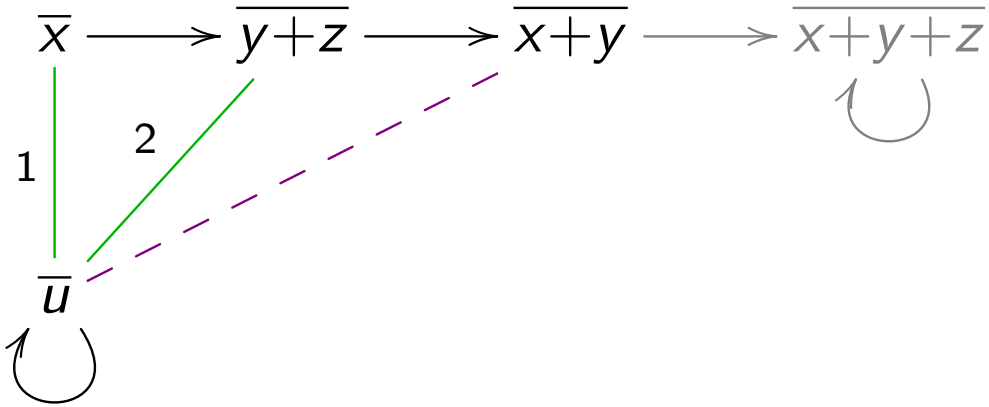


Checking language equivalence

One can do even better:



$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z\\
&= u && (2)
\end{aligned}
$$



Correctness

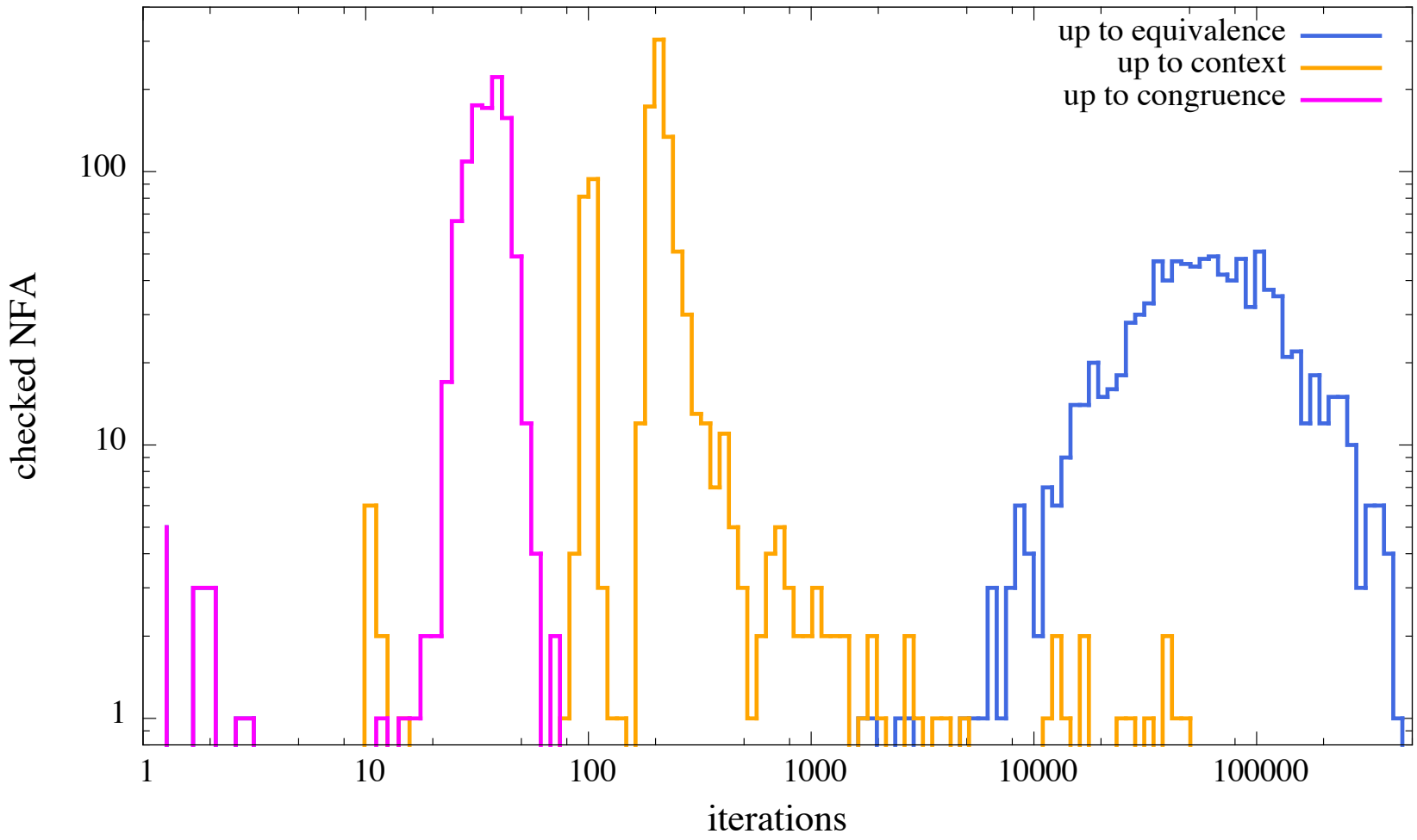
bisimulation up to congruence

Lemma: ...

In practice

http://perso.ens-lyon.fr/damien.pous/hkc/

In practice

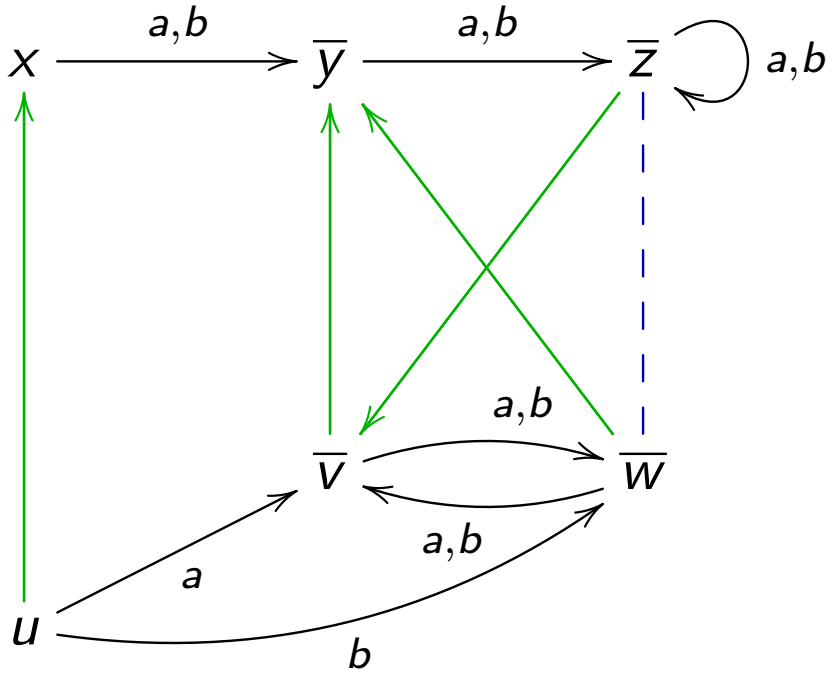

up to equivalence
up to context
up to congruence
checked NFA
100
10
1
1
10
100
1000
10000
100000
iterations

Outline

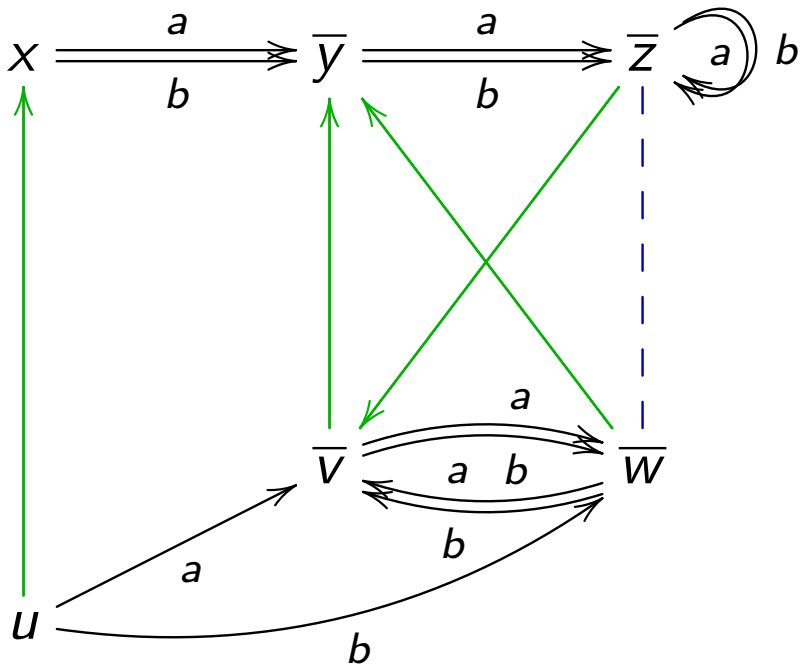
Many states

Many letters

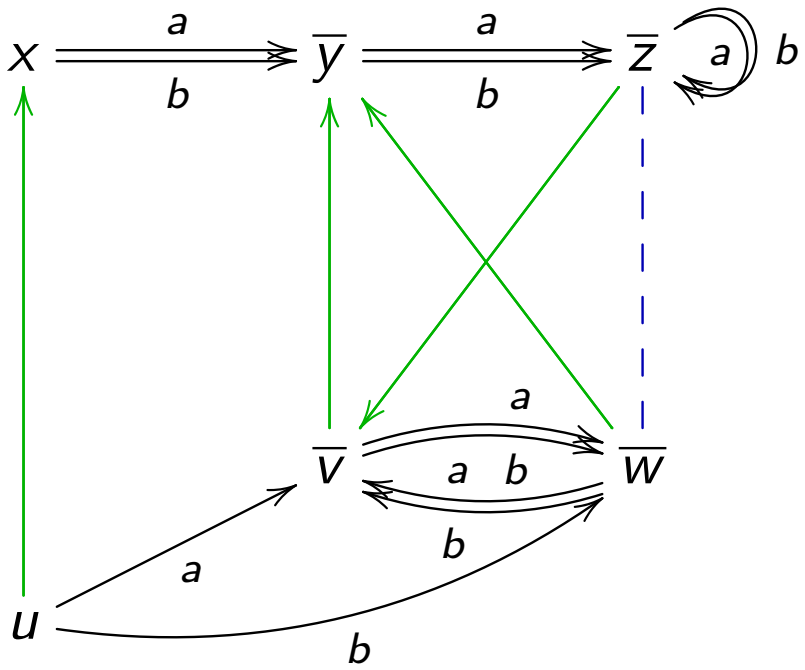
Few states, many letters



Few states, many letters

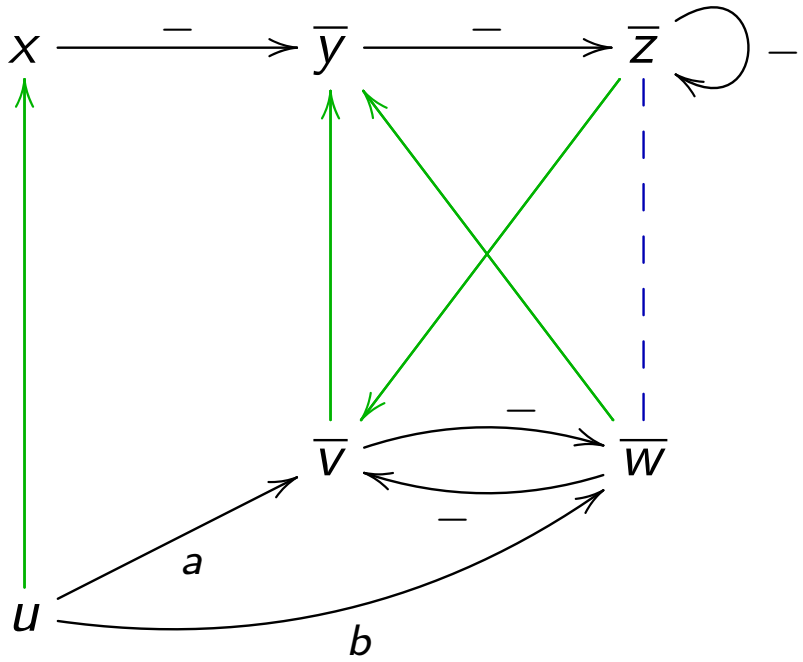


Few states, many letters



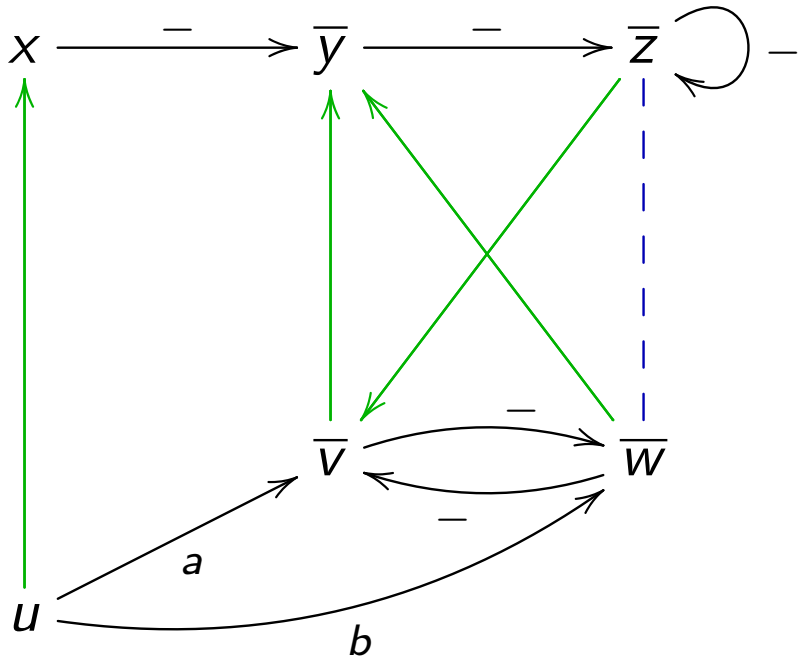
- standard practice: label transitions with formulas

Few states, many letters



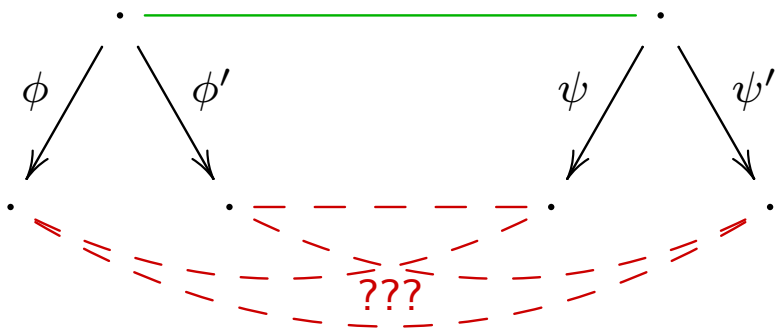
- standard practice: label transitions with formulas

Few states, many letters



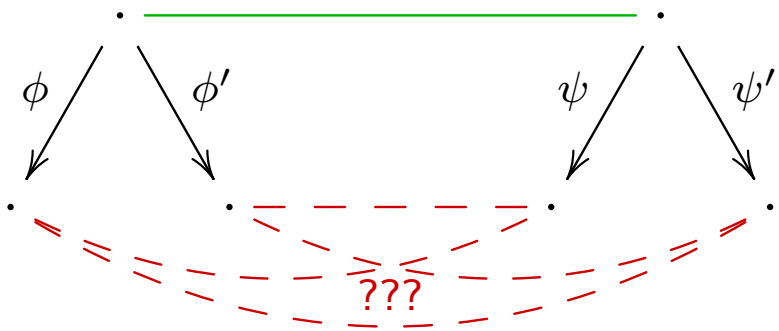
- standard practice: label transitions with formulas
  but then Hopcroft&Karp's algorithm can no longer be used

Few states, many letters



- standard practice: label transitions with formulas
  but then Hopcroft&Karp's algorithm can no longer be used

Few states, many letters

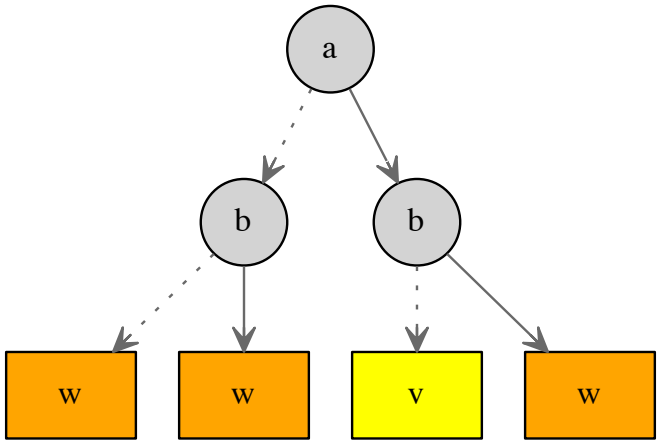


- standard practice: label transitions with formulas
  but then Hopcroft&Karp's algorithm can no longer be used
- more restrictive model here

Binary Decision Diagrams (BDDs) [Bryant'86]

Represent functions of type $2^A \to V$ with compressed decision trees

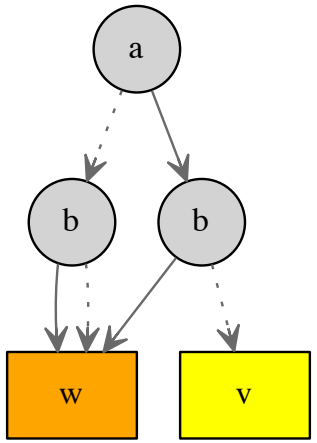
Example: A={a,b}, V={v,w}, $\alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$



Binary Decision Diagrams (BDDs) [Bryant'86]

Represent functions of type $2^A \to V$ with compressed decision trees

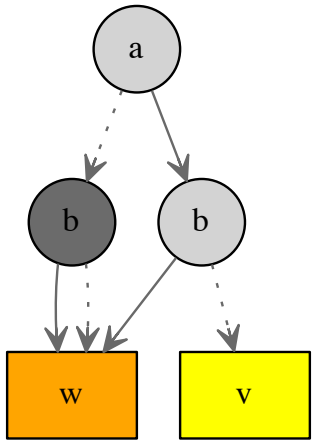
Example: A={a,b}, V={v,w}, $\alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$



Binary Decision Diagrams (BDDs) [Bryant'86]

Represent functions of type $2^A \to V$ with compressed decision trees

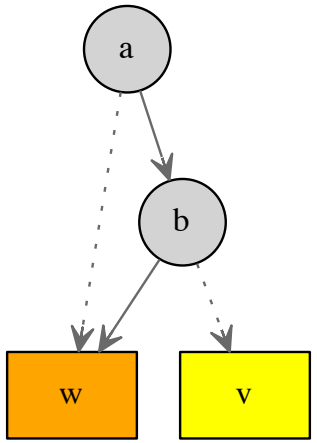
Example: A={a,b}, V={v,w}, $\alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$



Binary Decision Diagrams (BDDs) [Bryant'86]

Represent functions of type $2^A \to V$ with compressed decision trees

Example: A={a,b}, V={v,w}, $\alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$



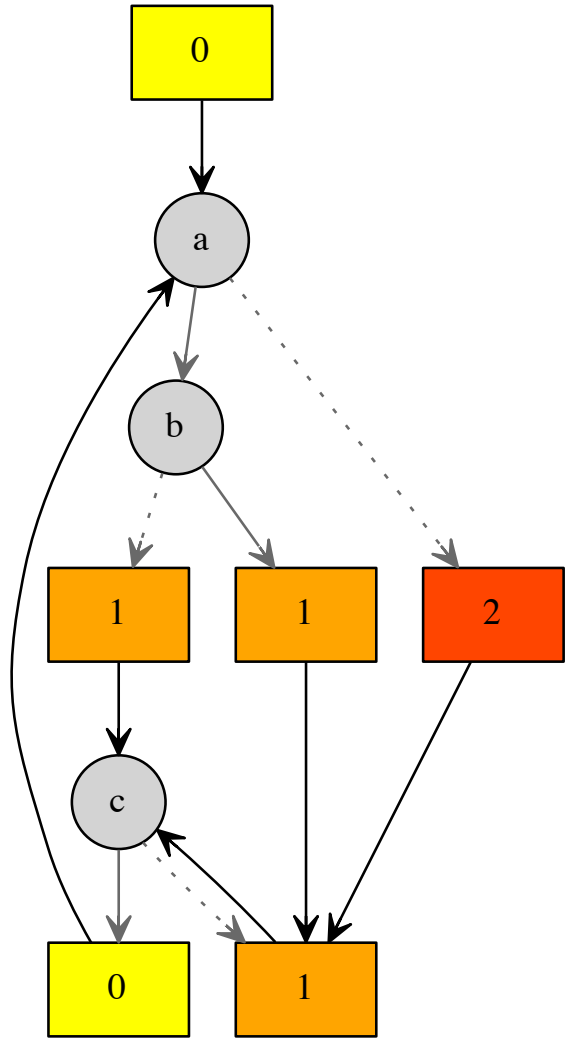
Symbolic Automata

Just represent the transition functions with BDDs

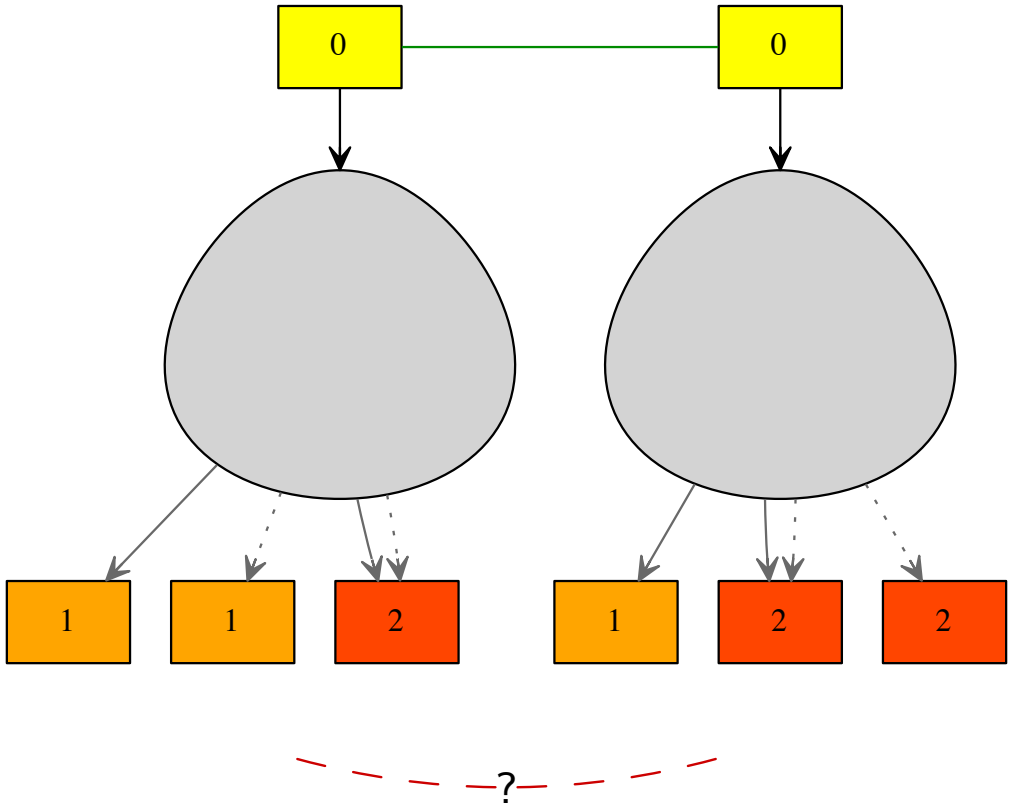
Example:

alphabet: 2^A with $A = \{a, b, c\}$

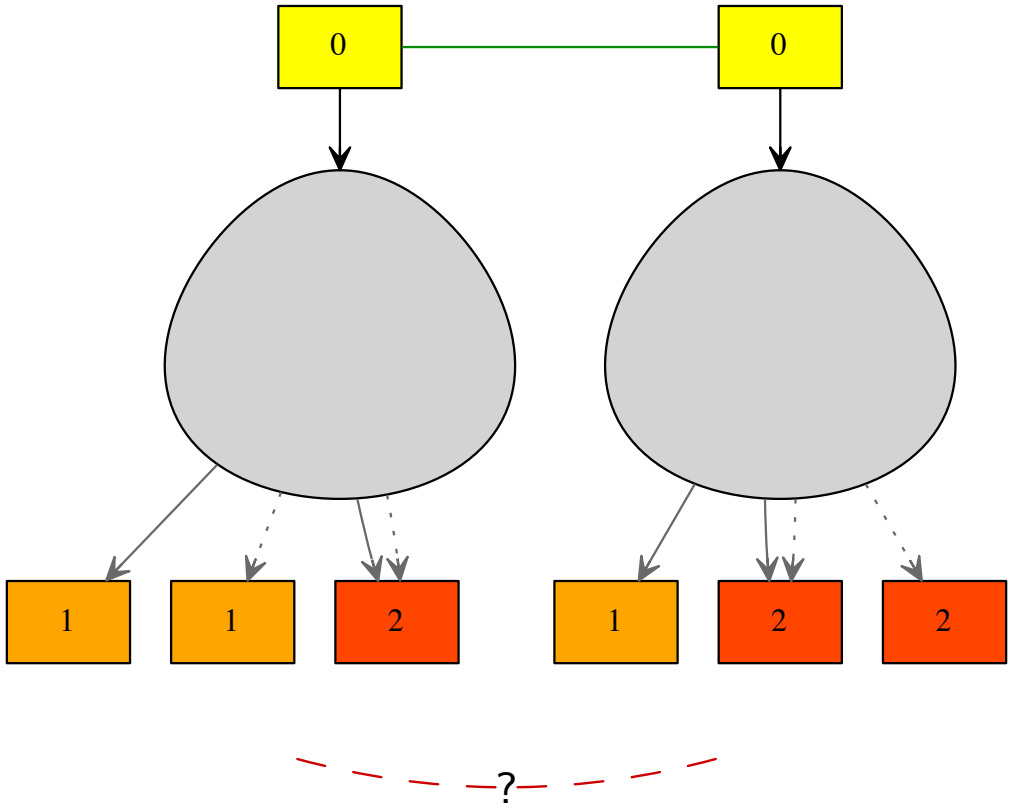
output set: $\mathbb{N}$



Iterate over the successors of two states

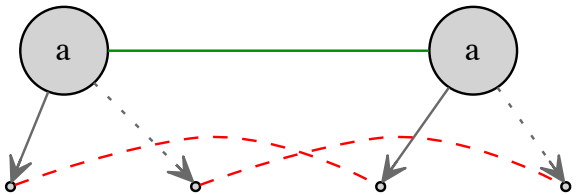


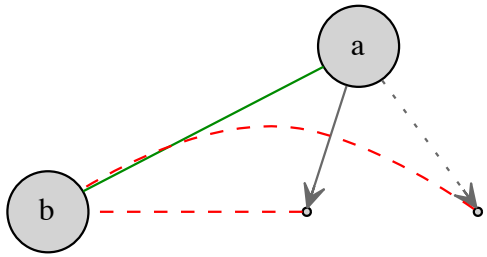
Iterate over the successors of two states

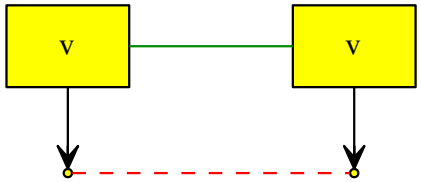


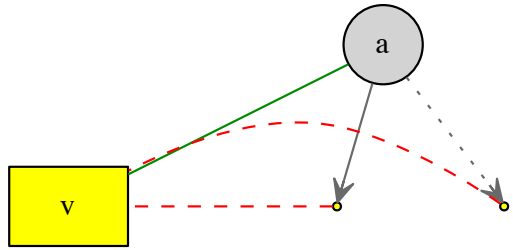
We just need an "`iter2`" function on BDDs

iter2 on BDDs

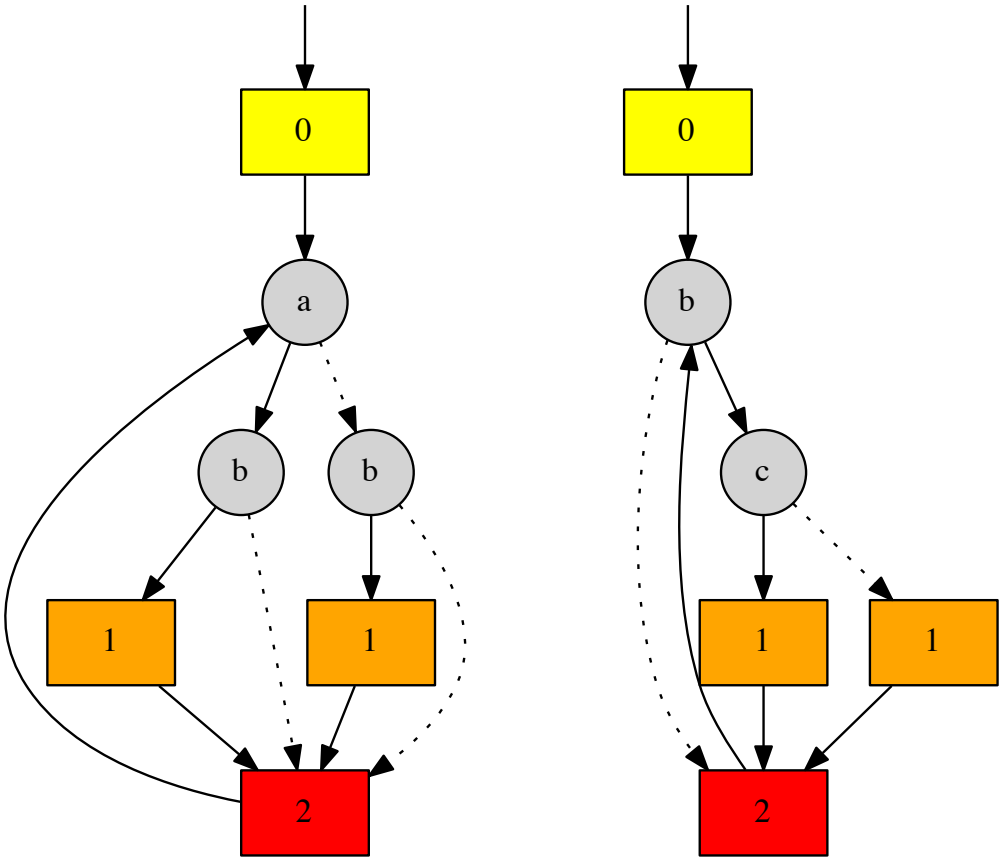




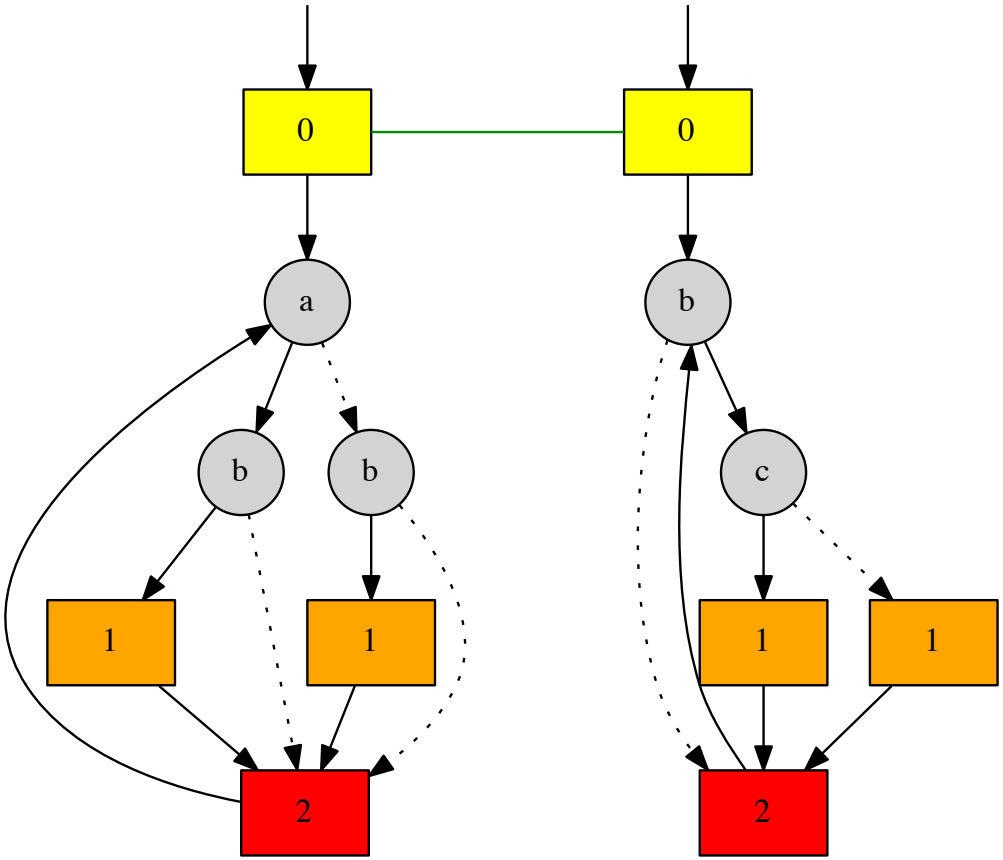




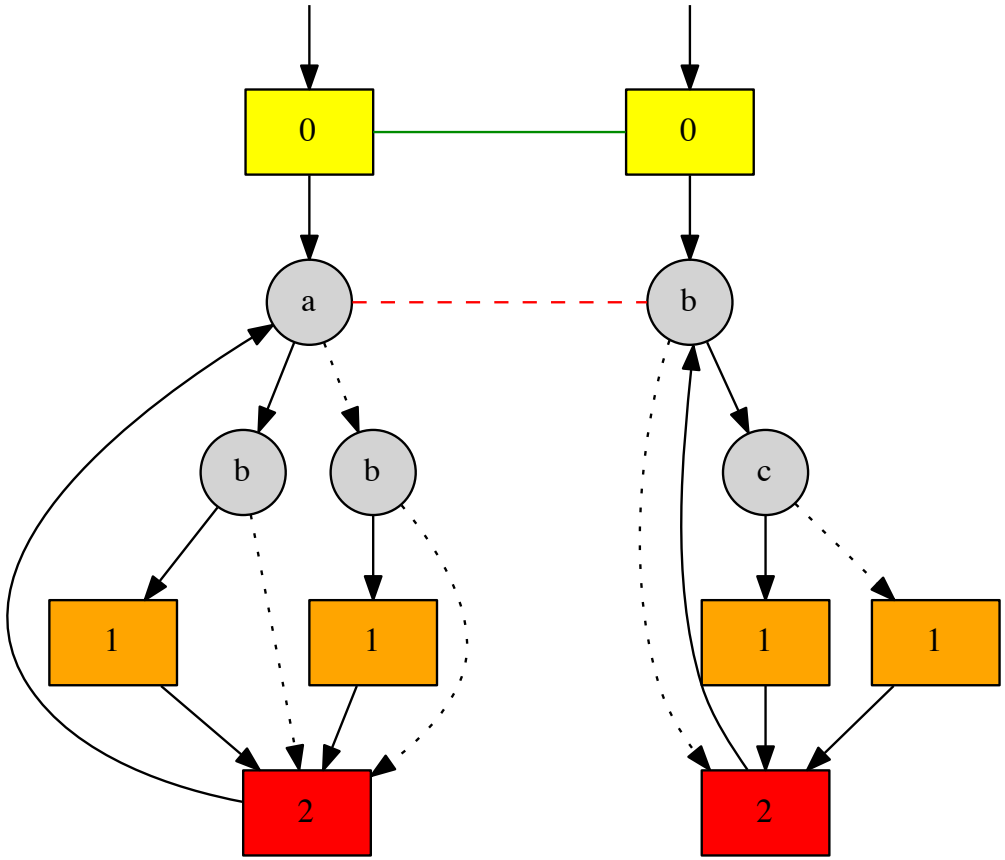
Symbolic algorithm



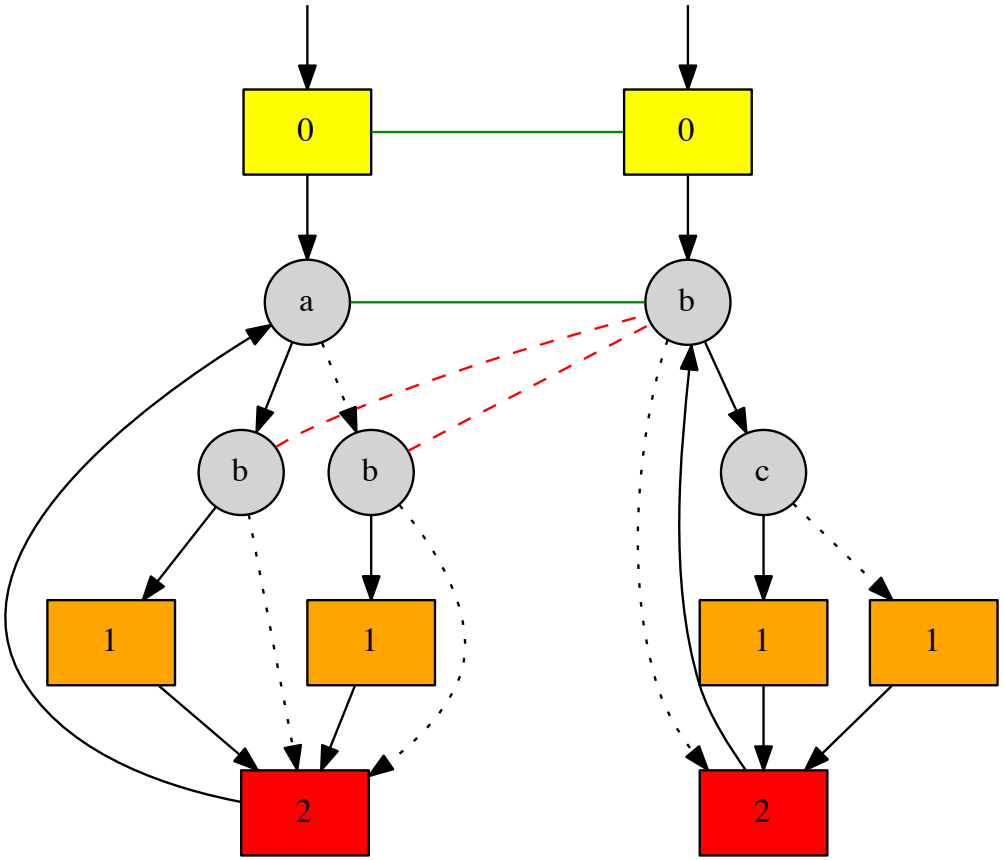
Symbolic algorithm



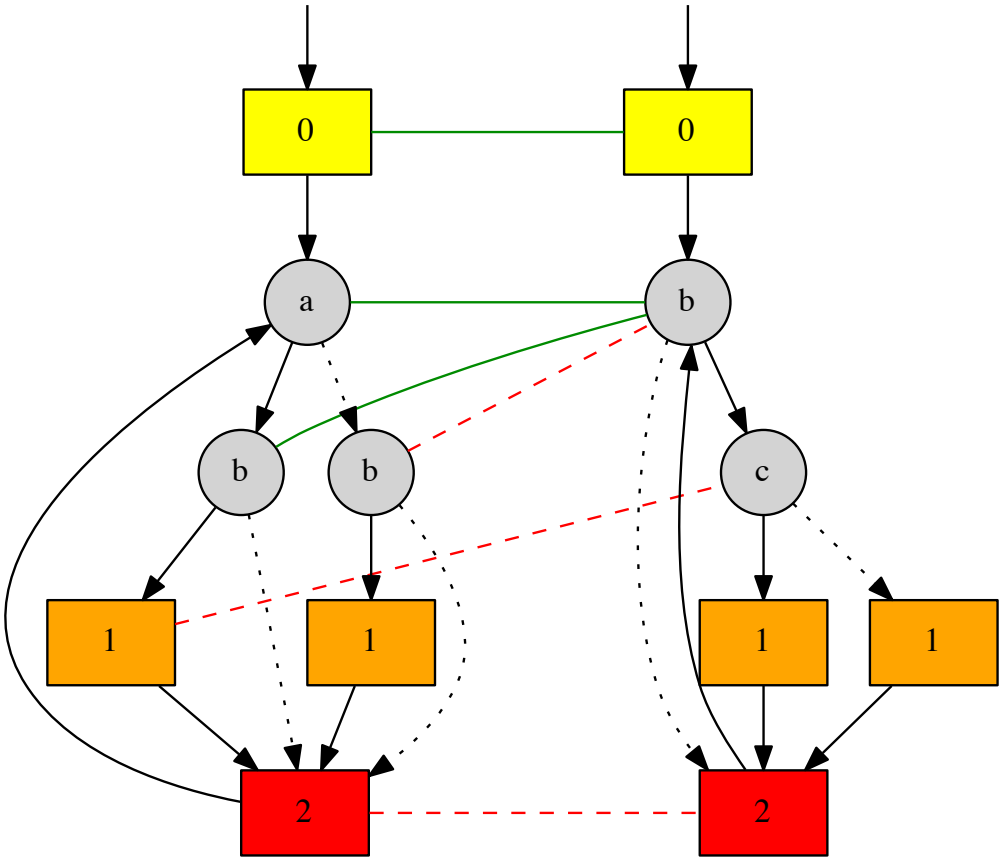
Symbolic algorithm



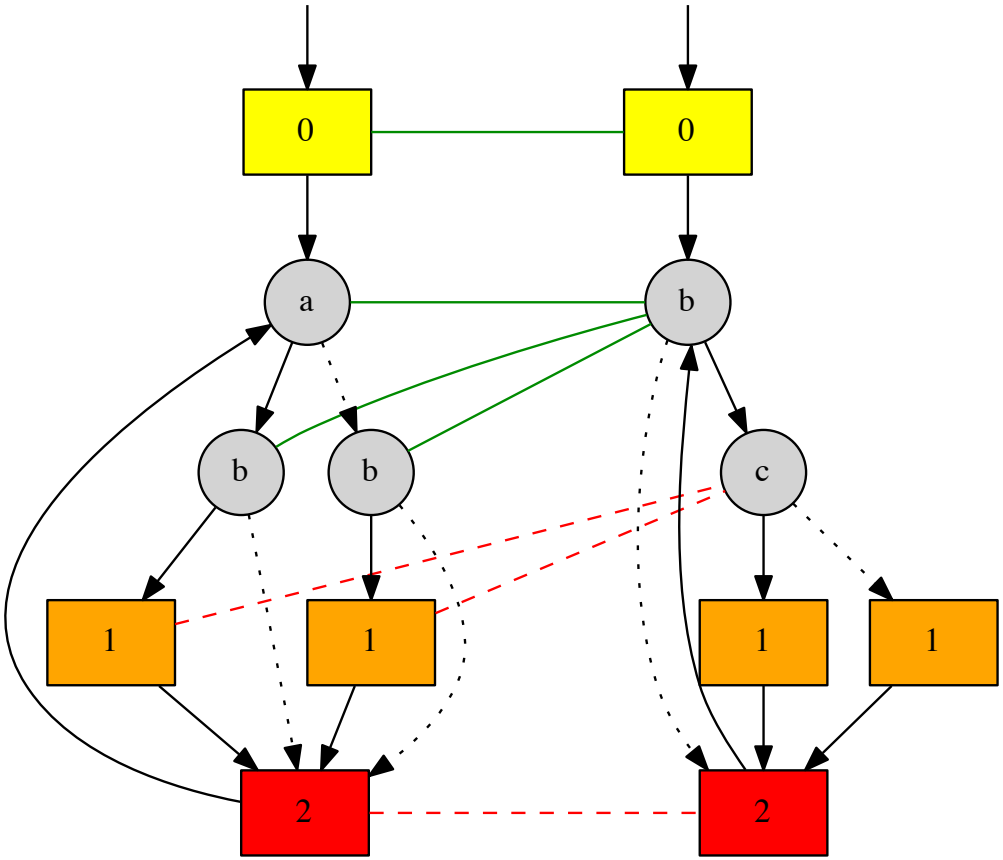
Symbolic algorithm



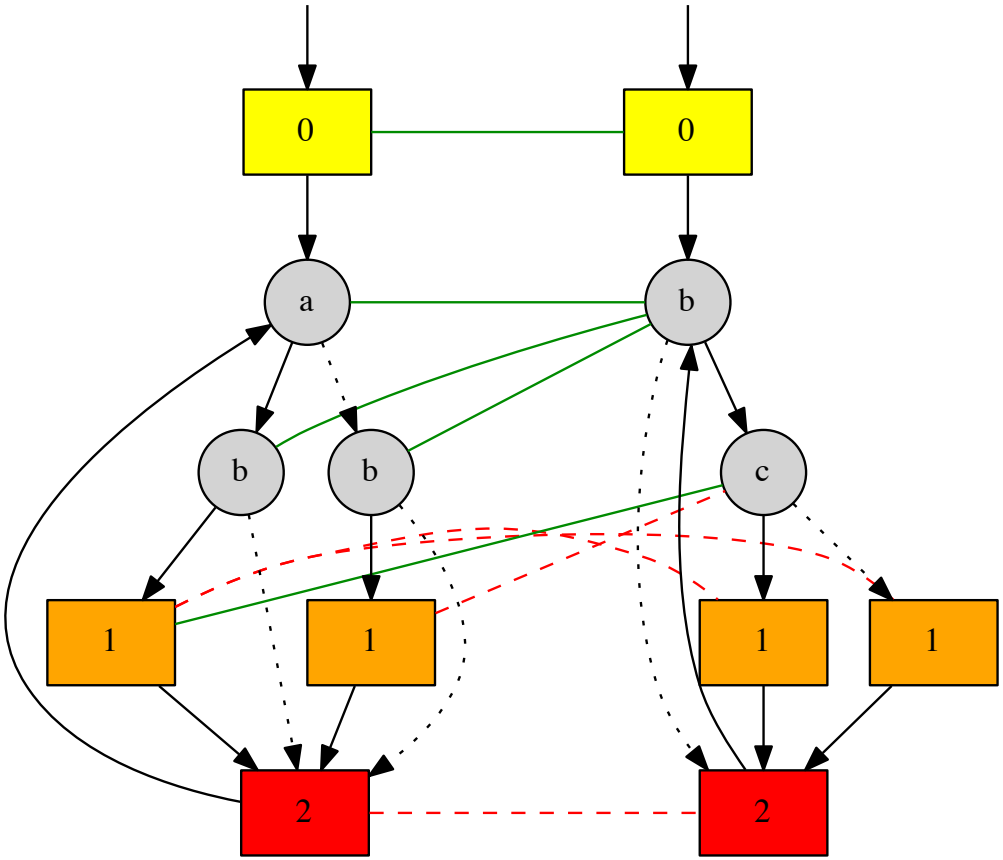
Symbolic algorithm



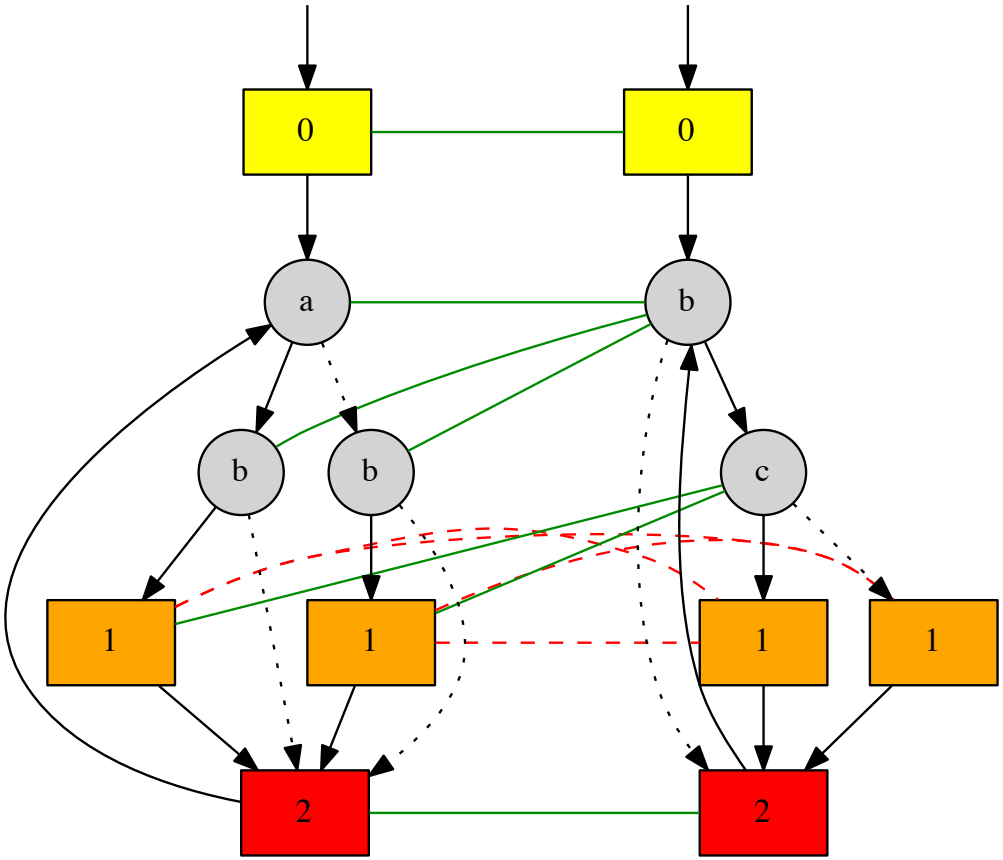
Symbolic algorithm



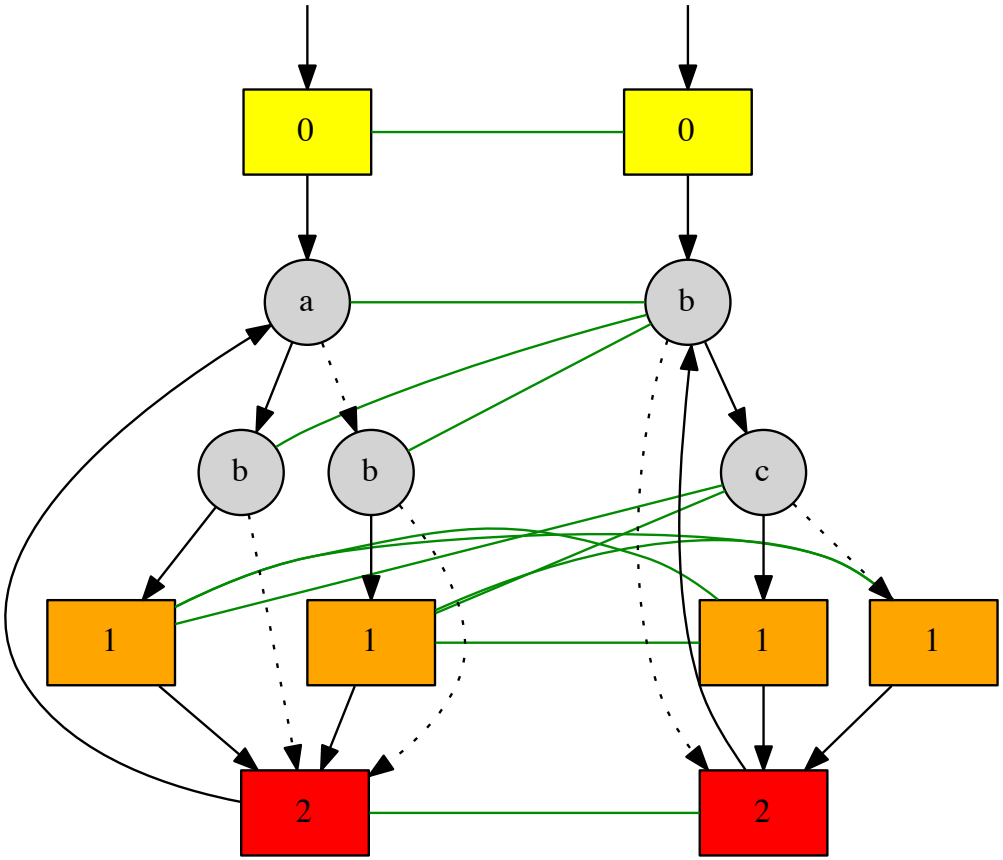
Symbolic algorithm



Symbolic algorithm



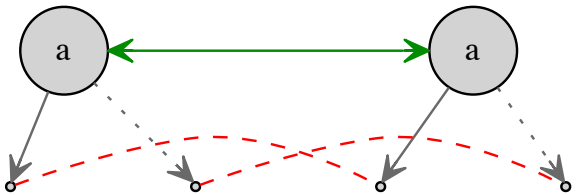
Symbolic algorithm

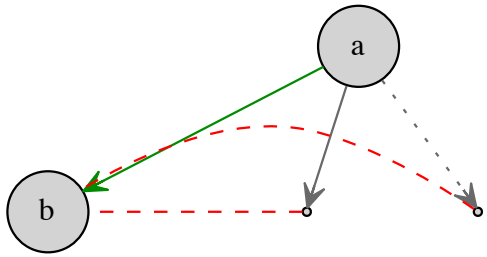


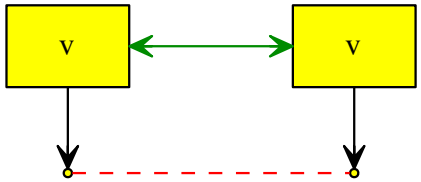
How to incorporate up to techniques?

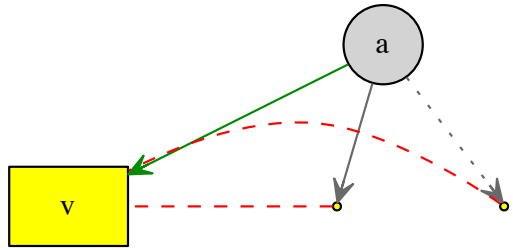
unifying BDDs

The union-find datastructure can be used between BDD nodes!

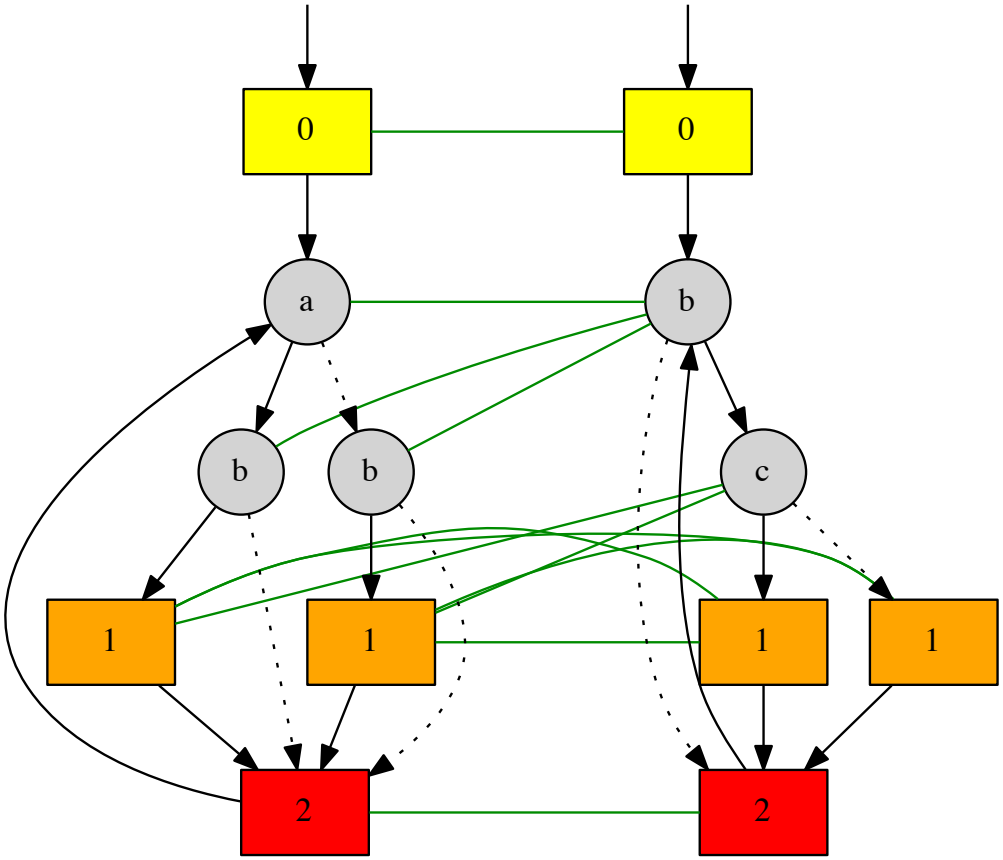




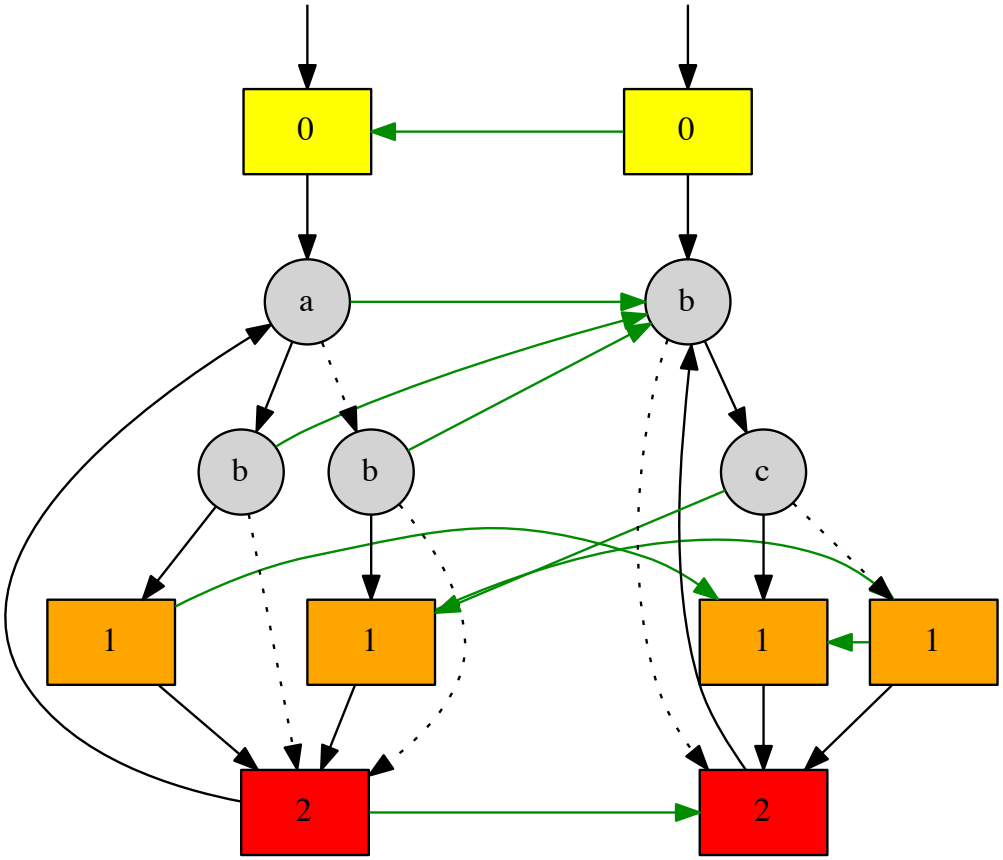




Symbolic Hopcroft & Karp's algorithm



Symbolic Hopcroft & Karp's algorithm



In practice

Applied to Kleene algebra with tests (KAT)

`http://perso.ens-lyon.fr/damien.pous/symkat/`

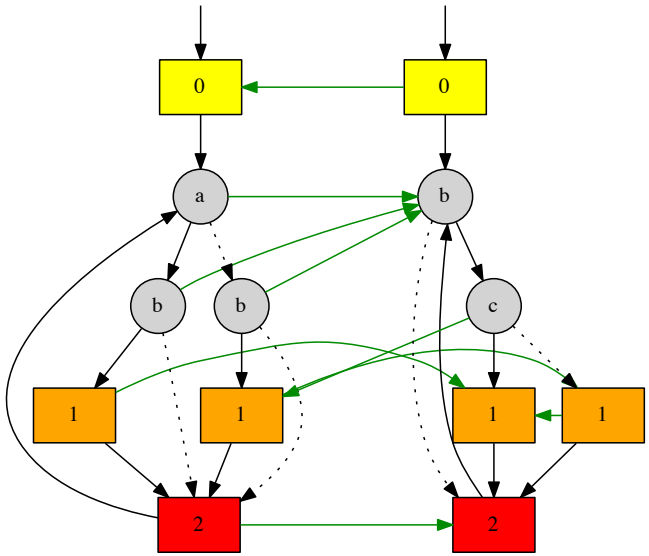
Complexity?

- Average complexity?
- Constrained union-find?

To remember

- Improvement of Hopcroft & Karp's algorithm [Bonchi, Pous '13]
  $\rightarrow$ using bisimulations up to congruence

- Symbolic version of Hopcroft & Karp's algorithm [Pous '15]
  $\rightarrow$ mixing BDDs and Union-Find



`http://perso.ens-lyon.fr/damien.pous/hkc/`
`http://perso.ens-lyon.fr/damien.pous/symkat/`