Kleene Algebras and Applications

Alexandra Silva

Course Content

Lecture 1

Kleene algebra
Brzozowski derivatives
Antimirov derivatives
Equivalence via automata

Lecture 2

Coinduction up-to

Lecture 3

Equivalence via axioms
Completeness

Lecture 4

Extensions with tests, observations, and concurrency

Lecture 1

Context

```
while a & b do
   p;
od;
while a do
   q ;
   while a & b do
      p;
   od
od
```

$\stackrel{?}{\equiv}$

```
while a do
   if b then
      p;
   else
   q;
od
```

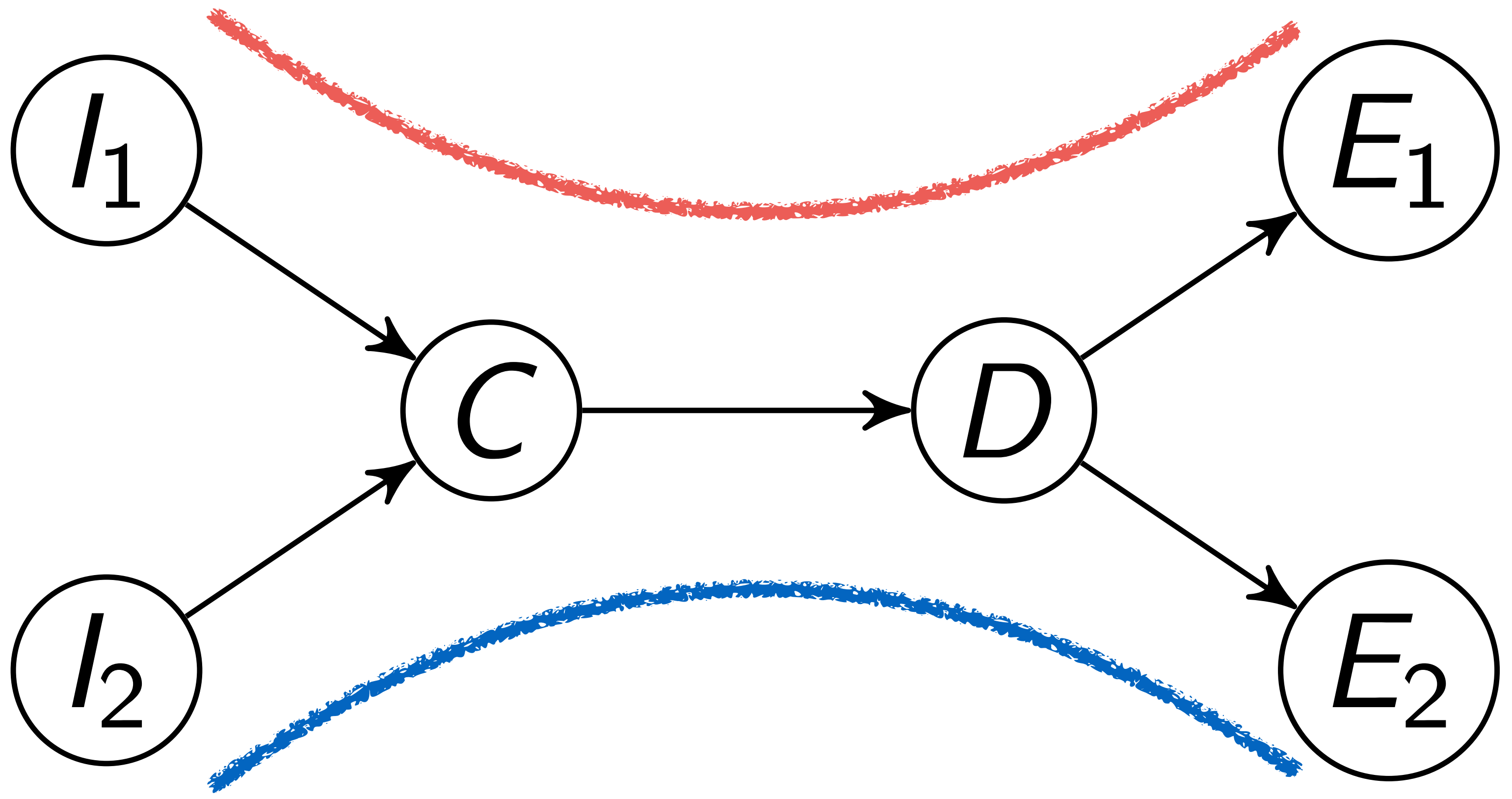
Context

```
while a & b do
   p;
od;
while a do
   q ;
   while a & b do
      p;
   od
od
```

$\stackrel{?}{\equiv}$

```
while a do
   if b then
      p;
   else
   q;
od
```

$$I_1 \; ; p \; ; E_1 \equiv 0$$



Context

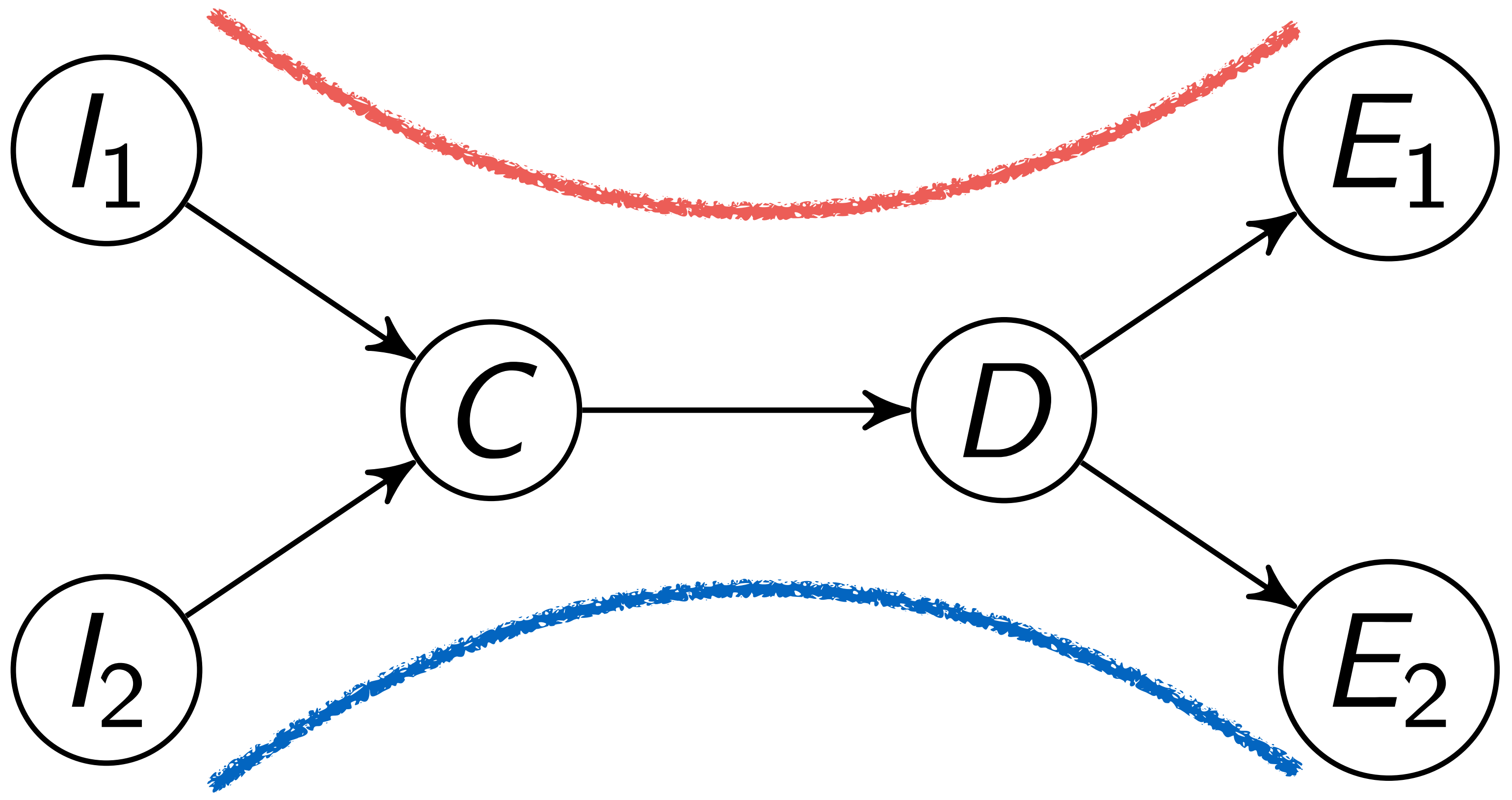
```
while a & b do
   p;
od;
while a do
   q ;
   while a & b do
      p;
   od
od
```

$\stackrel{?}{\equiv}$

```
while a do
   if b then
      p;
   else
   q;
od
```

Verification via Program Equivalence

$$I_1 \; ; p \; ; E_1 \equiv 0$$

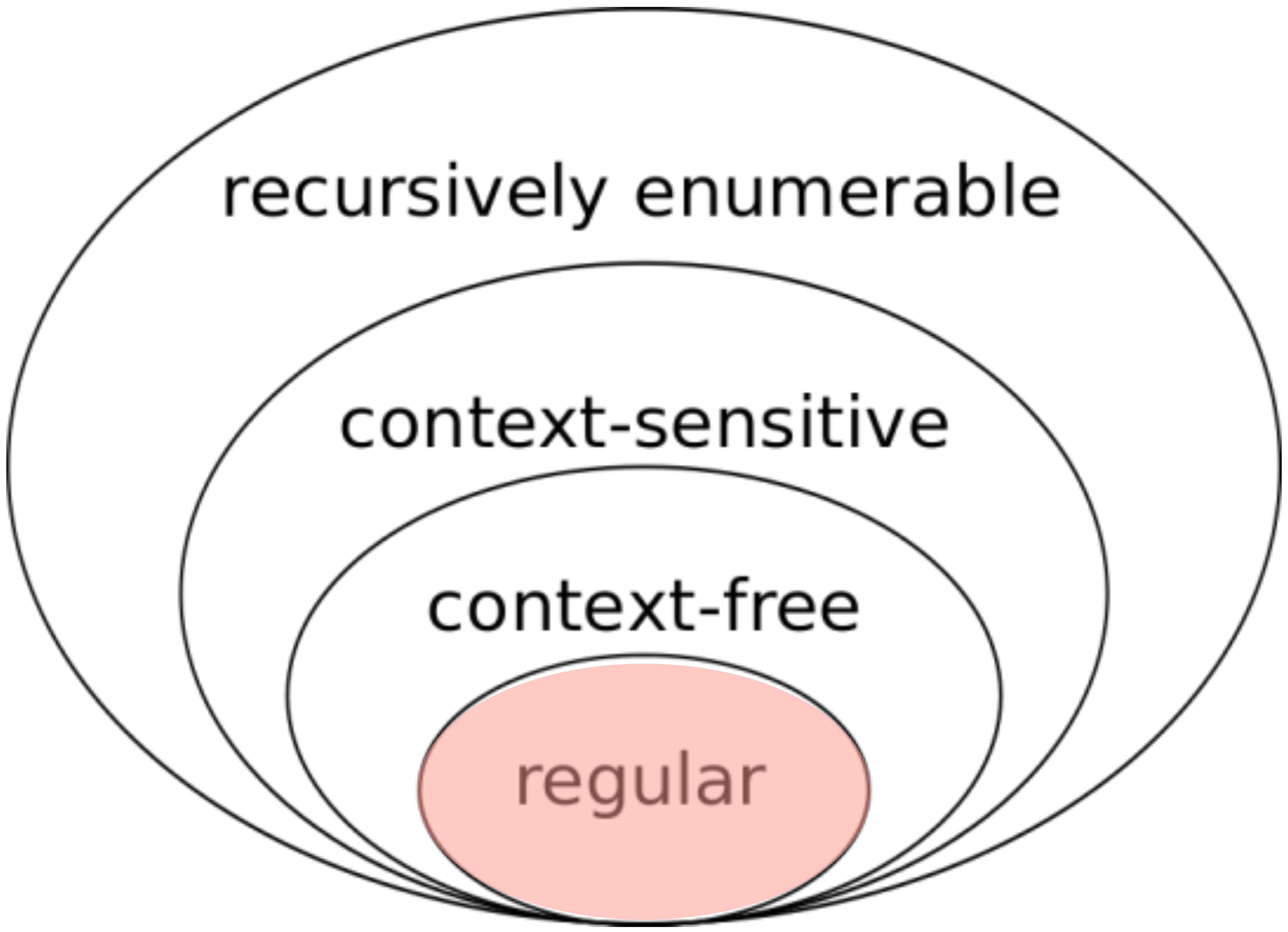


Languages of traces

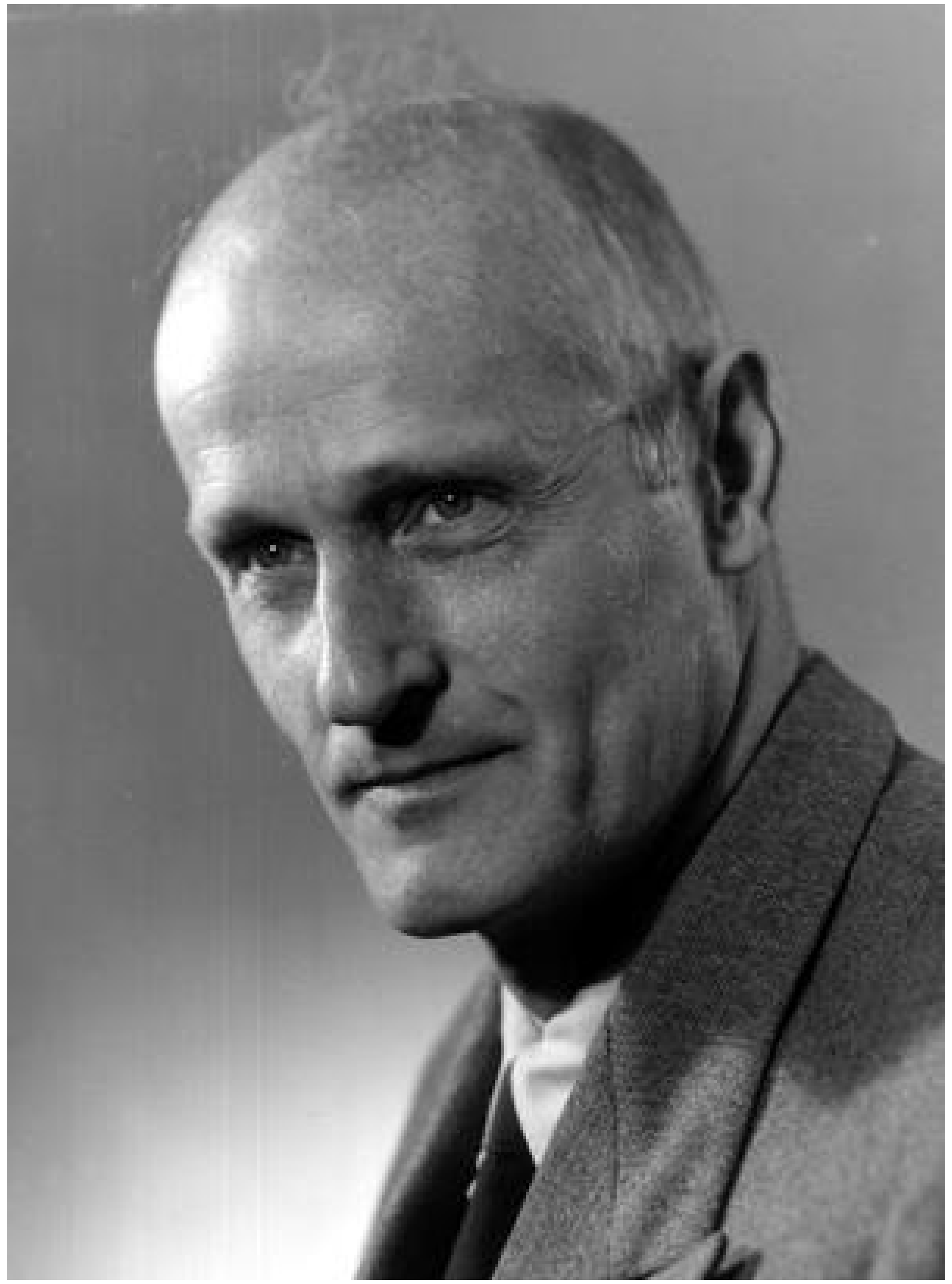
Some notation

A - finite alphabet
A* - finite words over A
Empty word - ε
Language - subset of words
$L \subseteq A^*$

Chomsky hierarchy



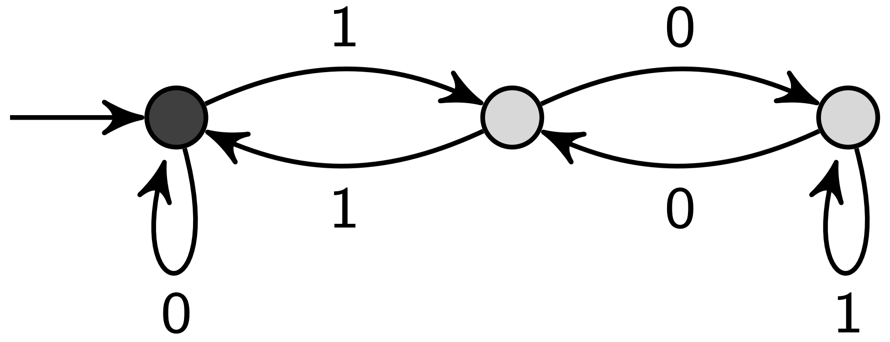
Kleene Algebra



Stephen Cole Kleene
(1909–1994)

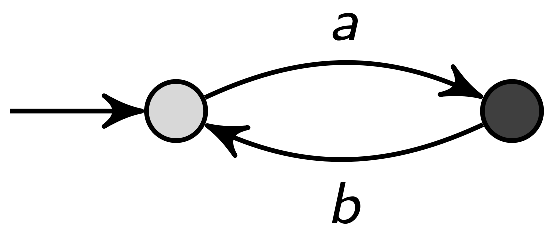
$(0 + 1(01^*0)^*1)^*$

{multiples of 3 in binary}



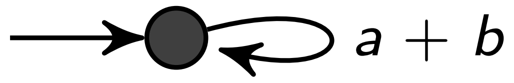
$(ab)^*a = a(ba)^*$

$\{a, aba, ababa, \ldots\}$



$(a + b)^* = a^*(ba^*)^*$

{all strings over $\{a, b\}$}



Regular expressions

$$r, r_1, r_2 ::= \underline{1} \mid \underline{0} \mid a \in A \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$$

Regular expressions

$$r, r_1, r_2 ::= \underline{1} \mid \underline{0} \mid a \in A \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$$

Language Semantics

$$L(\underline{1}) = \{\epsilon\} \qquad L(\underline{0}) = \emptyset \qquad L(a) = \{a\}$$

$$L(r_1 + r_2) = L(r_1) \cup L(r_2) \qquad L(r_1 r_2) = L(r_1) \cdot L(r_2) \qquad L(r^*) = L(r)^*$$

Exercise

What languages do these expressions describe?

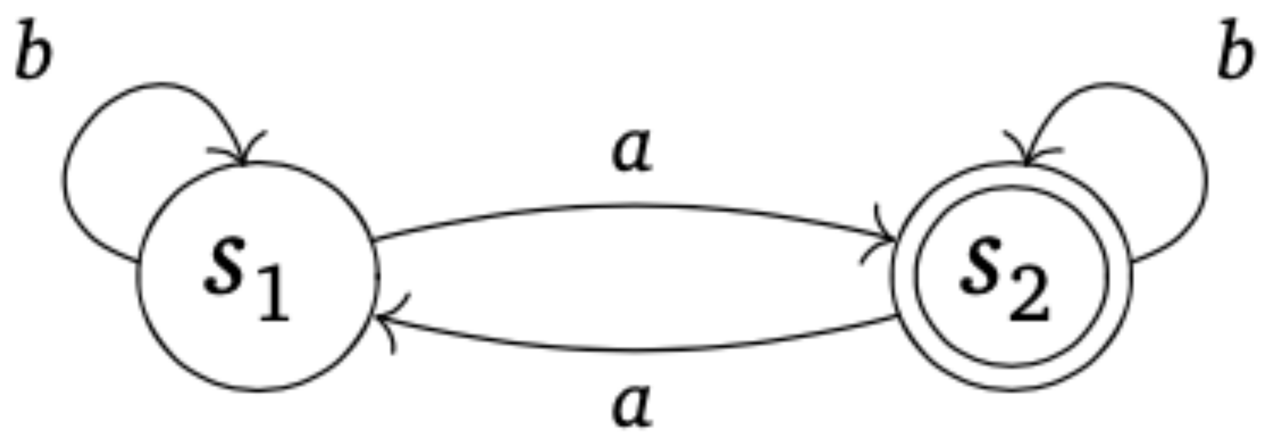
$$(a+b)(a+b)(a+b)^*$$

$$a^*ba + b^*ab$$

Give a regular expression describing this language?

$$L = \left\{ w \in A^* \,\middle|\, w \text{ contains } aba \text{ \underline{at least} once} \right\}$$

Deterministic Finite Automata



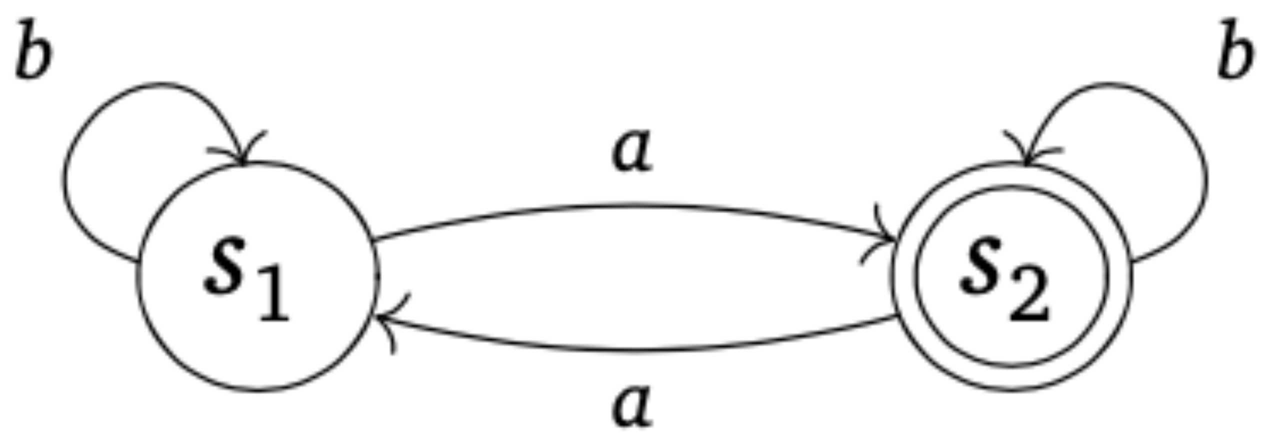
$$(S, \langle o, t \rangle)$$

States (set)

$$o \colon S \to 2$$ Final states

$$t \colon S \to S^A$$ Transition Function

Deterministic Finite Automata



$$(S, \langle o, t \rangle) \quad o \colon S \to 2 \quad t \colon S \to S^A$$

States (set)

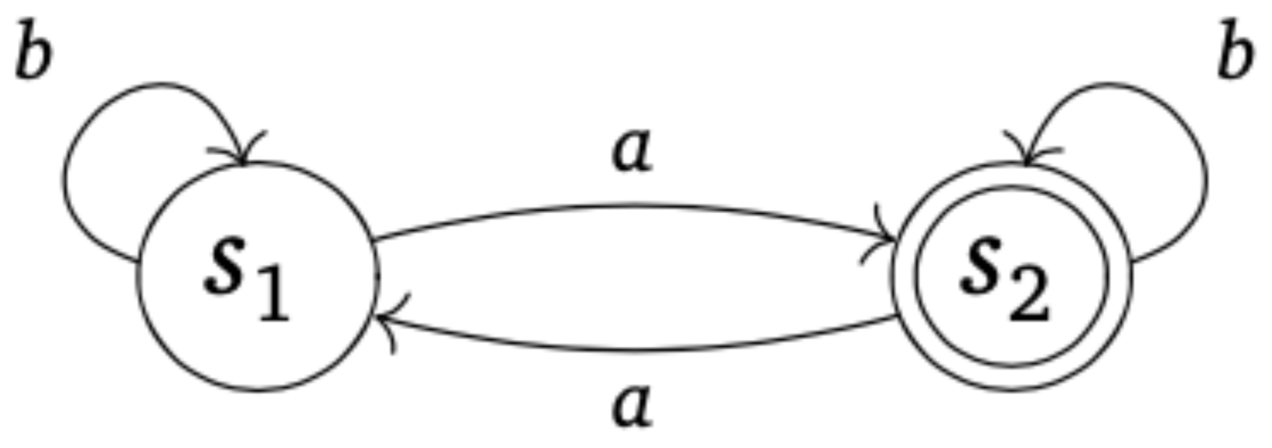
Final states

Transition Function

Inductive extension

$$t^*(s)(\varepsilon) = s$$
$$t^*(s)(aw) = t^*(t(s)(a))(w)$$

Deterministic Finite Automata



$$(S, \langle o, t \rangle)$$

States (set)

$$o \colon S \to 2$$ Final states

$$t \colon S \to S^A$$ Transition Function

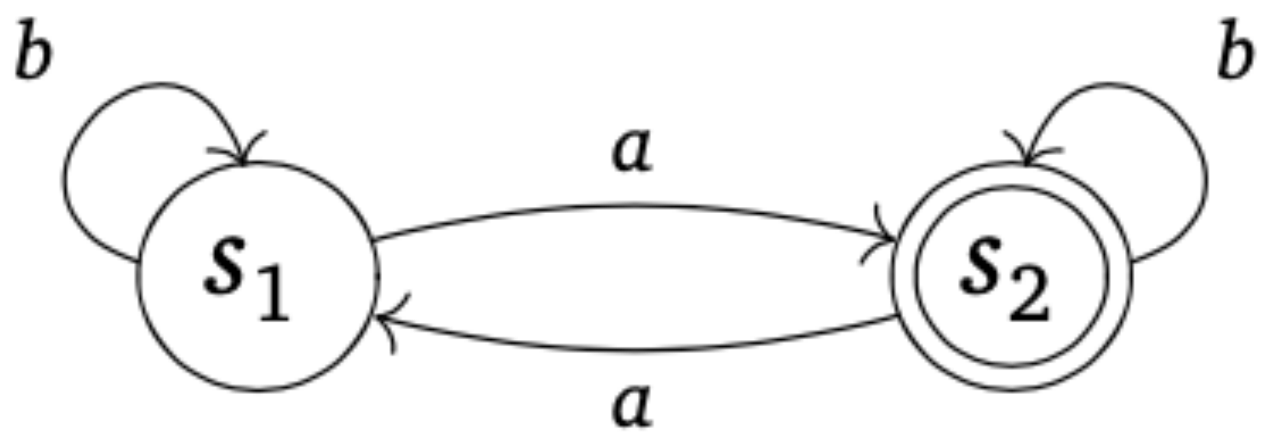
Inductive extension

$$t^*(s)(\varepsilon) = s$$
$$t^*(s)(aw) = t^*(t(s)(a))(w)$$

Notation

$$s_w = t^*(s)(w)$$

Deterministic Finite Automata



$$(S, \langle o, t \rangle)$$

States (set)

$$o \colon S \to 2$$ Final states

$$t \colon S \to S^A$$ Transition Function

Inductive extension

$$t^*(s)(\varepsilon) = s$$
$$t^*(s)(aw) = t^*(t(s)(a))(w)$$

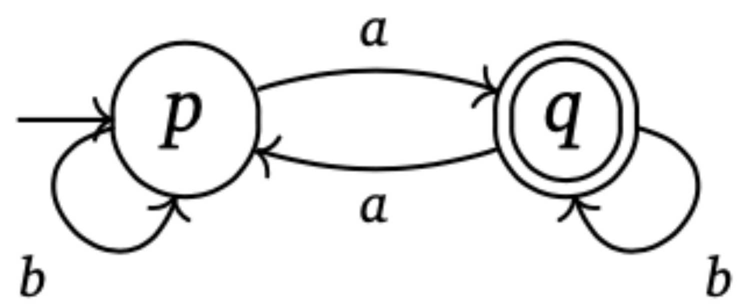
Notation

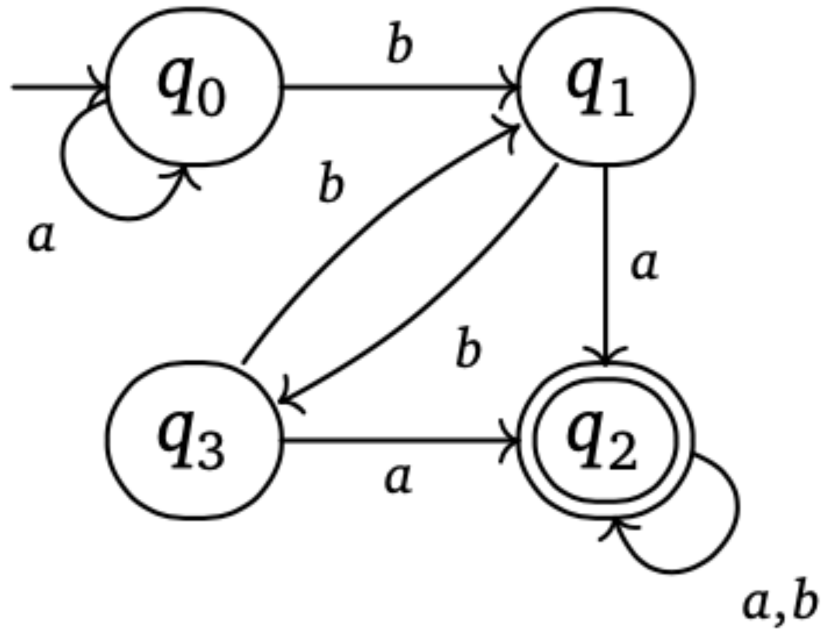
$$s_w = t^*(s)(w)$$

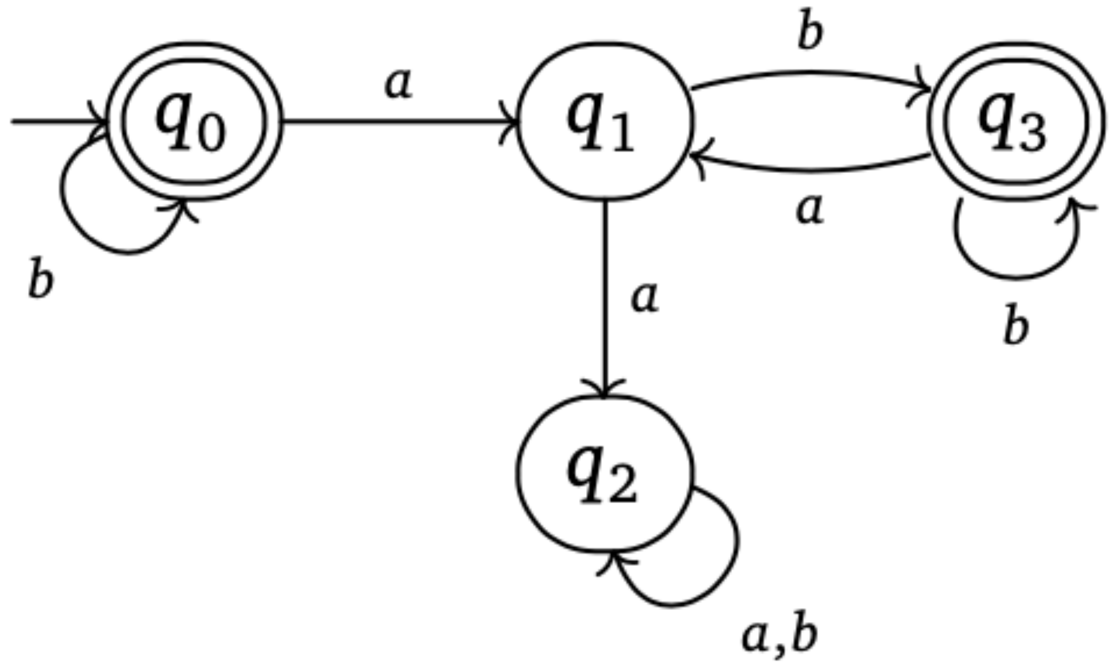
Language accepted by a state

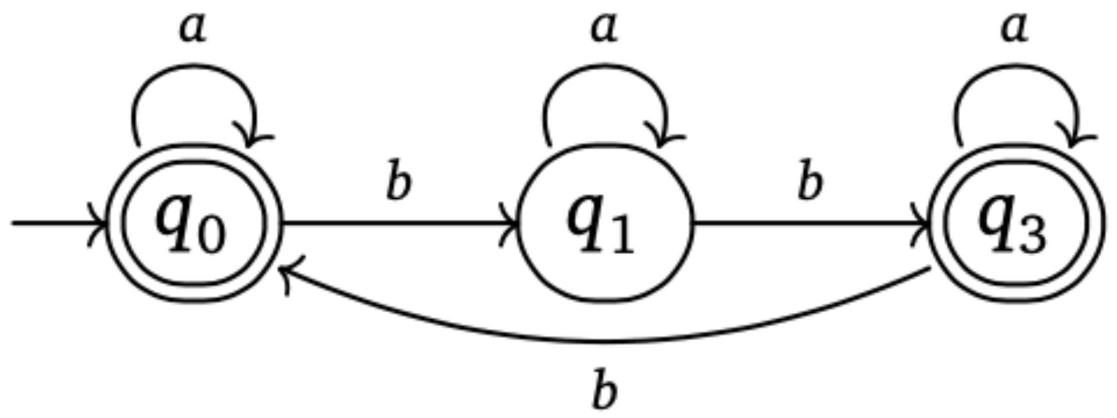
$$w \in L(s) \iff o(s_w) = 1$$

Exercise - DFA

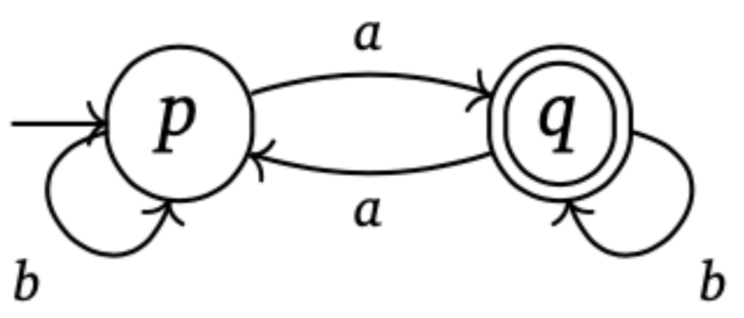




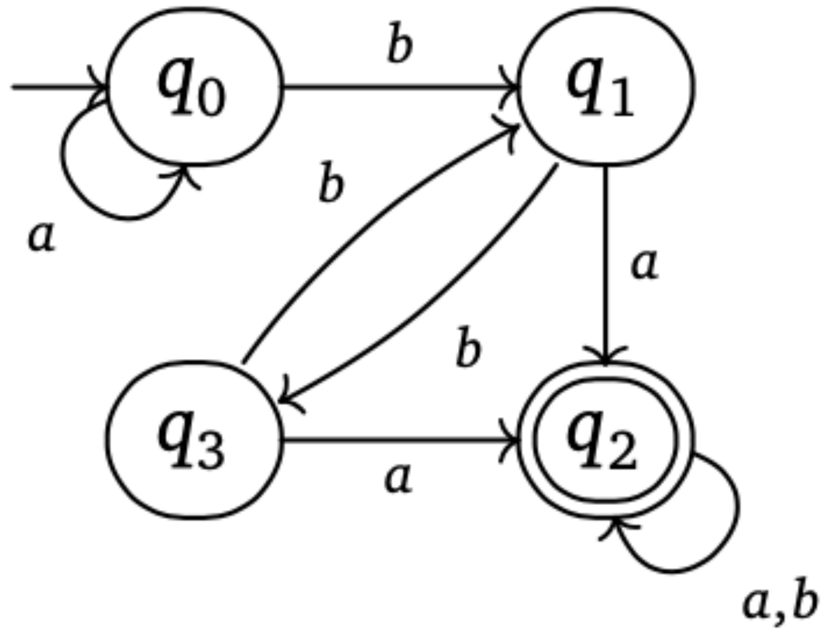




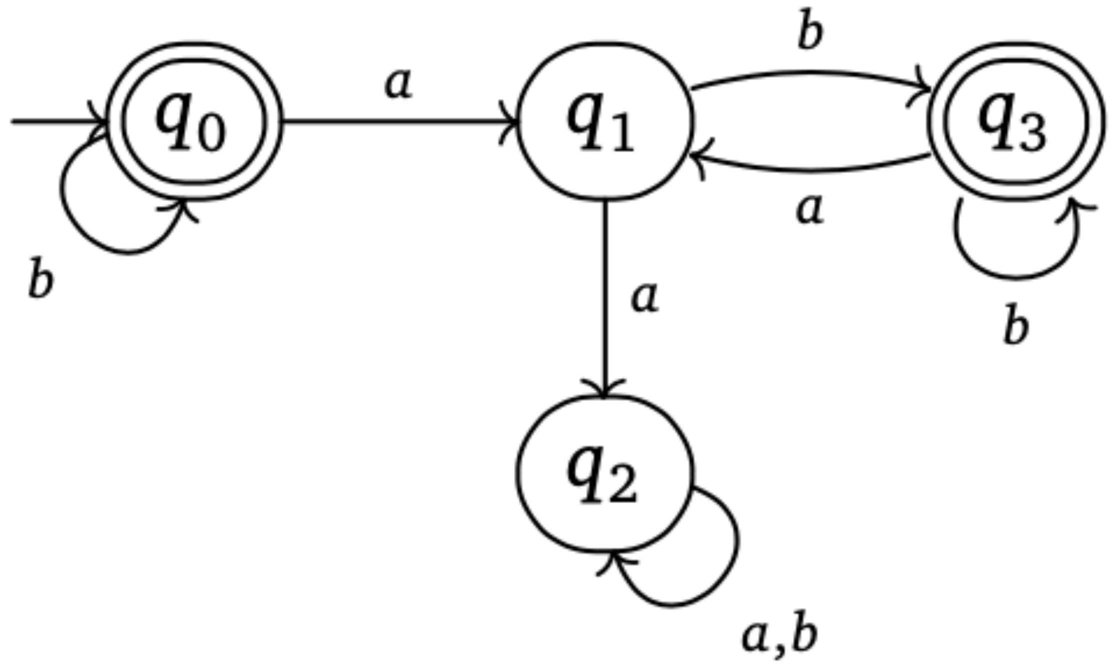
Exercise - DFA



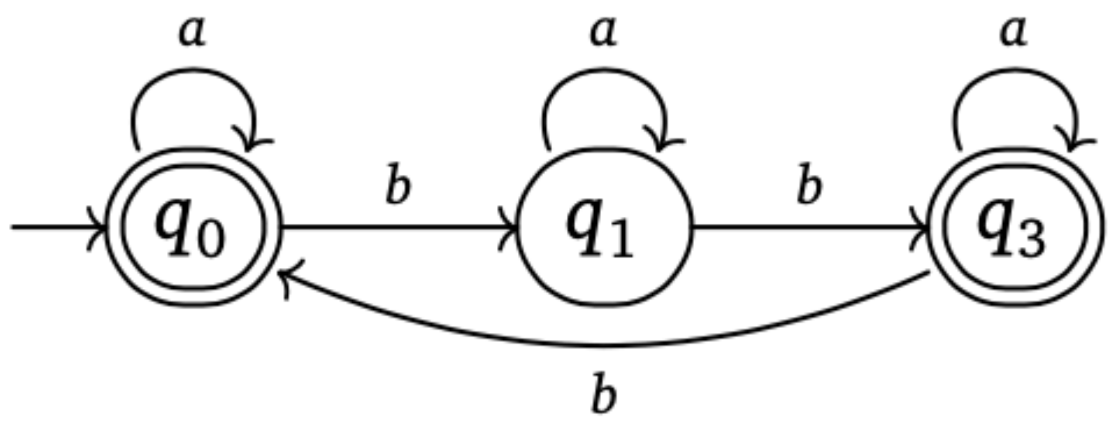
$\mathcal{L}(p) = \{w \in A^* \mid |w|_a \text{ is odd}\}.$



$\mathcal{L}(q_0) = \{w \in A^* \mid w \text{ has the subword } ba\}.$



$\mathcal{L}(q_0) = \{w \in A^* \mid \text{every } a \text{ in } w \text{ is followed by a } b\}.$



$\mathcal{L}(q_0) = \{w \in A^* \mid |w|_b = 3n + 2 \text{ or } |w|_b = 3n, n \in \mathbb{N}\}.$

Intermezzo: final coalgebra

$$
\begin{array}{ccc}
S & \overset{L}{\dashrightarrow} & 2^{A^*} \\
\Big\downarrow{\scriptstyle \langle o_S, t_S \rangle} & & \Big\downarrow{\scriptstyle \langle o_L, t_L \rangle} \\
2 \times S^A & \underset{id \times L^A}{\dashrightarrow} & 2 \times (2^{A^*})^A
\end{array}
$$

Kleene Algebra

$$
\begin{array}{llllll}
e_1 + (e_2 + e_3) & = & (e_1 + e_2) + e_3 & & & \text{(associativity of +)} \\
e_1 + e_2 & = & e_2 + e_1 & & & \text{(commutativity of +)} \\
e + e & = & e & & & \text{(idempotency of +)} \\
e + 0 & = & e & & & \text{(0 is an identity of +)} \\
\\
e_1(e_2e_3) & = & (e_1e_2)e_3 & & & \text{(associativity of }\cdot\text{)} \\
e1 & = & e & = & 1e & \text{(1 is an identity of }\cdot\text{)} \\
e0 & = & 0 & = & 0e & \text{(0 is an annihilator of }\cdot\text{)} \\
\\
(e_2 + e_3)e_1 & = & e_2e_1 + e_3e_1 & & & \text{(right distributivity)} \\
e_1(e_2 + e_3) & = & e_1e_2 + e_1e_3 & & & \text{(left distributivity)} \\
e^*e + \lambda & = & e^* & & & \\
ee^* + \lambda & = & e^* & & &
\end{array}
$$

Exercise

1. $x^*x^* = x^*$
2. $x^*=x^{**}$
3. $(x+y)^* = (x^*y)^*x^*$ denesting
4. $x(yx)^* = (xy)^*x$ sliding
5. $xy=yz \Rightarrow x^*y = yz^*$

Kleene Algebra, examples

- Regular languages
  - **+ is union**
  - **; is pointwise concatenation**
  - *** is iteration**

Kleene Algebra, examples

- Regular languages
  - **+ is union**
  - **; is pointwise concatenation**
  - *** is iteration**

- Binary Relations
  - **+ is union**
  - **; is relational composition**
  - *** is reflexive transitive closure**

Kleene Algebra, examples

- Regular languages
  - **+ is union**
  - **; is pointwise concatenation**
  - *** is iteration**

- Binary Relations
  - **+ is union**
  - **; is relational composition**
  - *** is reflexive transitive closure**

- Square Matrices over a KA **K**
  - **+ and ; lifted to usual matrices ops**
  - *** iteratively from 2 x 2**

Kleene Algebra, examples

- Regular languages
  - **+ is union**
  - **; is pointwise concatenation**
  - *** is iteration**

- Binary Relations
  - **+ is union**
  - **; is relational composition**
  - *** is reflexive transitive closure**

- Square Matrices over a KA **K**
  - **+ and ; lifted to usual matrices ops**
  - *** iteratively from 2 x 2**

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^* = \begin{bmatrix} (a + bd^*c)^* & (a + bd^*c)^* bd^* \\ (d + ca^*b)^* ca^* & (d + ca^*b)^* \end{bmatrix}$$

Kleene Algebra, examples

- Regular languages
  - **+ is union**
  - **; is pointwise concatenation**
  - *** is iteration**

Important for relational verification

- Binary Relations
  - **+ is union**
  - **; is relational composition**
  - *** is reflexive transitive closure**

- Square Matrices over a KA **K**
  - **+ and ; lifted to usual matrices ops**
  - *** iteratively from 2 x 2**

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^* = \begin{bmatrix} (a+bd^*c)^* & (a+bd^*c)^*bd^* \\ (d+ca^*b)^*ca^* & (d+ca^*b)^* \end{bmatrix}$$

Kleene Algebra, examples

- Regular languages
  - **+ is union**
  - **; is pointwise concatenation**
  - *** is iteration**

Important for relational verification

- Binary Relations
  - **+ is union**
  - **; is relational composition**
  - *** is reflexive transitive closure**

Important for automata representations

- Square Matrices over a KA **K**
  - **+ and ; lifted to usual matrices ops**
  - *** iteratively from 2 x 2**

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^* = \begin{bmatrix} (a+bd^*c)^* & (a+bd^*c)^*bd^* \\ (d+ca^*b)^*ca^* & (d+ca^*b)^* \end{bmatrix}$$

Kleene's Theorem

Theorem (Kleene'52): Let L be a subset of A^*. TFAE:

1. L is regular
2. L is accepted by a deterministic finite automaton

Proof

1 => 2 : Syntactic Brzozowski derivatives

2 => 1 : State elimination

Brzozowski Derivatives

$$r, r_1, r_2 ::= \underline{1} \mid \underline{0} \mid a \in A \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$$

Brzozowski Derivatives

$$r, r_1, r_2 ::= \underline{1} \mid \underline{0} \mid a \in A \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$$

$$\begin{aligned} o_{\mathcal{R}}(\underline{0}) &= 0 \\ o_{\mathcal{R}}(\underline{1}) &= 1 \\ o_{\mathcal{R}}(a) &= 0 \end{aligned}$$

$$\begin{aligned} o_{\mathcal{R}}(r_1 + r_2) &= o_{\mathcal{R}}(r_1) \vee o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(r_1 r_2) &= o_{\mathcal{R}}(r_1) \wedge o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(r^*) &= 1 \end{aligned}$$

Brzozowski Derivatives

$$r, r_1, r_2 ::= \underline{1} \mid \underline{0} \mid a \in A \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$$

$$\begin{aligned} o_{\mathcal{R}}(\underline{0}) &= 0 \\ o_{\mathcal{R}}(\underline{1}) &= 1 \\ o_{\mathcal{R}}(a) &= 0 \end{aligned}$$

$$\begin{aligned} o_{\mathcal{R}}(r_1 + r_2) &= o_{\mathcal{R}}(r_1) \vee o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(r_1 r_2) &= o_{\mathcal{R}}(r_1) \wedge o_{\mathcal{R}}(r_2) \\ o_{\mathcal{R}}(r^*) &= 1 \end{aligned}$$

$$\begin{aligned} (\underline{0})_a &= \underline{0} \\ (\underline{1})_a &= \underline{0} \\ (a)_{a'} &= \begin{cases} \underline{1} & \text{if } a = a' \\ \underline{0} & \text{if } a \neq a' \end{cases} \end{aligned}$$

$$\begin{aligned} (r_1 + r_2)_a &= (r_1)_a + (r_2)_a \\ (r_1 r_2)_a &= \begin{cases} (r_1)_a r_2 & \text{if } o_{\mathcal{R}}(r_1) = 0 \\ (r_1)_a r_2 + (r_2)_a & \text{otherwise} \end{cases} \\ (r^*)_a &= r_a r^* \end{aligned}$$

Are we done?

$$(a^*)^*$$

$$\begin{aligned}
((a^*)^*)_a &= (\underline{1}a^*)(a^*)^* \\
((\underline{1}a^*)(a^*)^*)_a &= (\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^* \\
((\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*)_a &= ((\underline{0}a^* + \underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*) + ((\underline{1}a^*)(a^*)^*)_a \\
&\vdots
\end{aligned}$$

Are we done?

$$(a^*)^*$$

$$\begin{aligned}
((a^*)^*)_a &= (\underline{1}a^*)(a^*)^* \\
((\underline{1}a^*)(a^*)^*)_a &= (\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^* \\
((\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*)_a &= ((\underline{0}a^* + \underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*) + ((\underline{1}a^*)(a^*)^*)_a \\
&\vdots
\end{aligned}$$

NOT FINITE!

Are we done?

$$(a^*)^*$$

$$\begin{aligned}
((a^*)^*)_a &= (\underline{1}a^*)(a^*)^* \\
((\underline{1}a^*)(a^*)^*)_a &= (\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^* \\
((\underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*)_a &= ((\underline{0}a^* + \underline{0}a^* + \underline{1}a^*)(a^*)^* + (\underline{1}a^*)(a^*)^*) + ((\underline{1}a^*)(a^*)^*)_a \\
&\vdots
\end{aligned}$$

NOT FINITE!

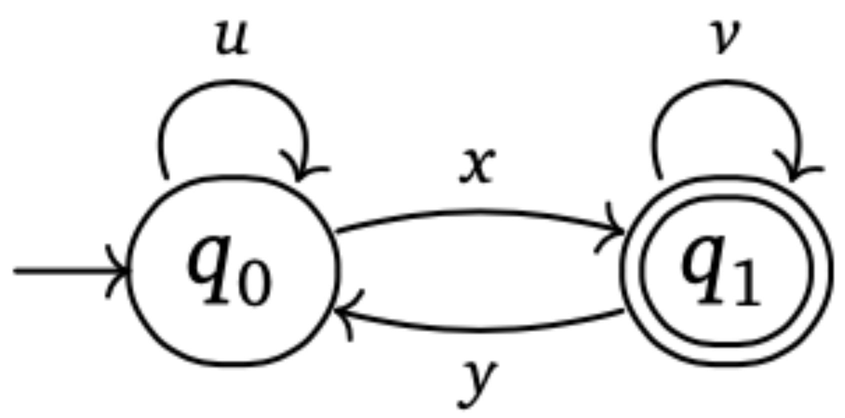
<u>Theorem</u> (Brzozowski): Let r be a regular expression. The set of syntactic Brzozowski derivatives is finite if it taken modulo ACI.

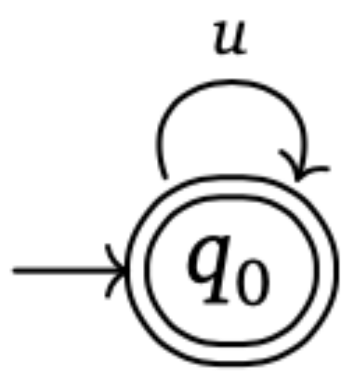
Intermezzo : final coalgebra

$$
\begin{array}{ccc}
\mathcal{R}(A) & \overset{L}{\dashrightarrow} & 2^{A^*} \\
\Big\downarrow {\scriptstyle \langle o_{\mathcal{R}}, t_{\mathcal{R}} \rangle} & & \Big\downarrow {\scriptstyle \langle o_L, t_L \rangle} \\
2 \times (\mathcal{R}(A))^A & \underset{id \times L^A}{\dashrightarrow} & 2 \times (2^{A^*})^A
\end{array}
$$

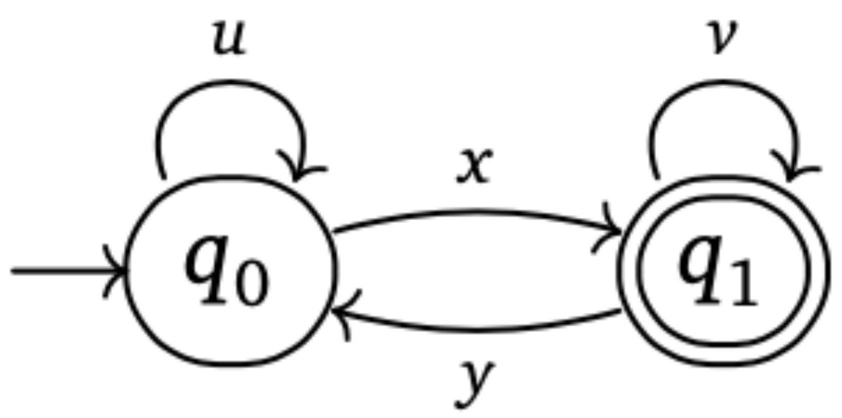
Alternative proof 1=>2
Thompson + epsilon-elim + subset construction

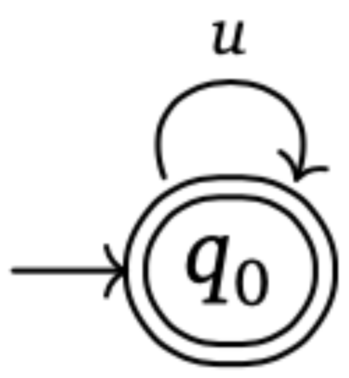
State Elimination





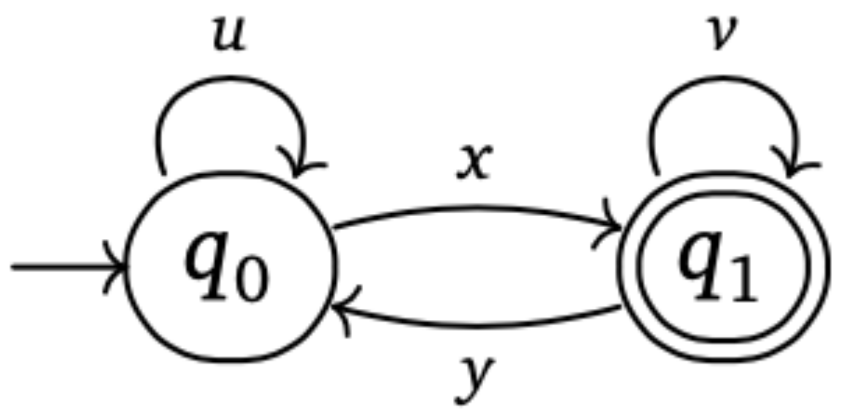
State Elimination

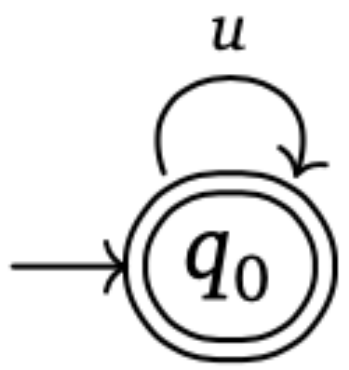




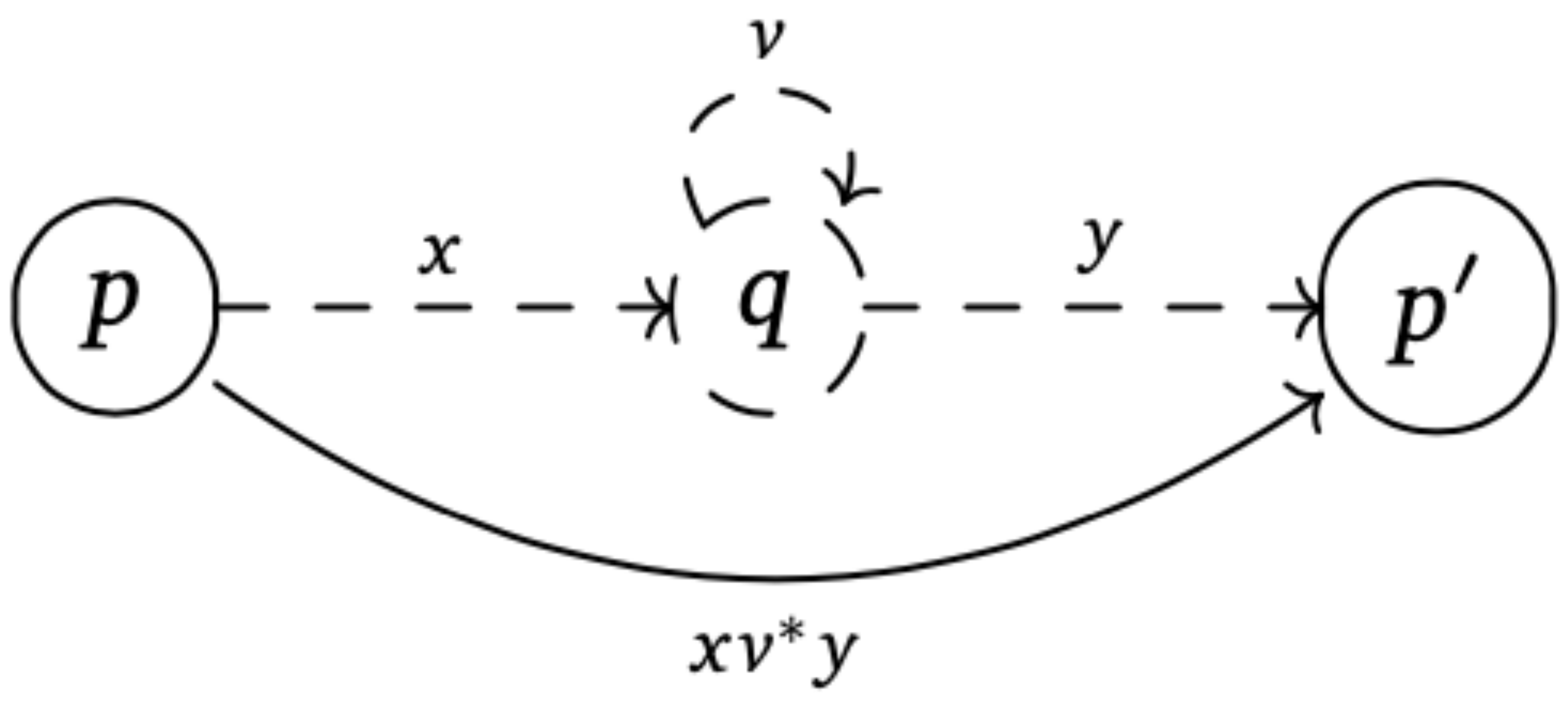
$$u^*x(v + yu^*x)^*$$

State Elimination

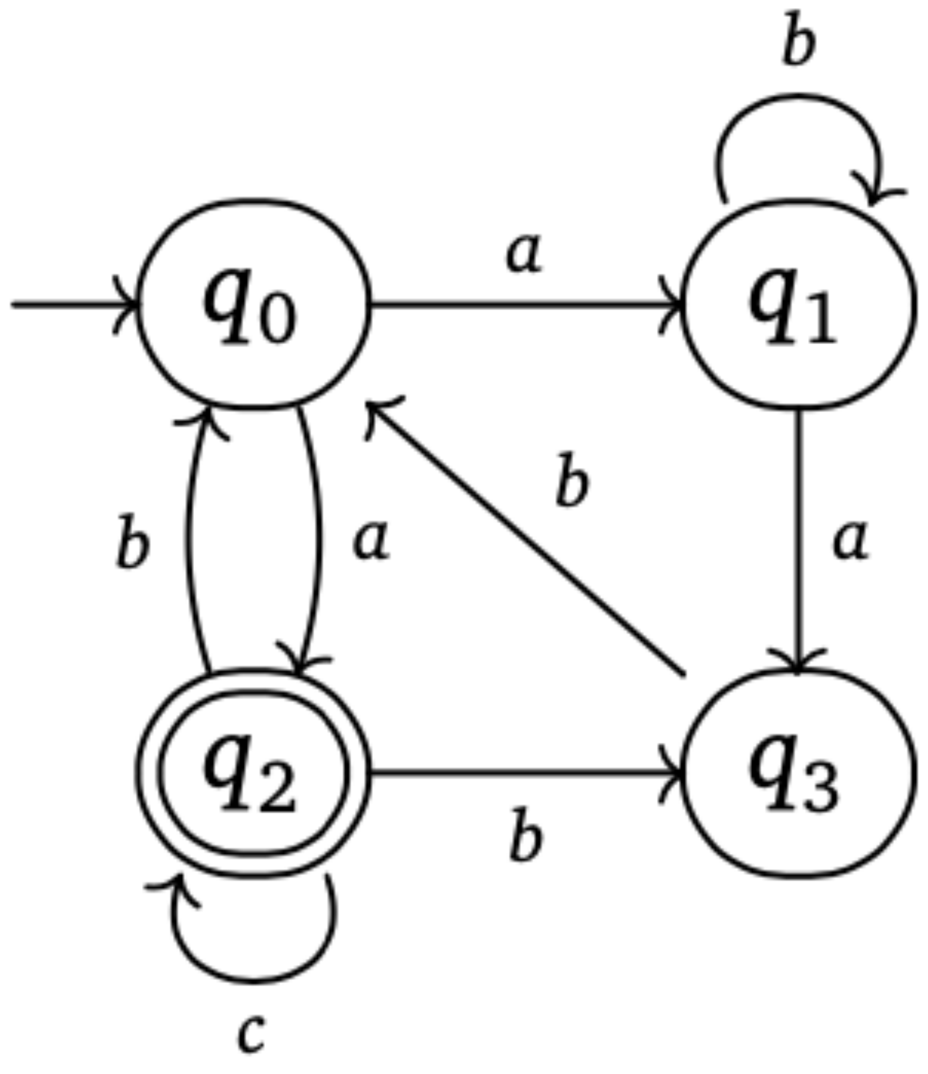




$$u^*x(v + yu^*x)^*$$

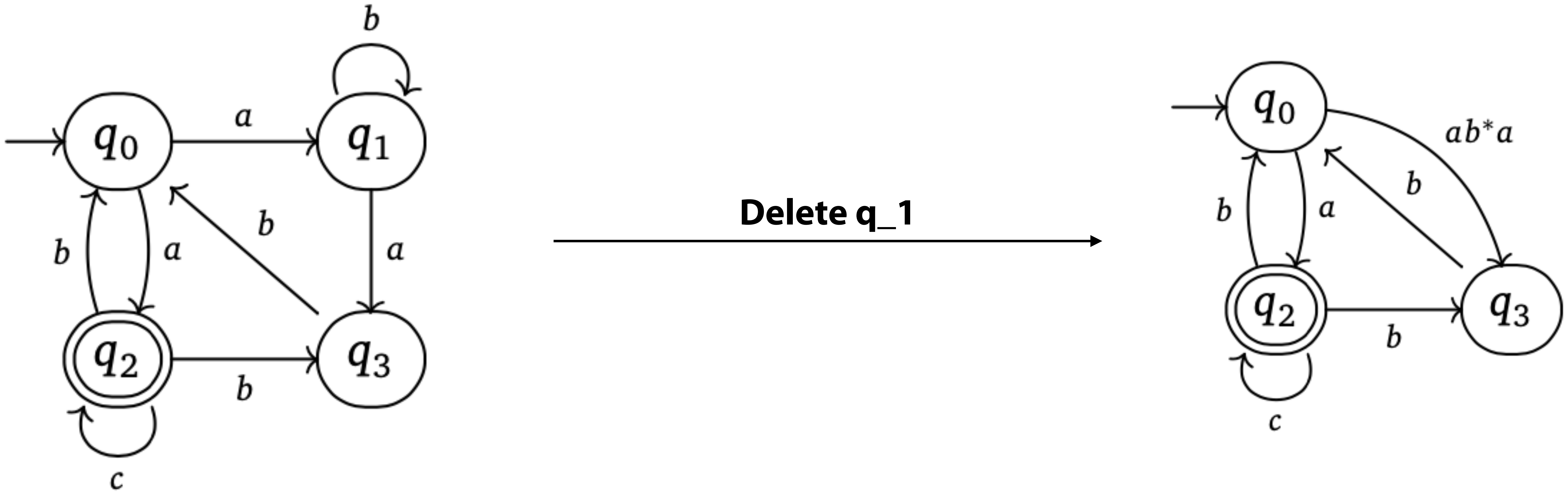


Exercise



Delete q_1

Exercise



Lecture 2

Alternative proof 2 => 1
Solving systems of equations

$$x \equiv r_1 + r_2 x$$

Alternative proof 2 => 1
Solving systems of equations

$$x \equiv r_1 + r_2 x$$

$$r_s \equiv \sum_{a \in A} a r_{s_a} + \underline{o_S(s)}$$

Alternative proof 2 => 1
Solving systems of equations

$$x \equiv r_1 + r_2 x$$

$$r_s \equiv \sum_{a \in A} a r_{s_a} + \underline{o_S(s)}$$

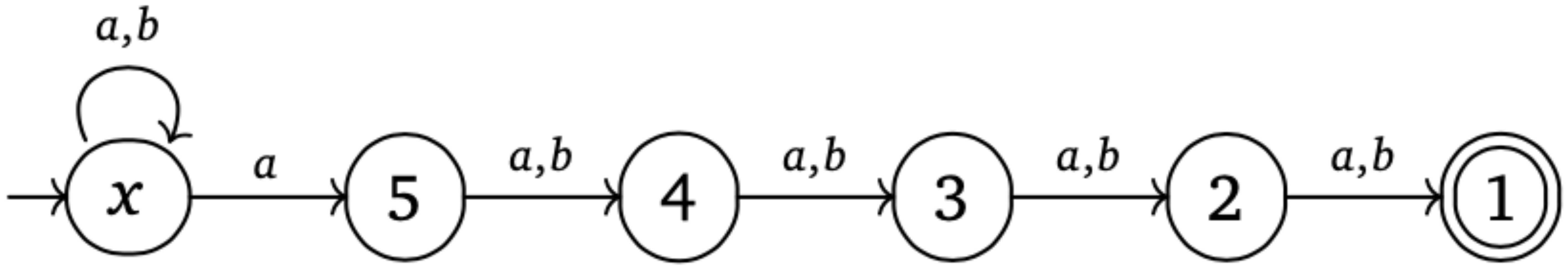
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^* = \begin{bmatrix} (a + bd^* c)^* & (a + bd^* c)^* bd^* \\ (d + ca^* b)^* ca^* & (d + ca^* b)^* \end{bmatrix}$$

NFAs

$$(a+b)^*a(a+b)(a+b)(a+b)(a+b)$$

NFAs

$$(a+b)^*a(a+b)(a+b)(a+b)(a+b)$$



Antimirov Derivatives

Context

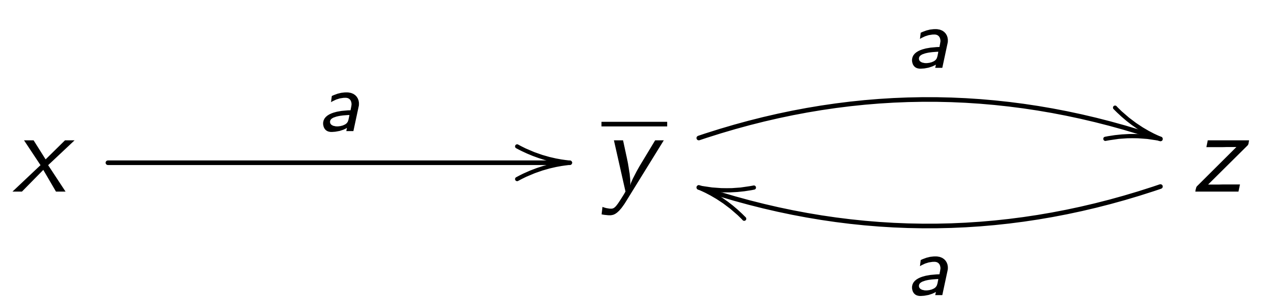
Tools and proof techniques for systems equivalence

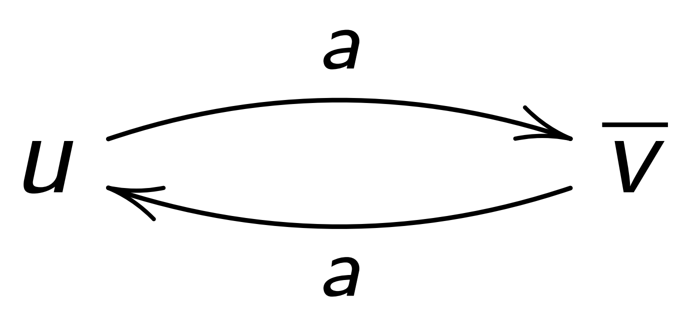
Methodology:

1. First do it naively
2. Then improve the associated proof method

Deterministic finite automata

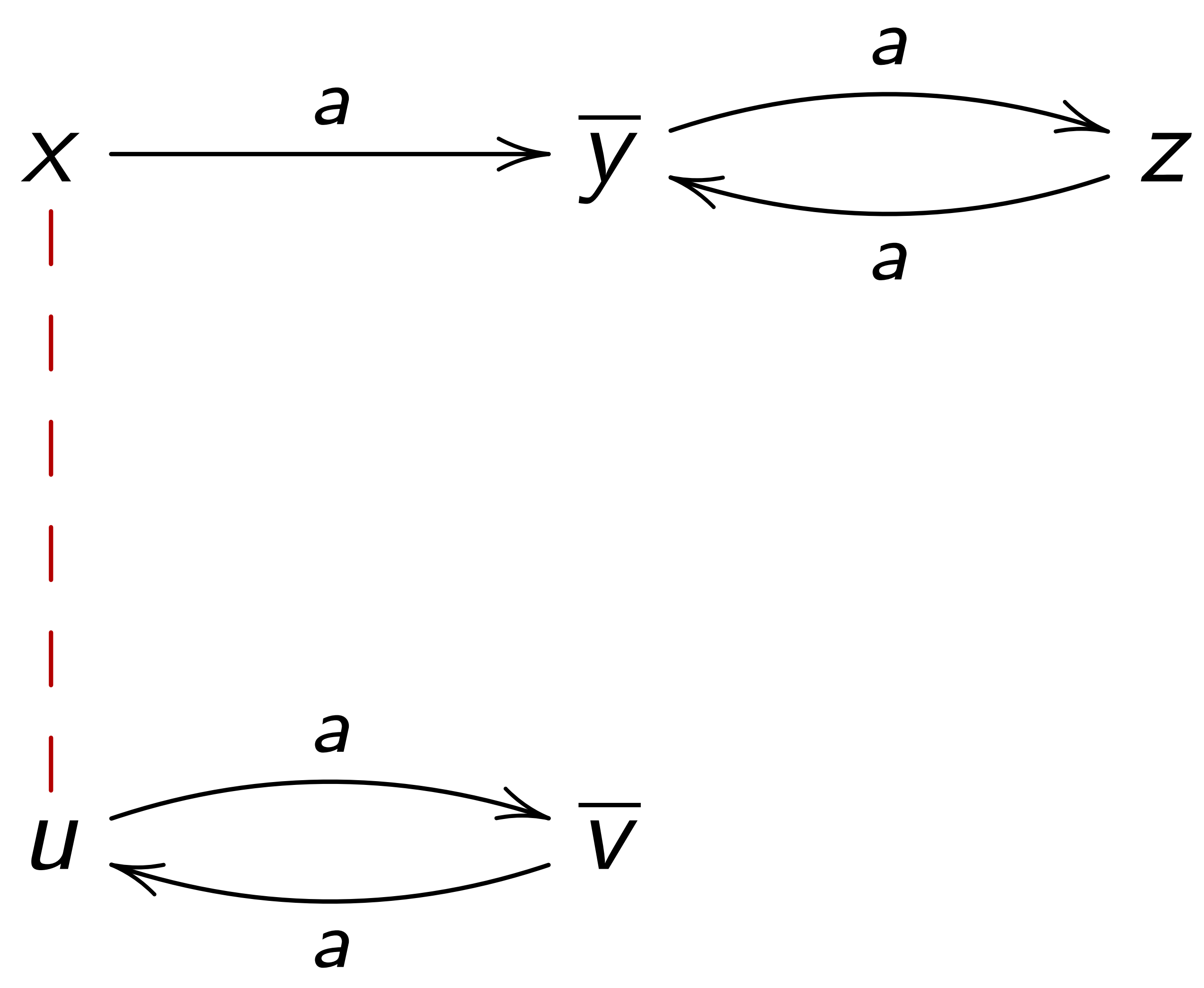
The states x and u are language equivalent





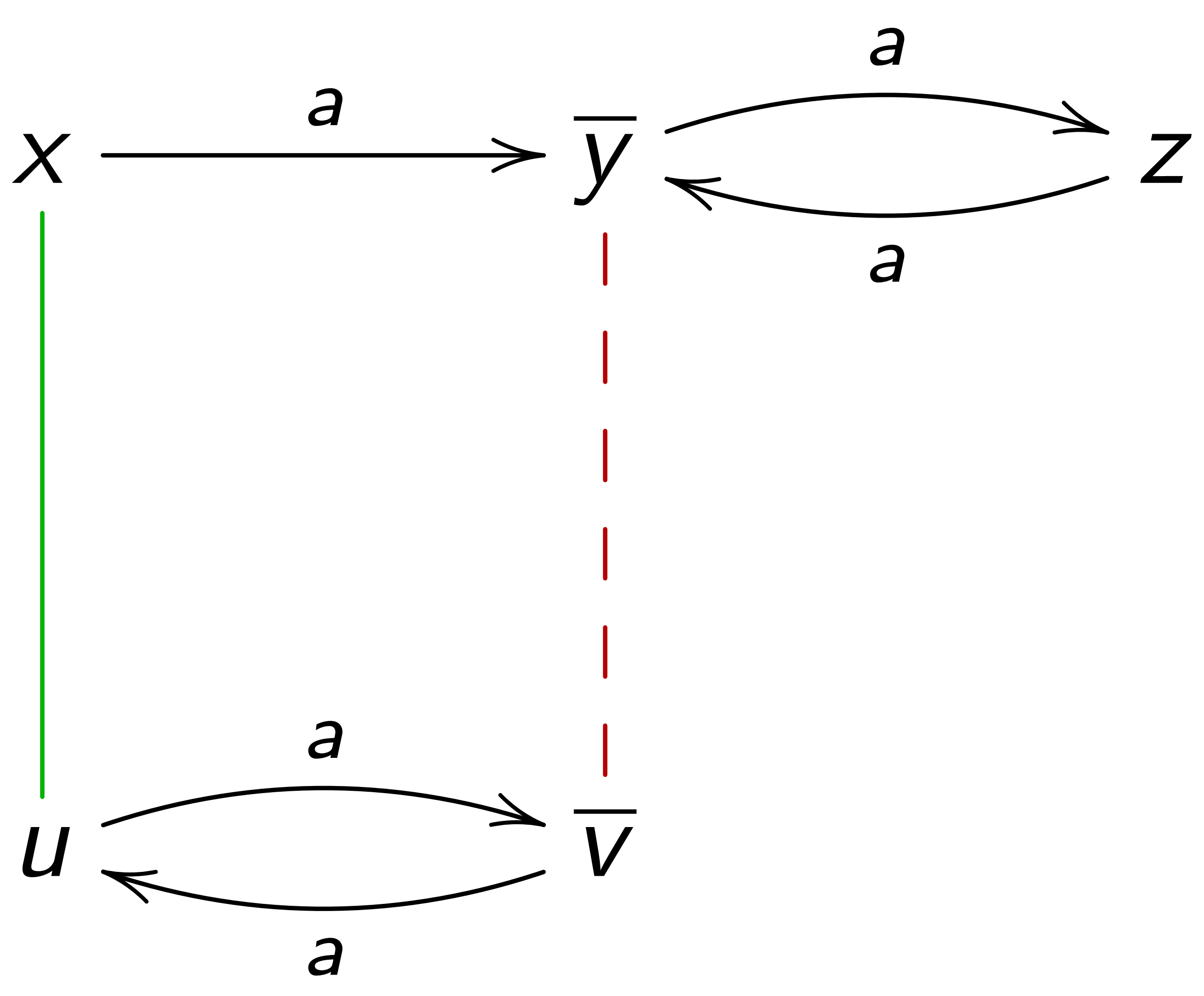
Deterministic finite automata

The states x and u are language equivalent



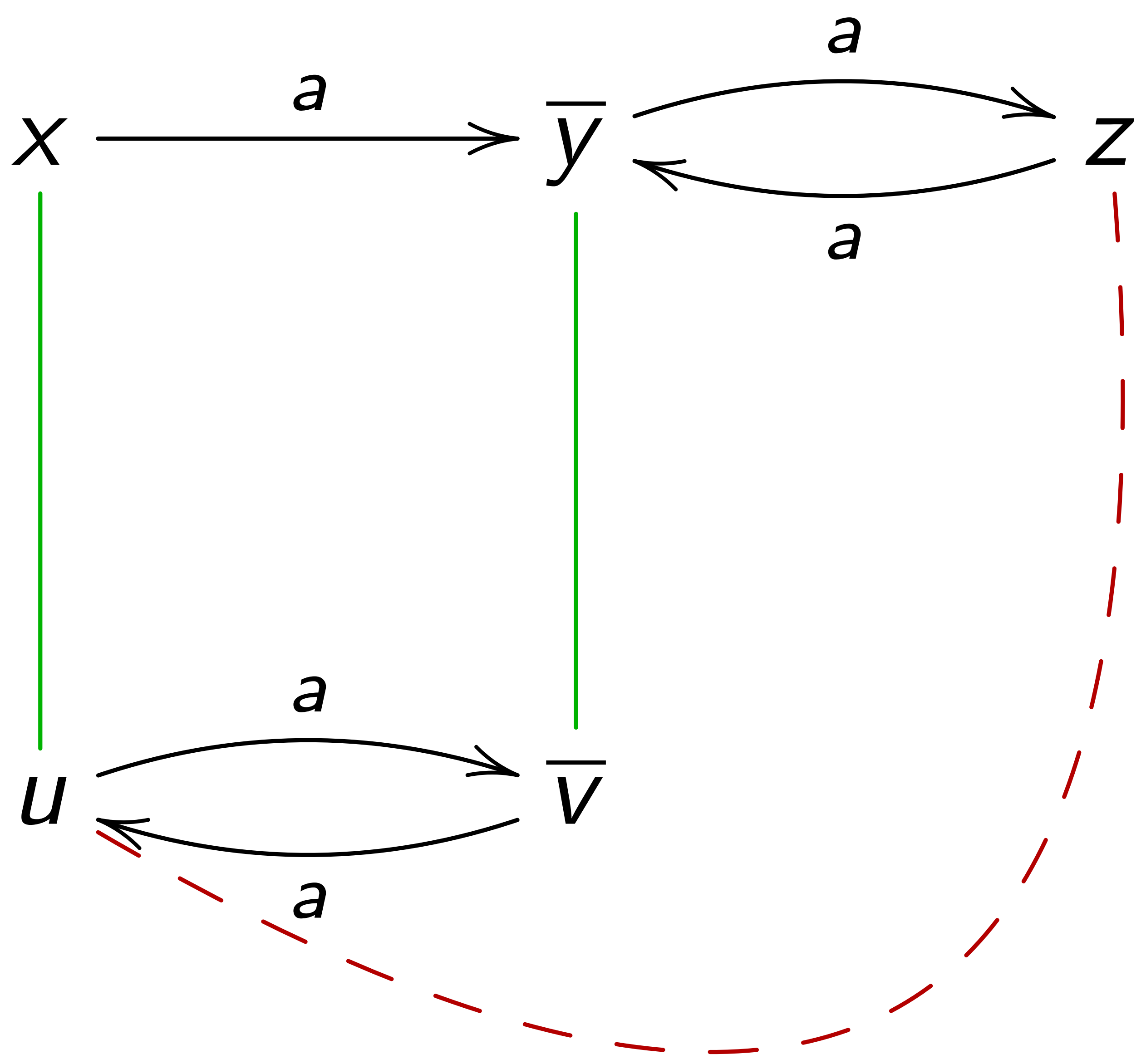
Deterministic finite automata

The states x and u are language equivalent



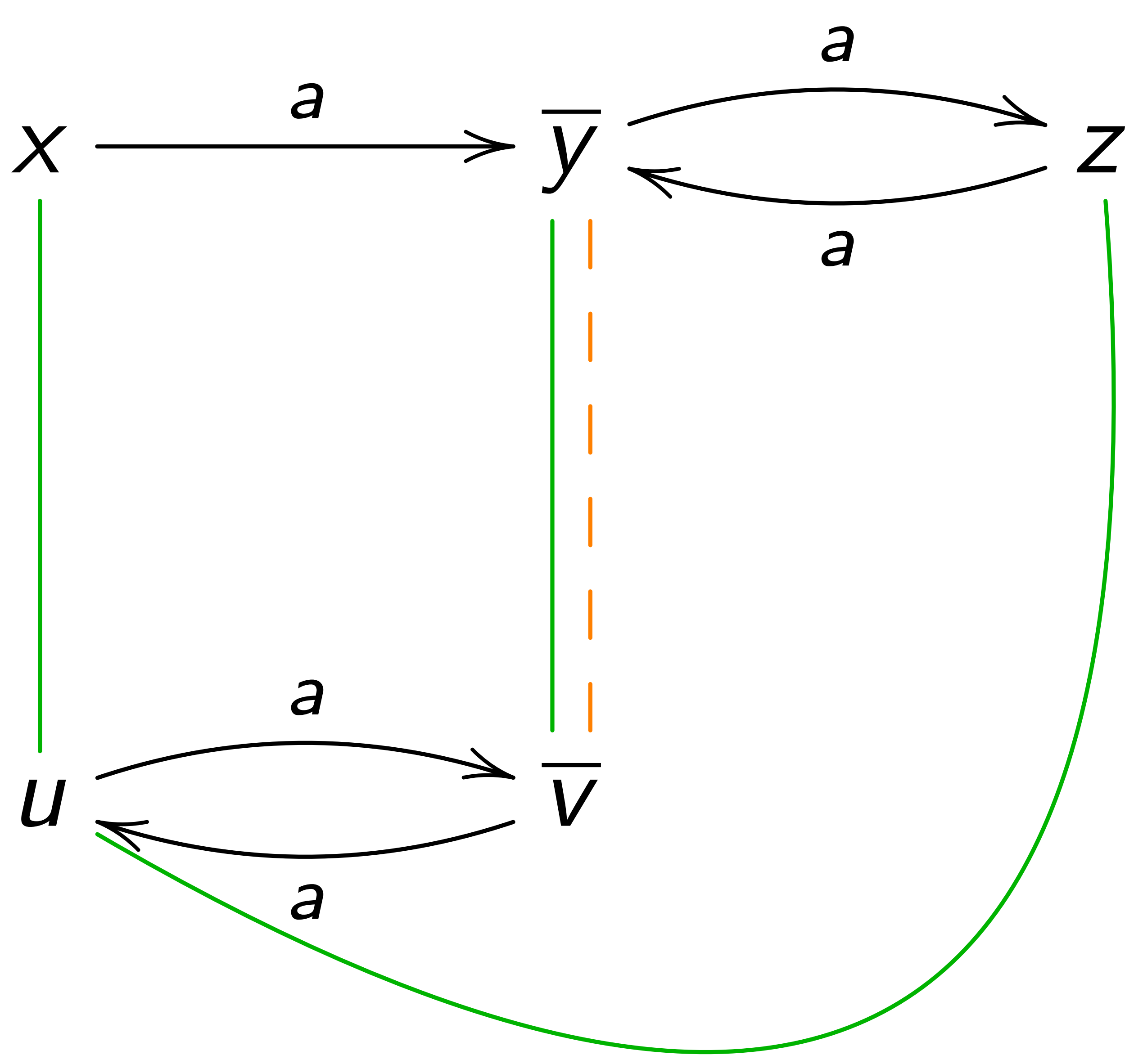
Deterministic finite automata

The states x and u are language equivalent



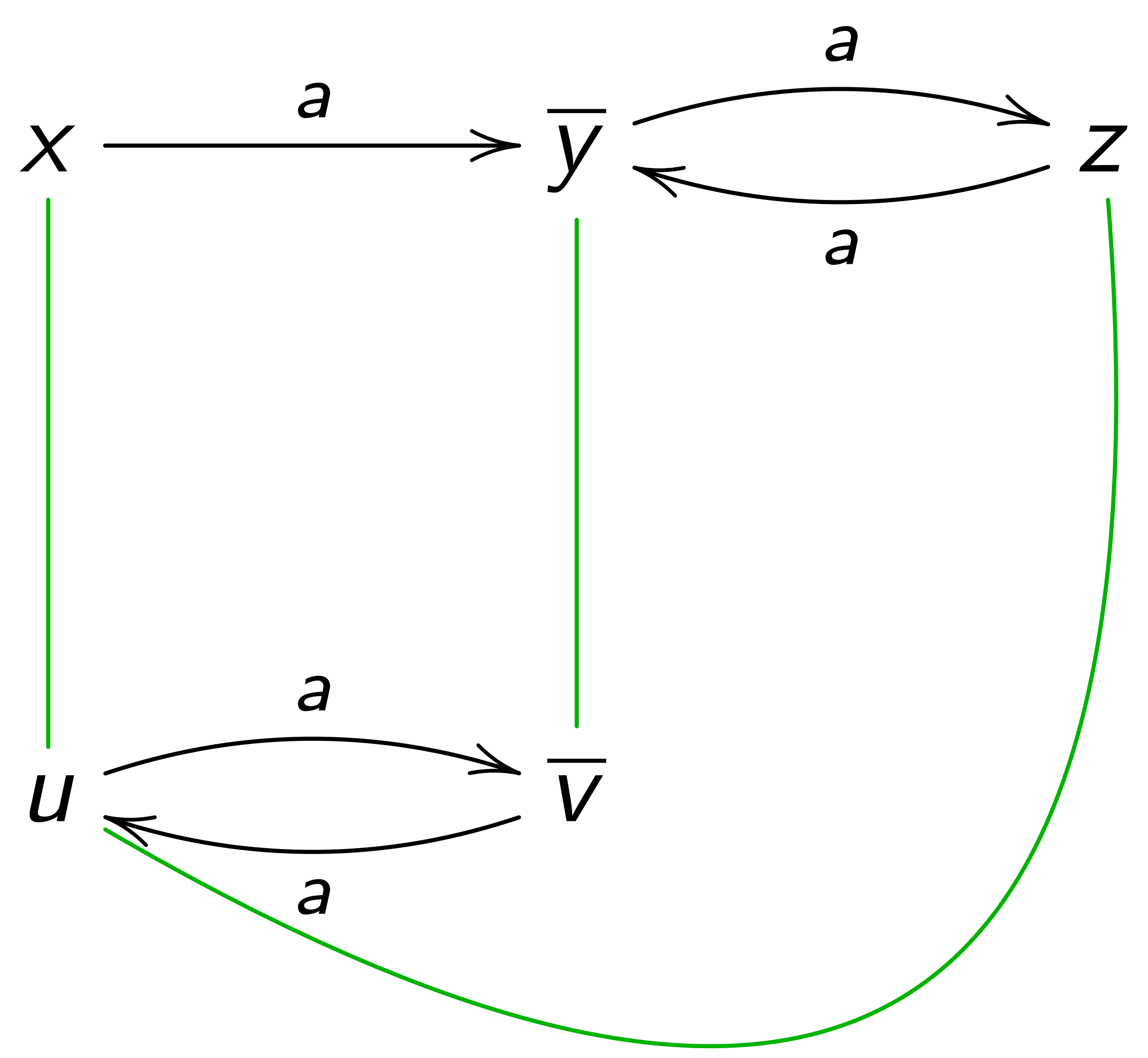
Deterministic finite automata

The states x and u are language equivalent



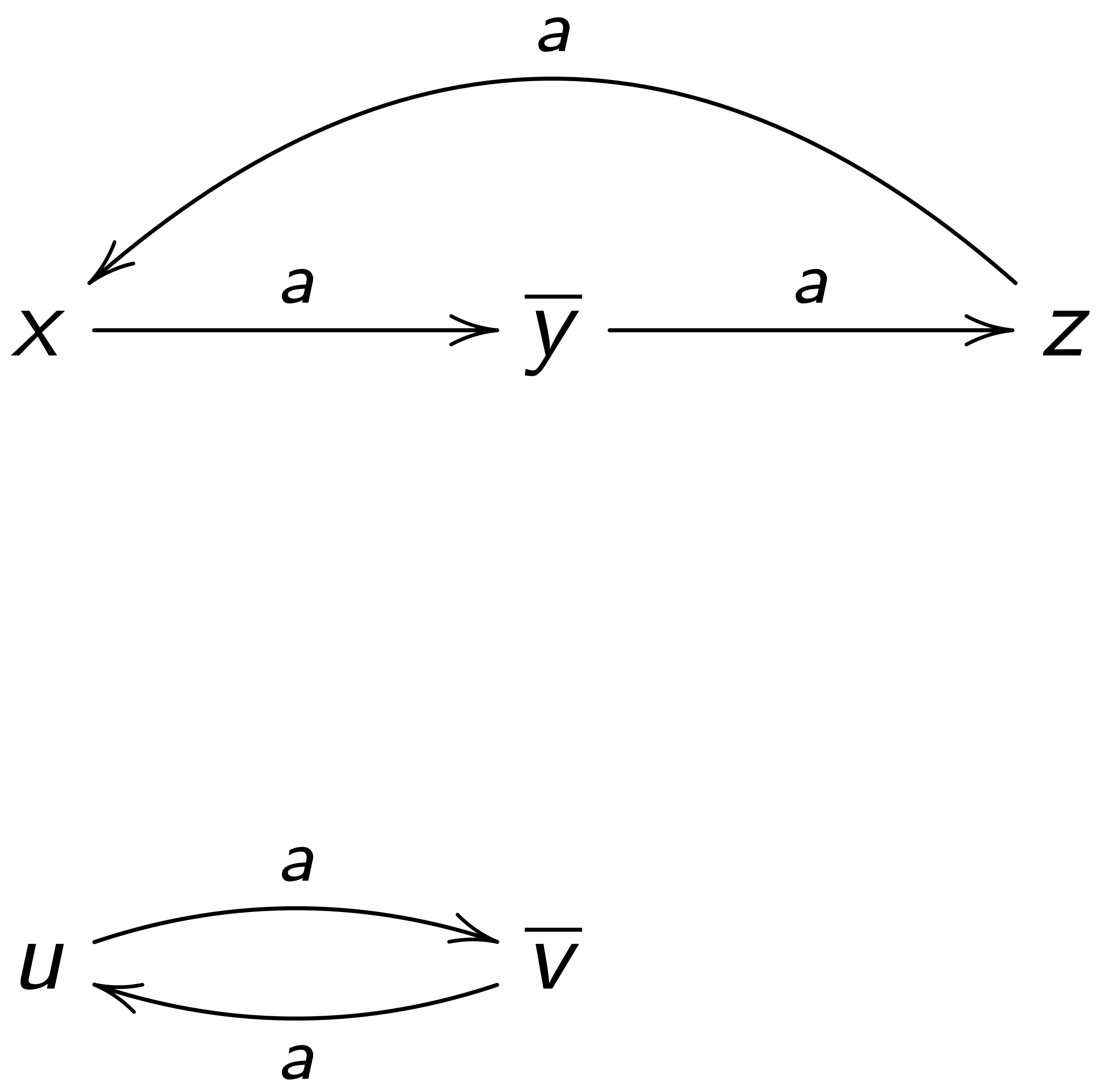
Deterministic finite automata

The states x and u are language equivalent



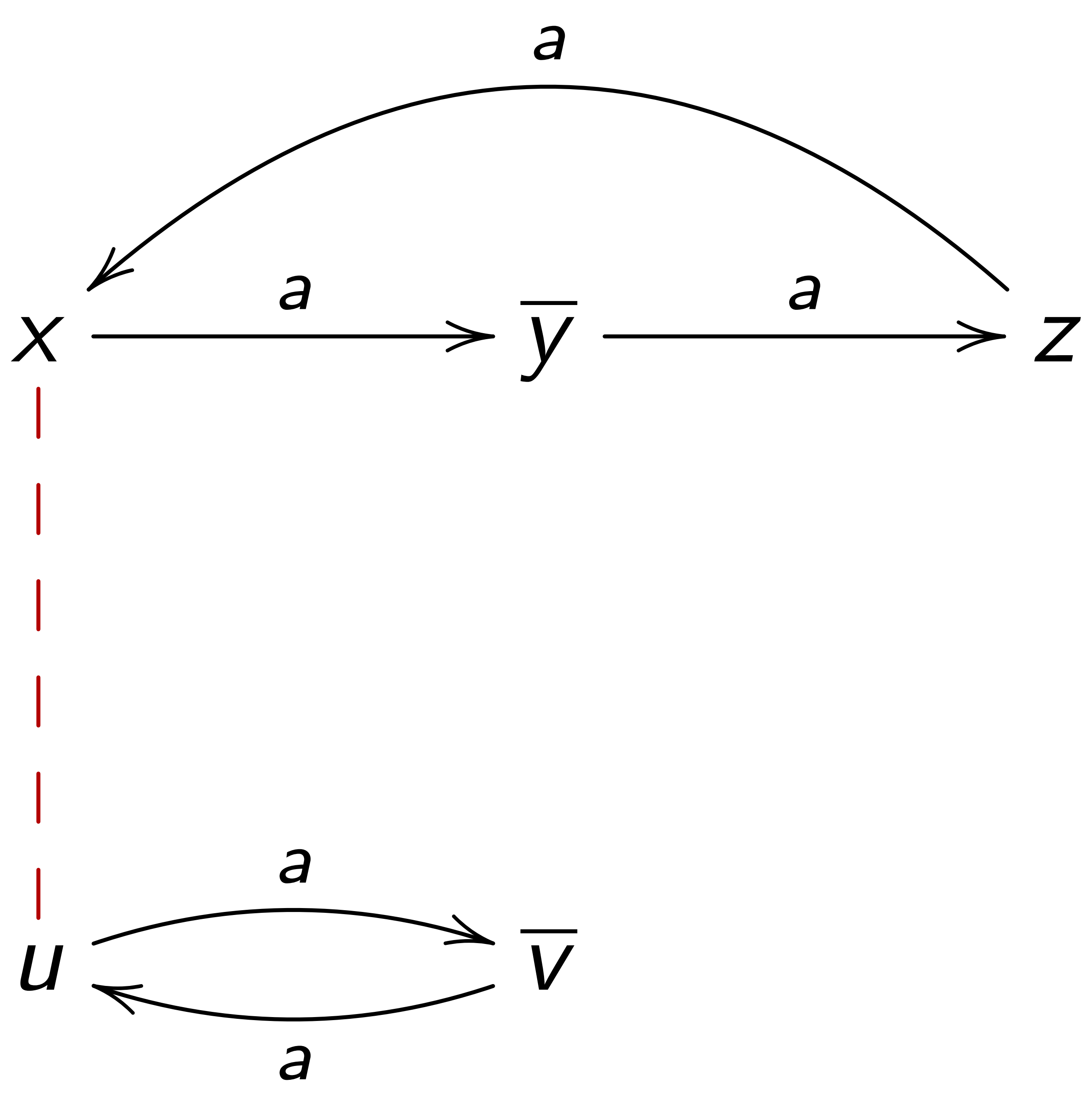
Deterministic finite automata

x and u are **not** equivalent



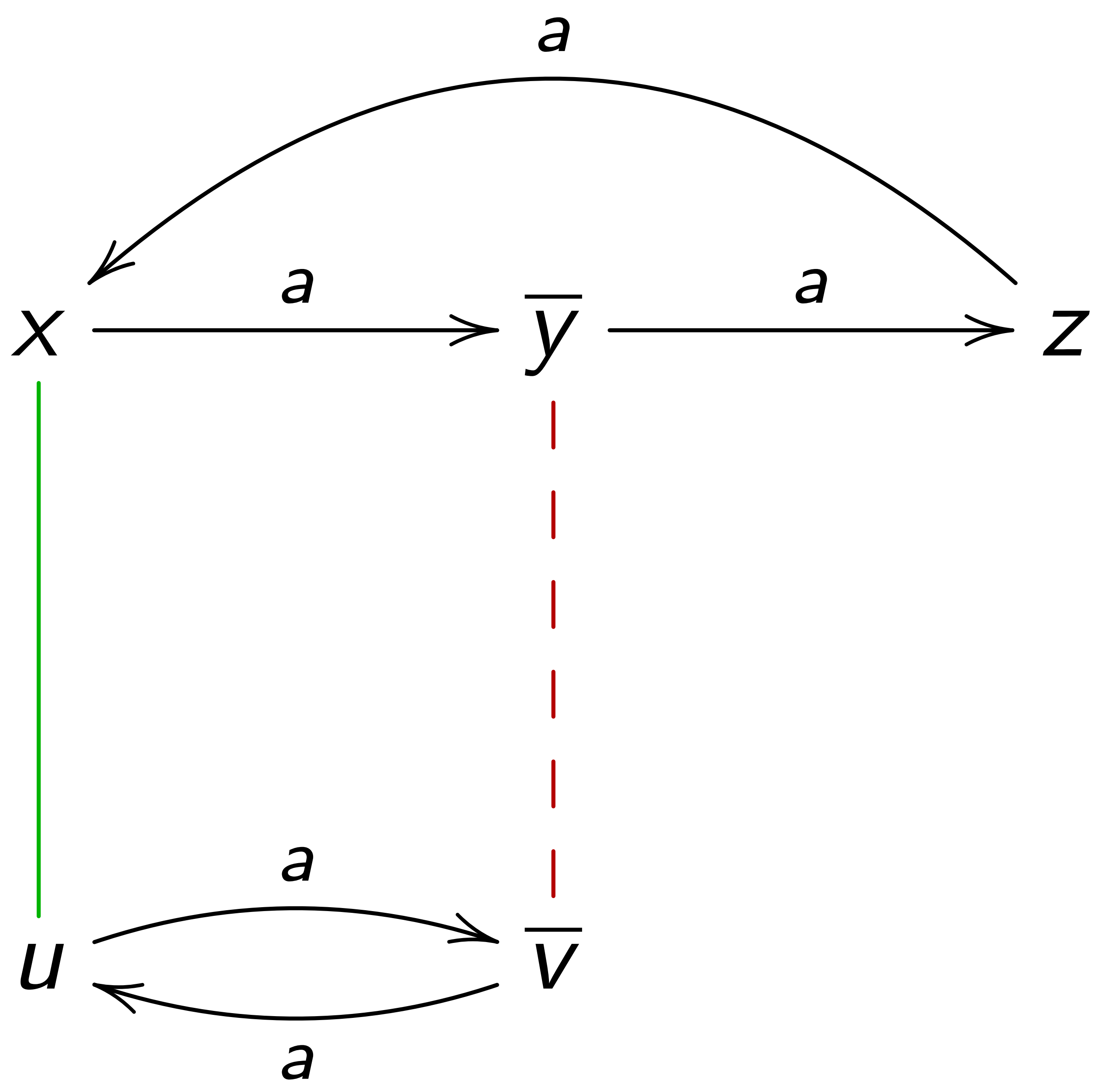
Deterministic finite automata

x and u are **not** equivalent



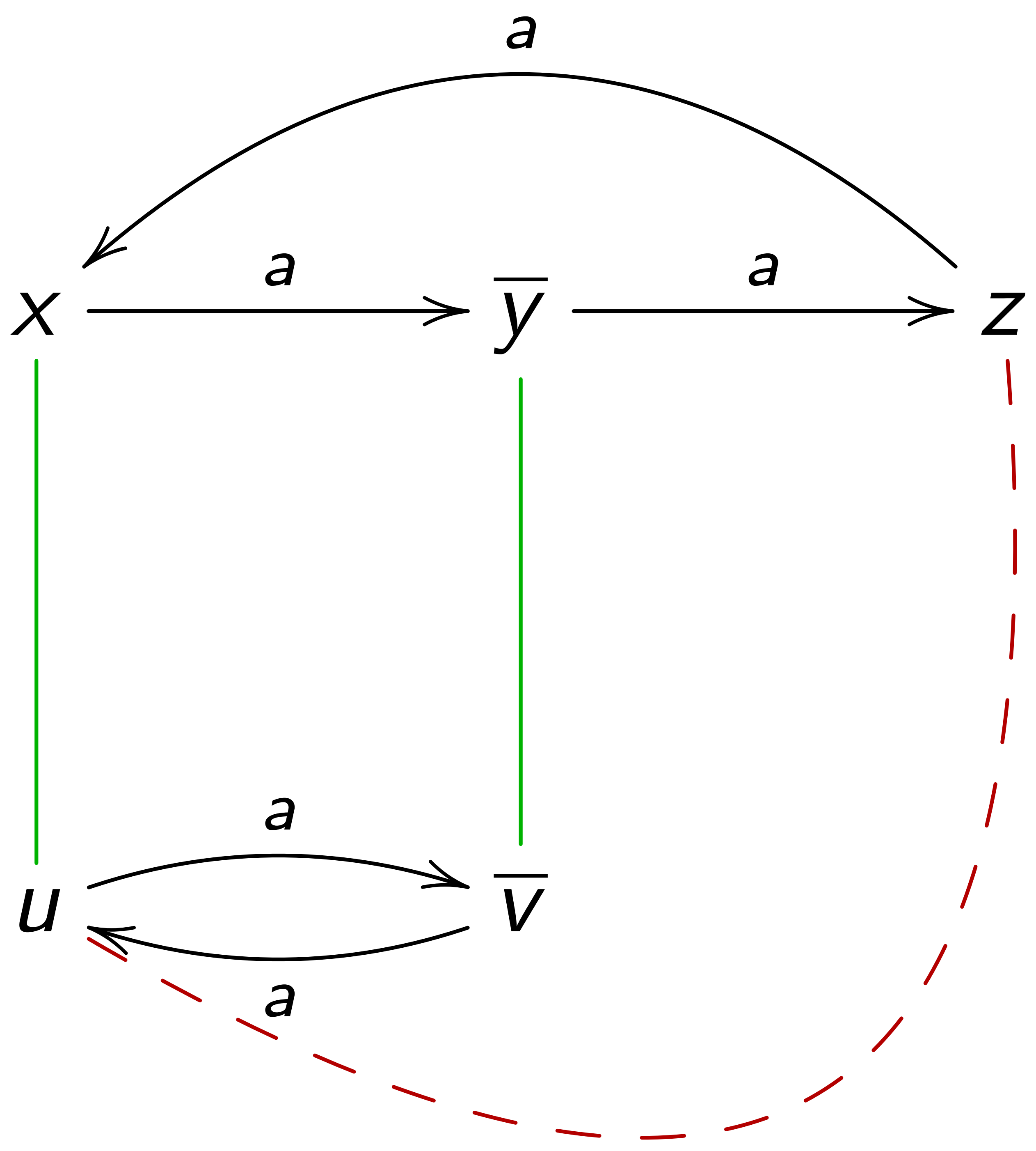
Deterministic finite automata

x and u are **not** equivalent



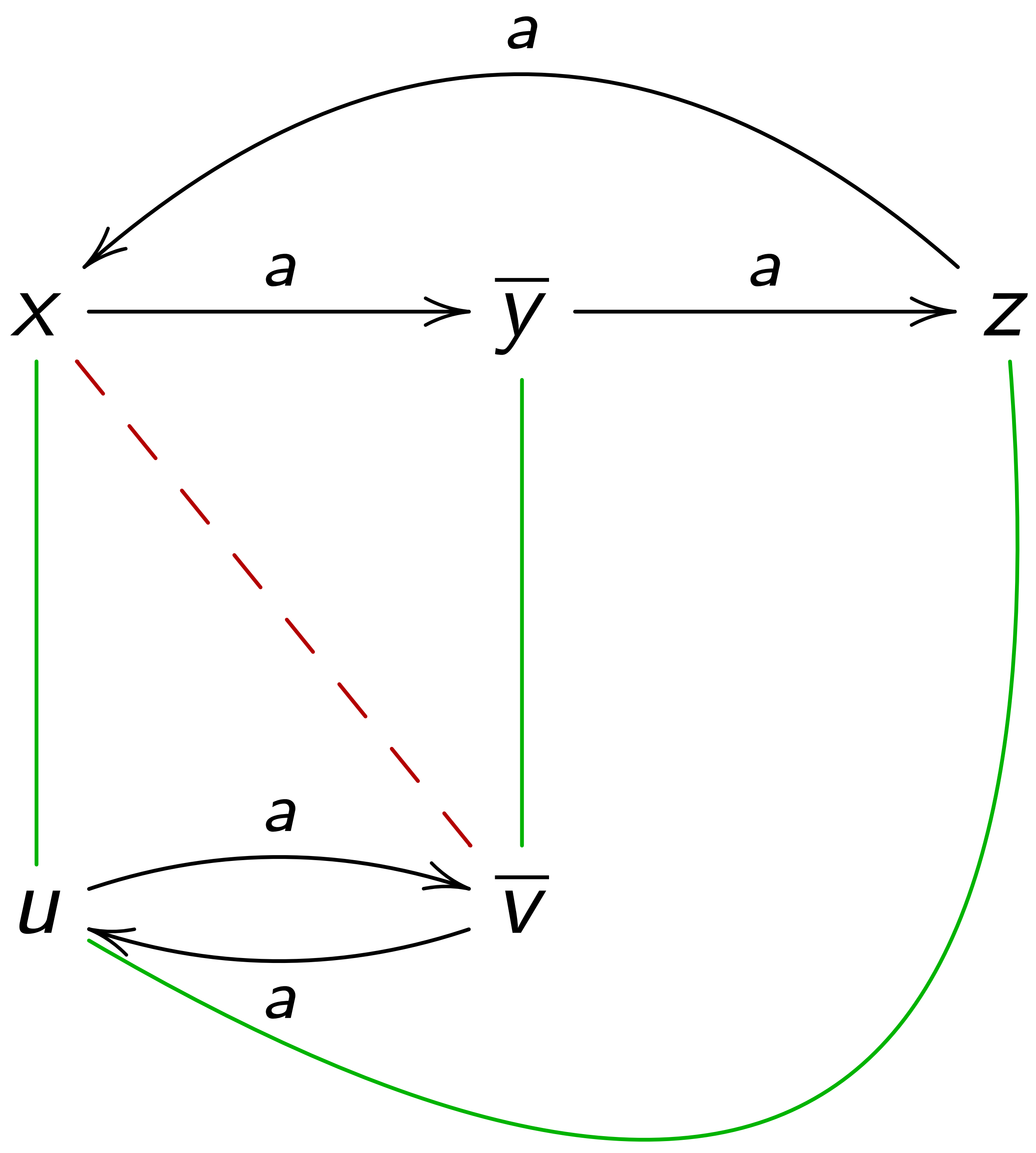
Deterministic finite automata

x and u are **not** equivalent



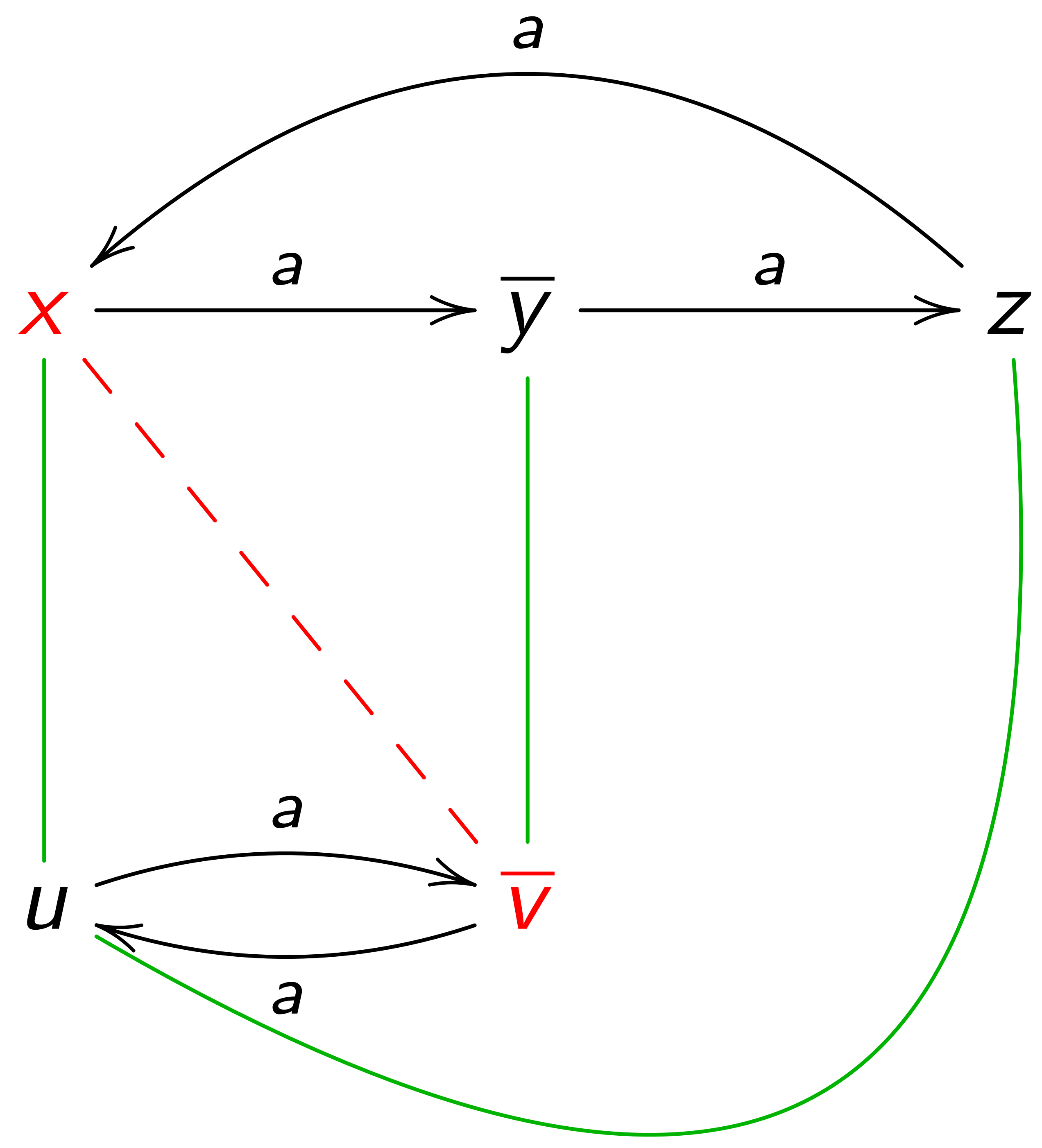
Deterministic finite automata

x and u are **not** equivalent



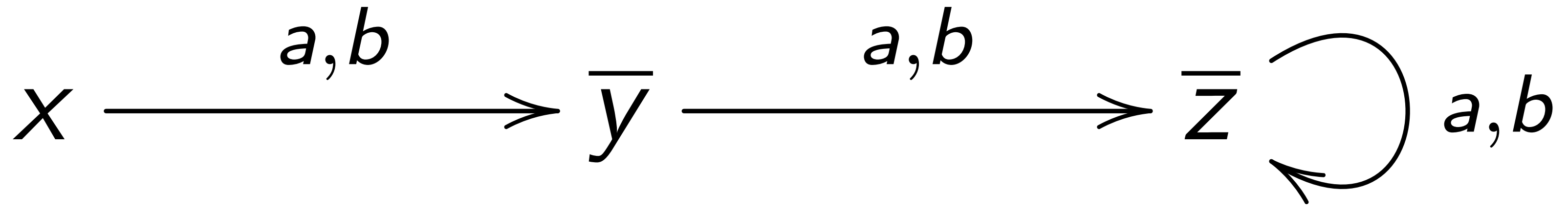
Deterministic finite automata

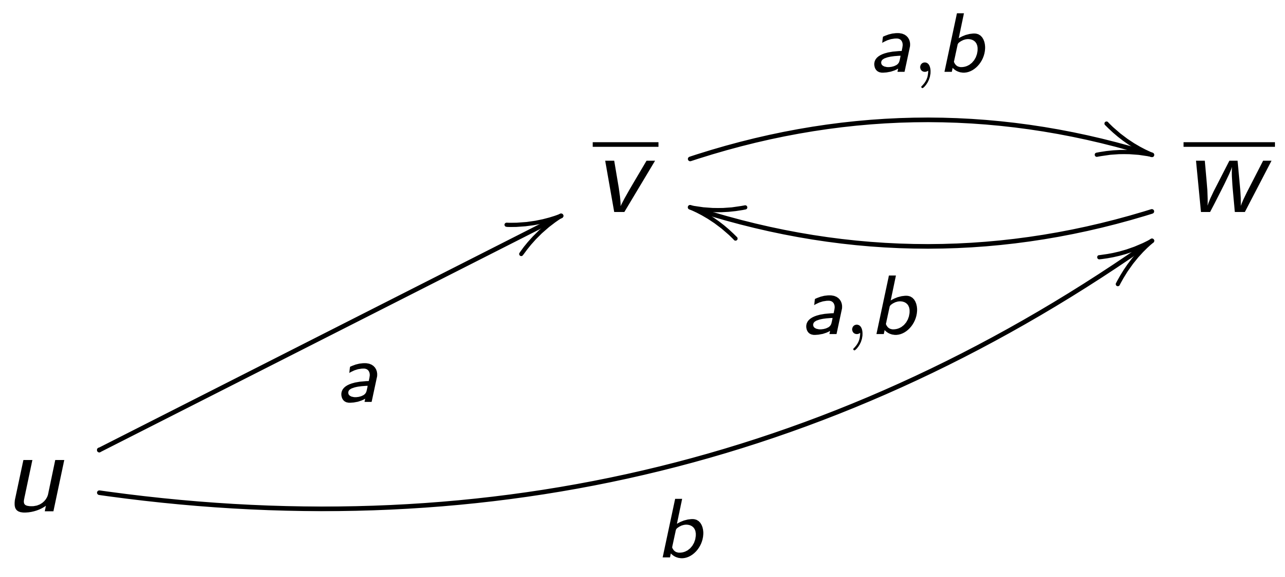
x and u are **not** equivalent



Deterministic finite automata

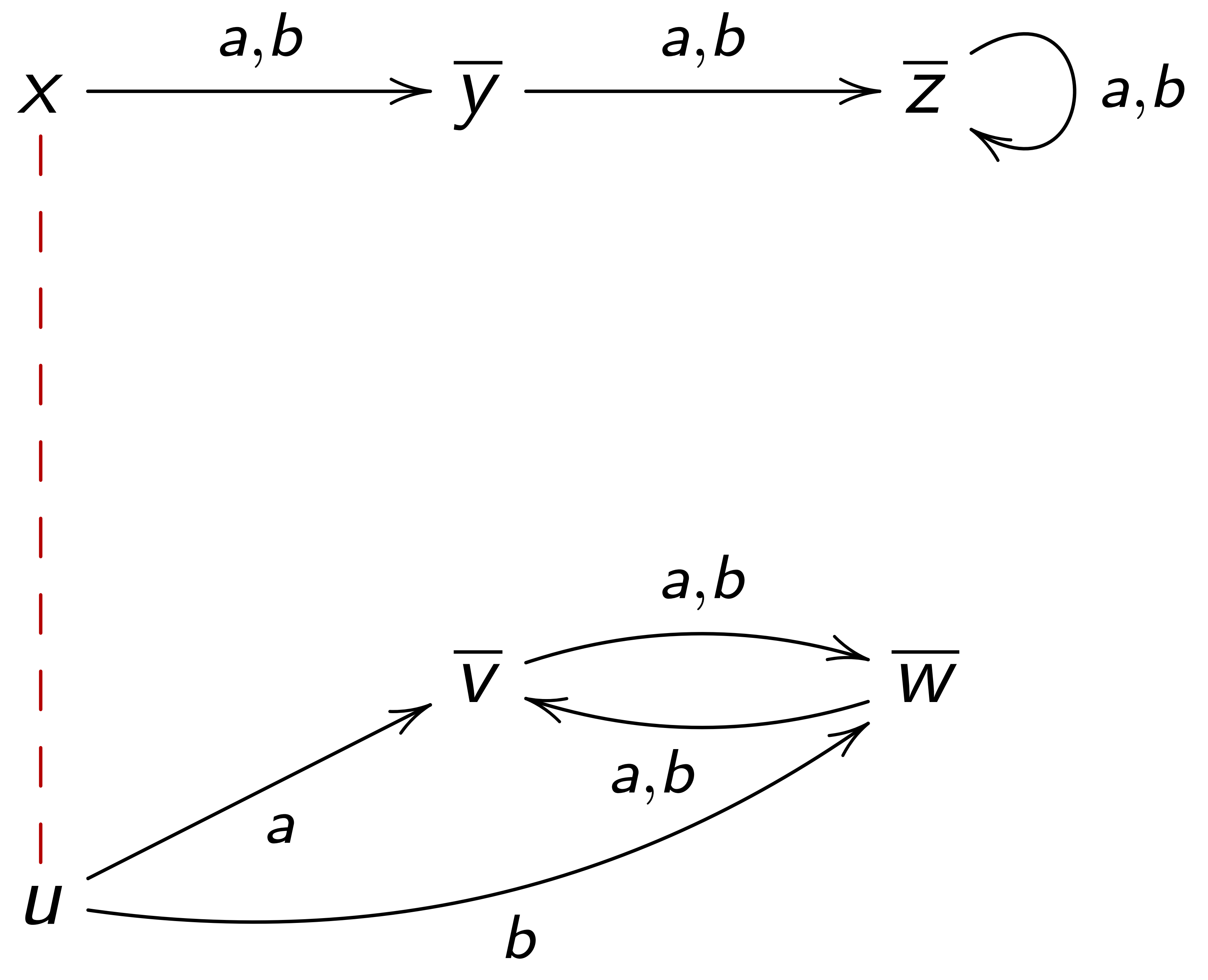
Last example, with two letters





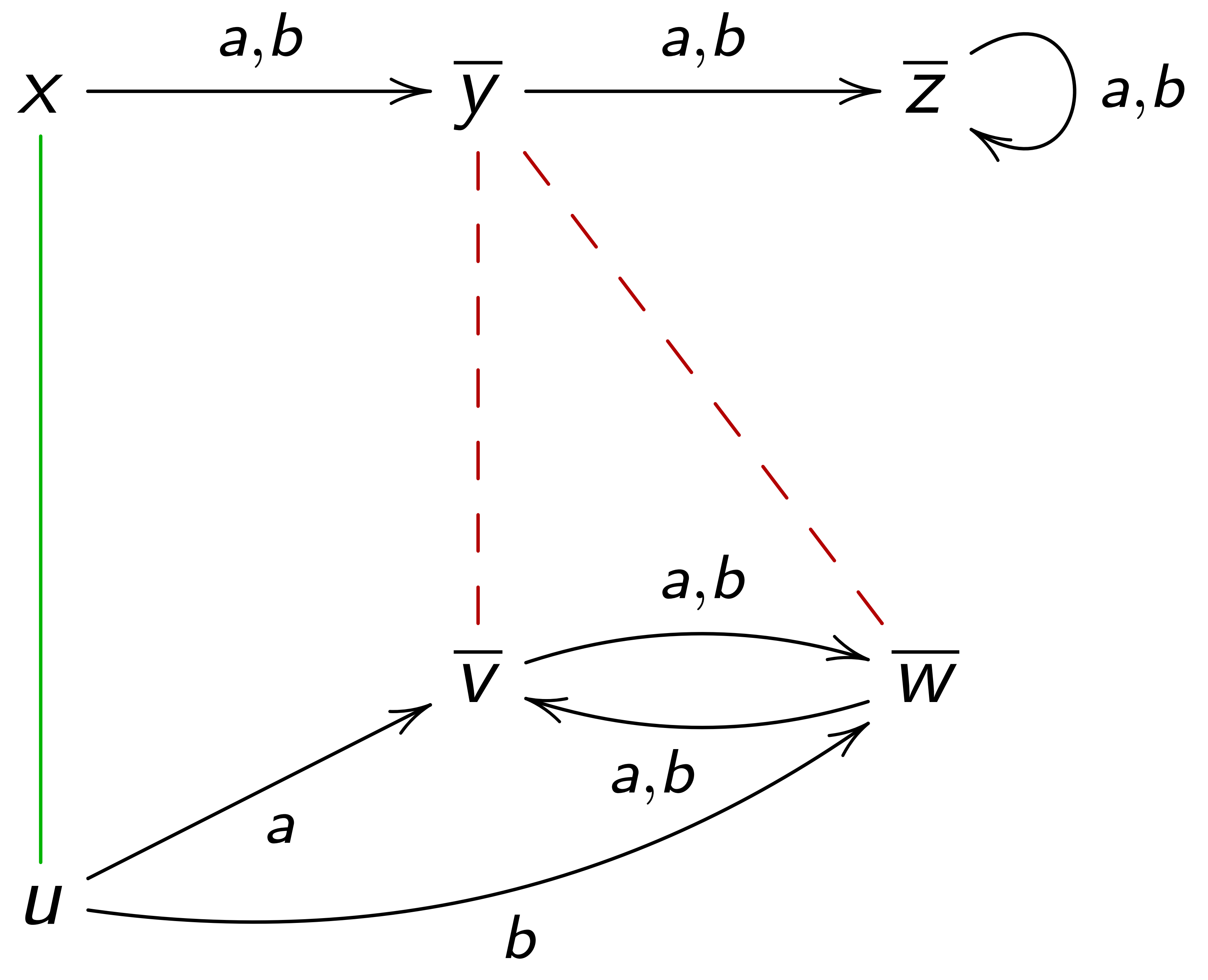
Deterministic finite automata

Last example, with two letters



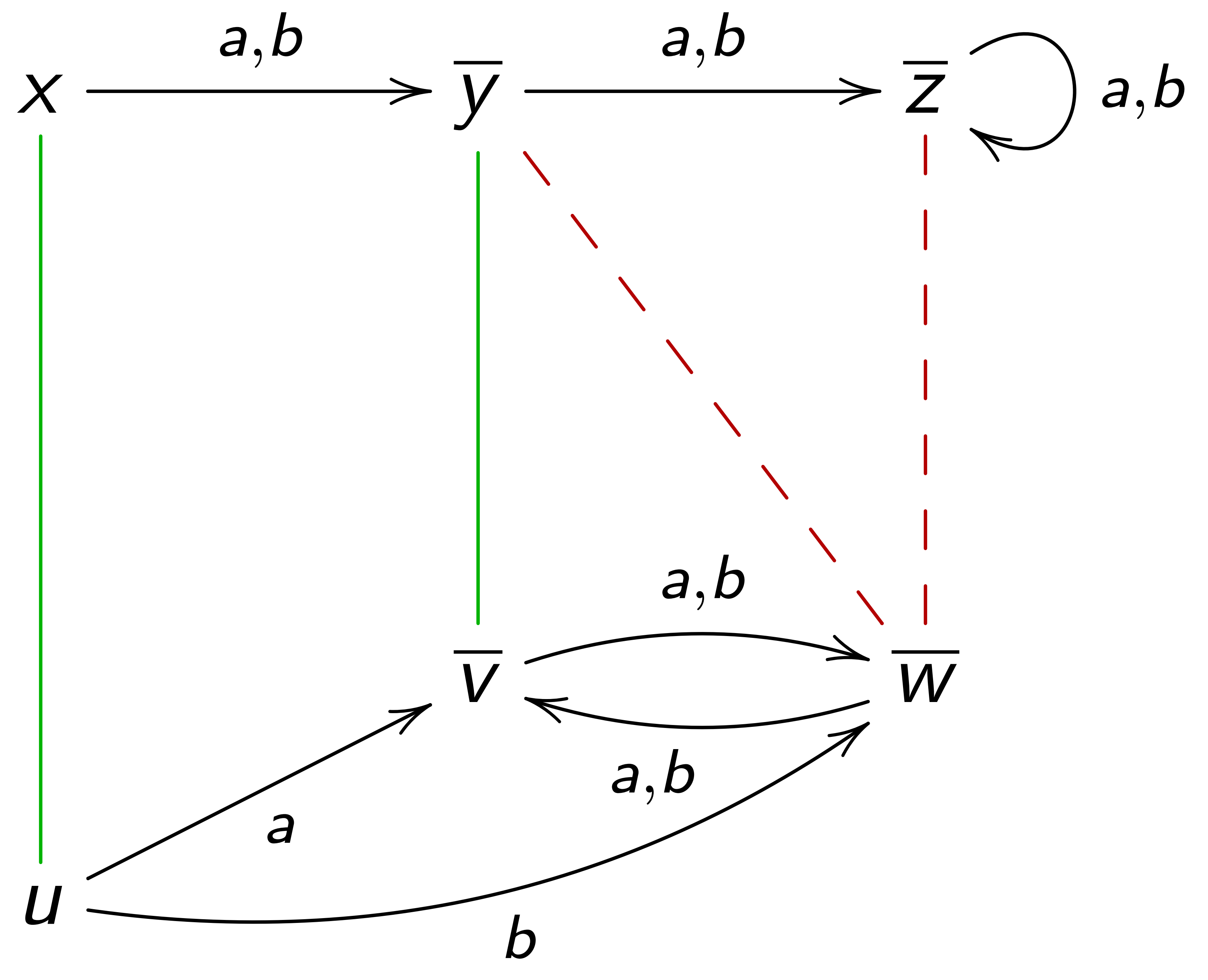
Deterministic finite automata

Last example, with two letters



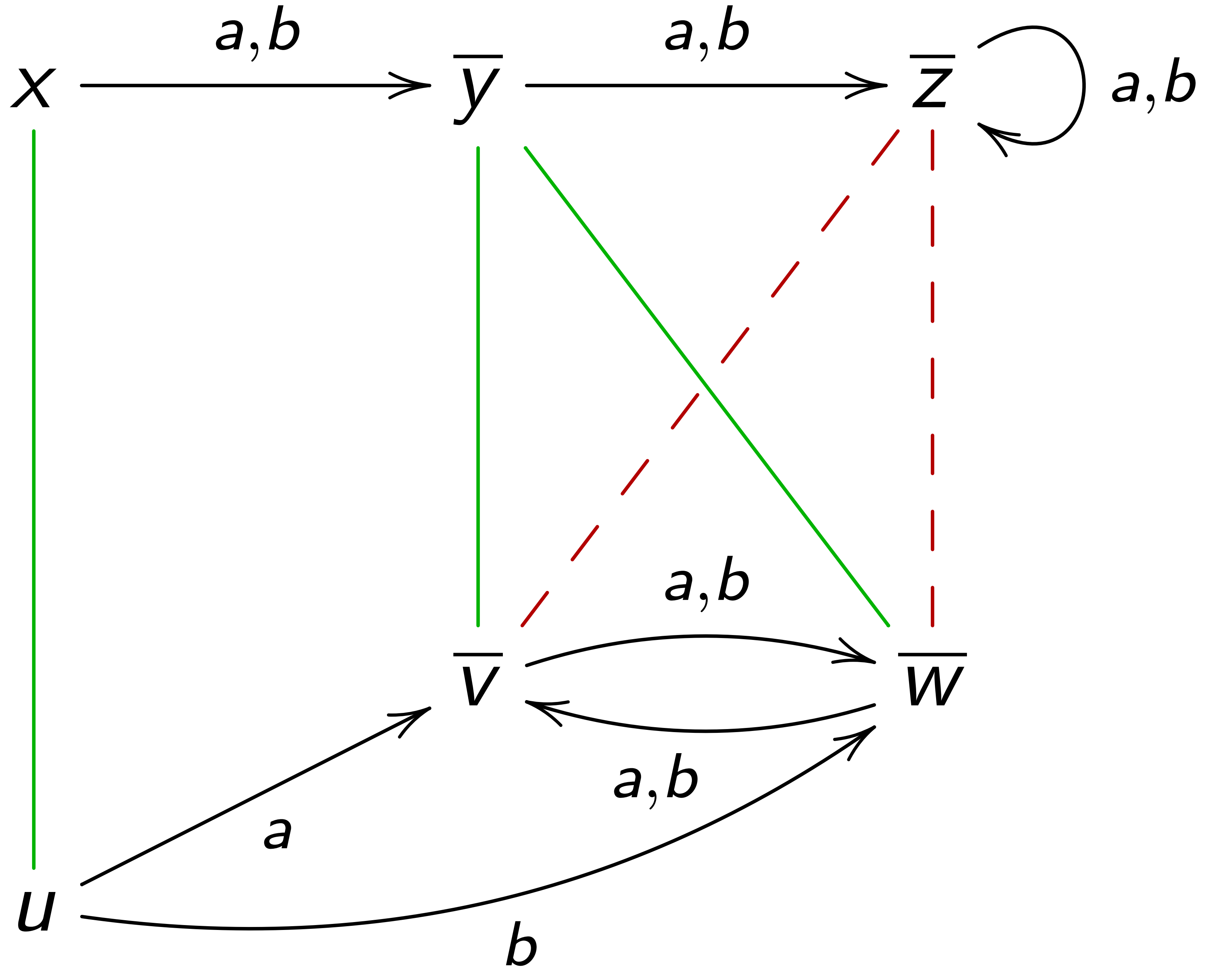
Deterministic finite automata

Last example, with two letters



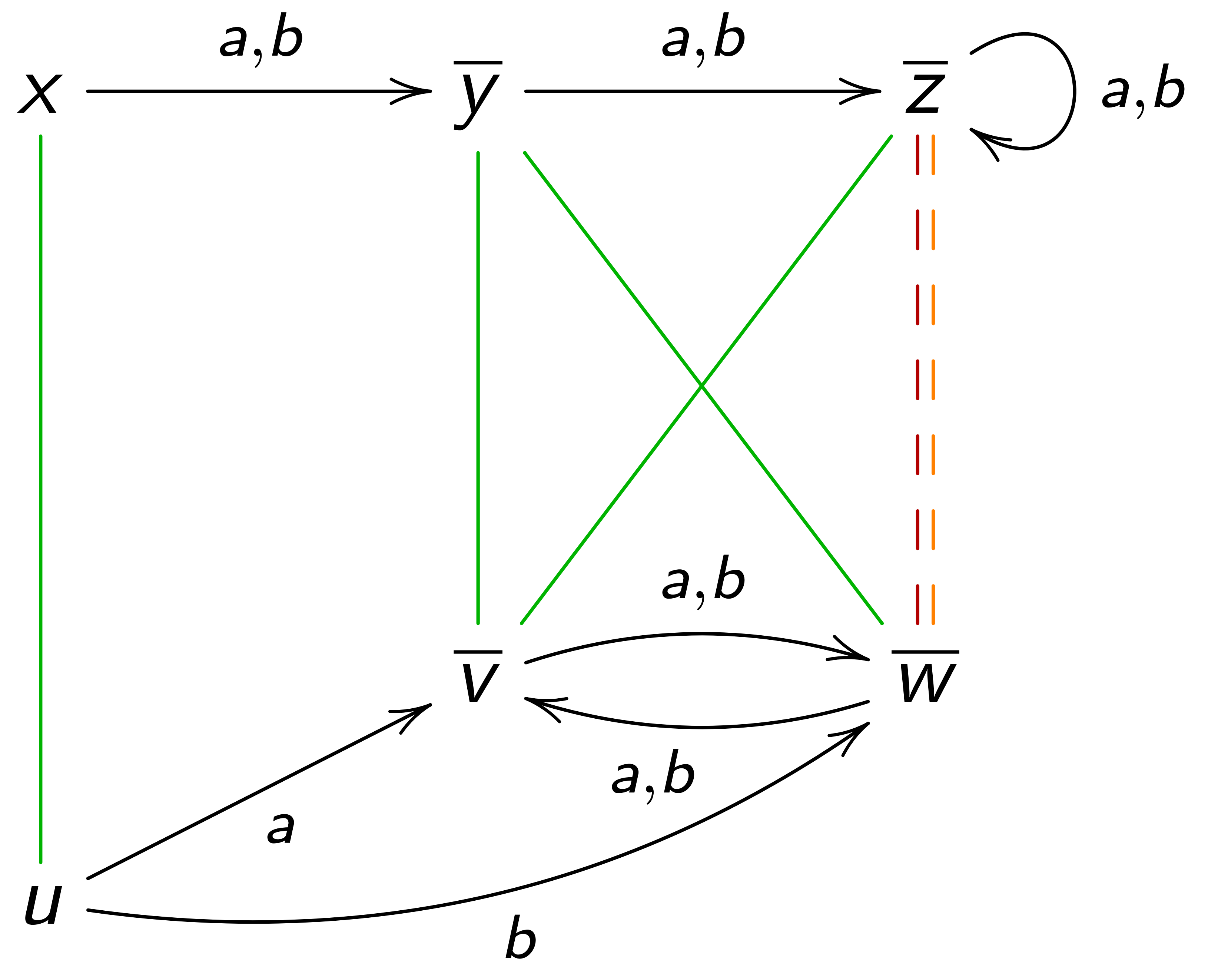
Deterministic finite automata

Last example, with two letters



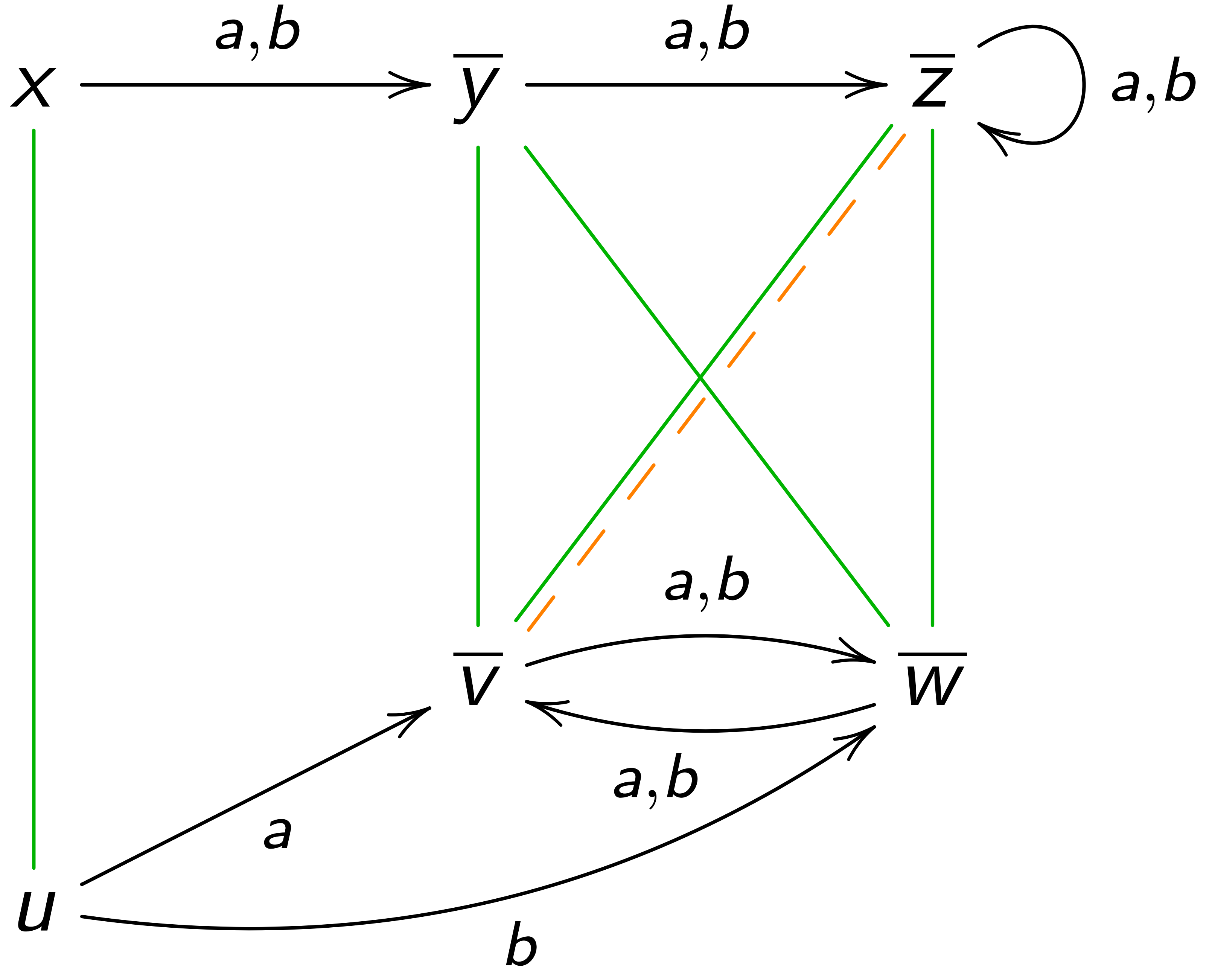
Deterministic finite automata

Last example, with two letters



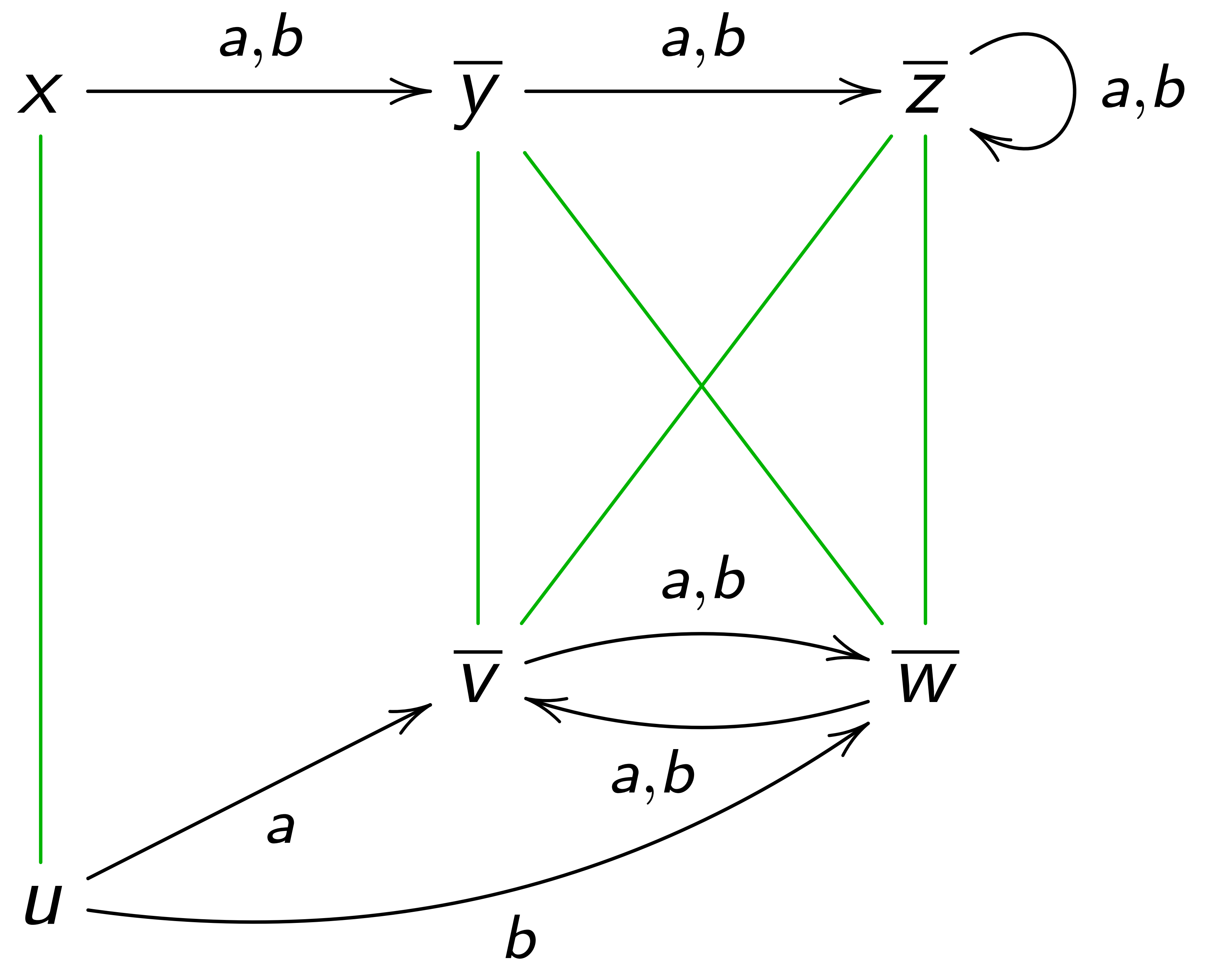
Deterministic finite automata

Last example, with two letters



Deterministic finite automata

Last example, with two letters



Correctness

- A relation R is a proof of equivalence (bisimulation) if $x \; R \; y$ entails
  - $o(x) = o(y)$;
  - for all a, $t_a(x) \; R \; t_a(y)$.

Correctness

- A relation R is a proof of equivalence (bisimulation) if $x\ R\ y$ entails
  - $o(x) = o(y)$;
  - for all a, $t_a(x)\ R\ t_a(y)$.
- *Theorem:* $L(x) = L(y)$ iff
  there exists a bisimulation R with $x\ R\ y$

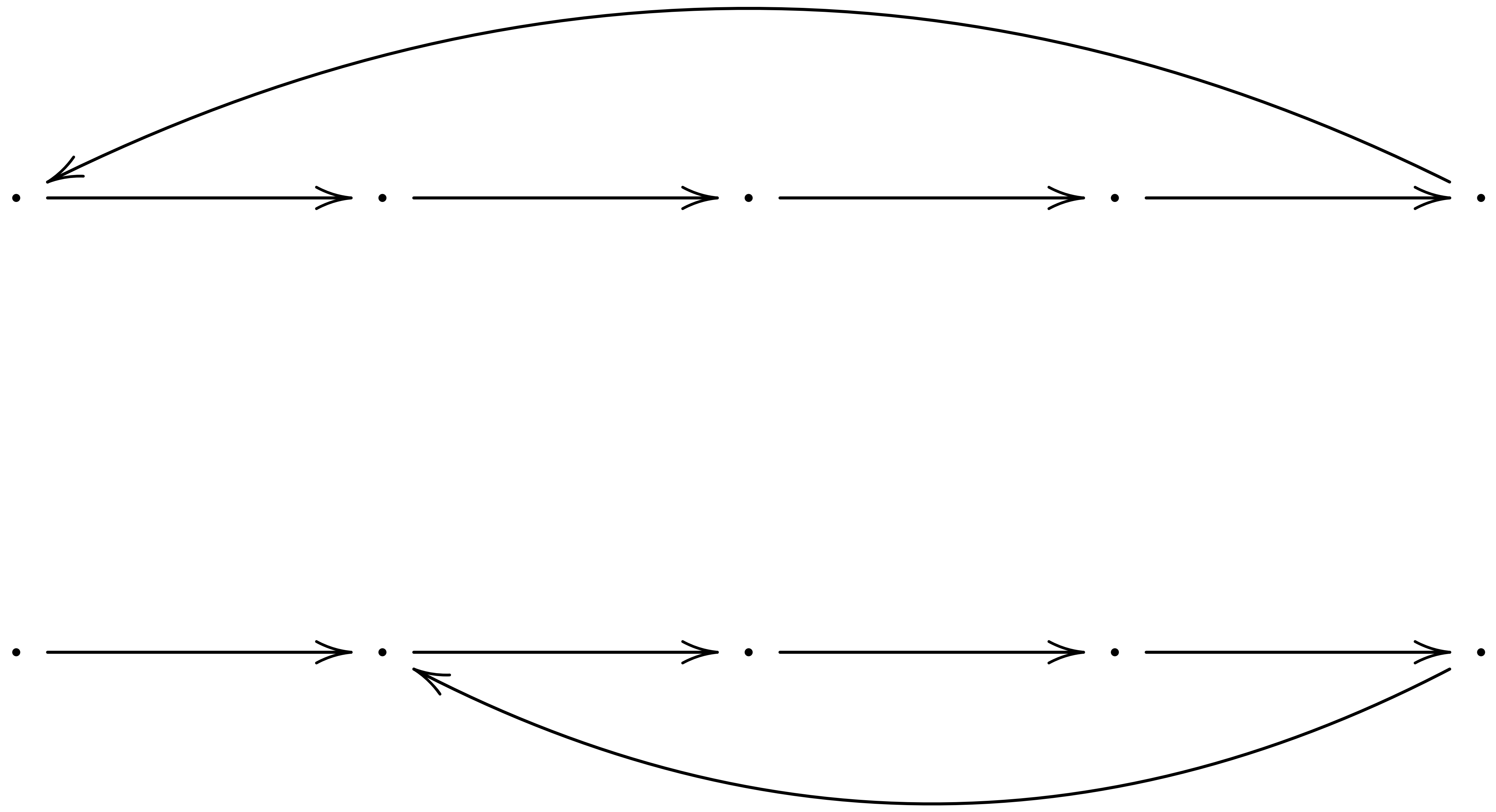
Correctness

- A relation R is a proof of equivalence (bisimulation) if $x \mathrel{R} y$ entails
  - $o(x) = o(y)$;
  - for all a, $t_a(x) \mathrel{R} t_a(y)$.
- *Theorem:* $L(x) = L(y)$ iff
  there exists a bisimulation R with $x \mathrel{R} y$

The previous algorithm attempts to construct a bisimulation

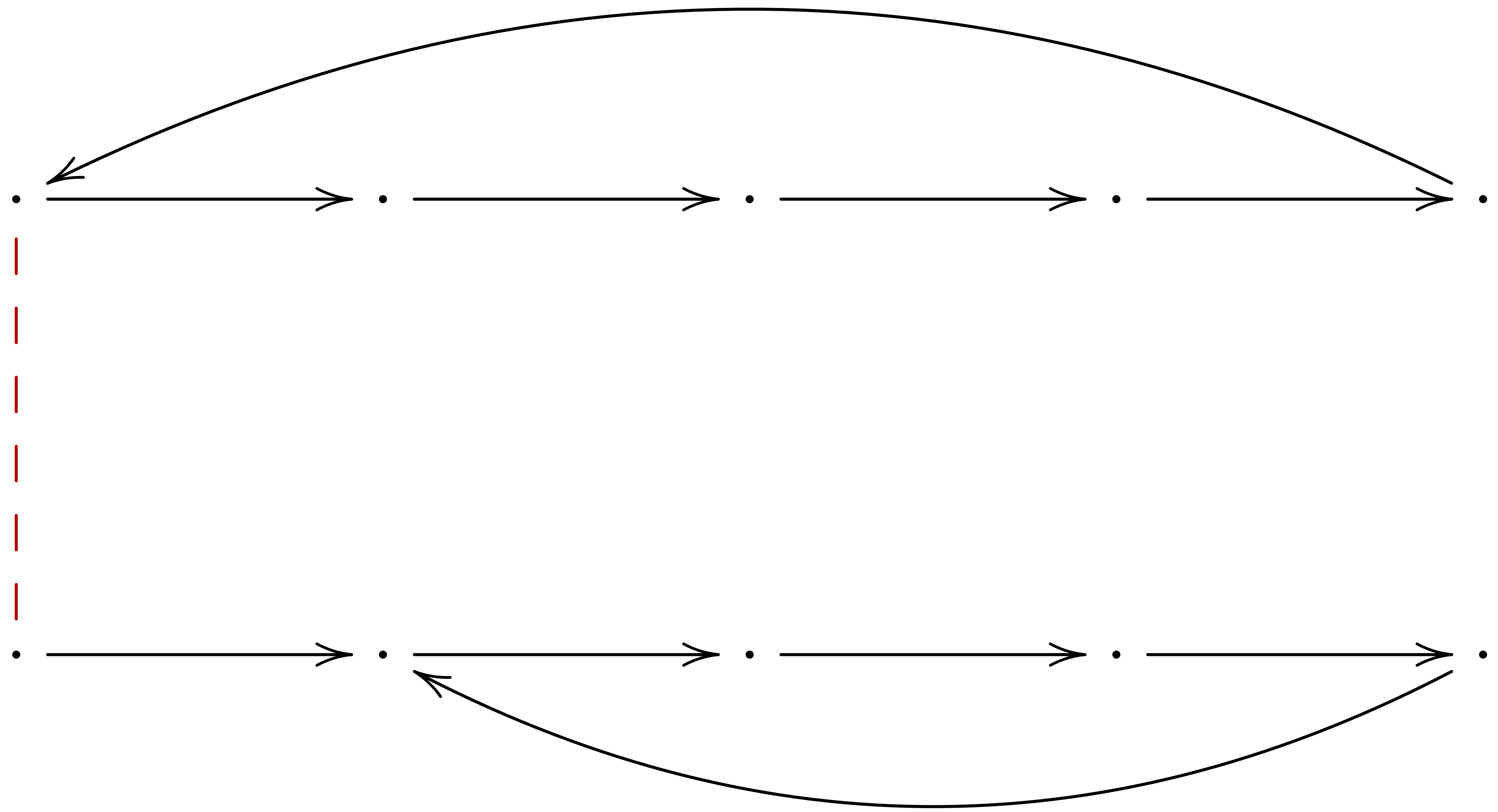
Complexity

The previous algorithm is quadratic



Complexity

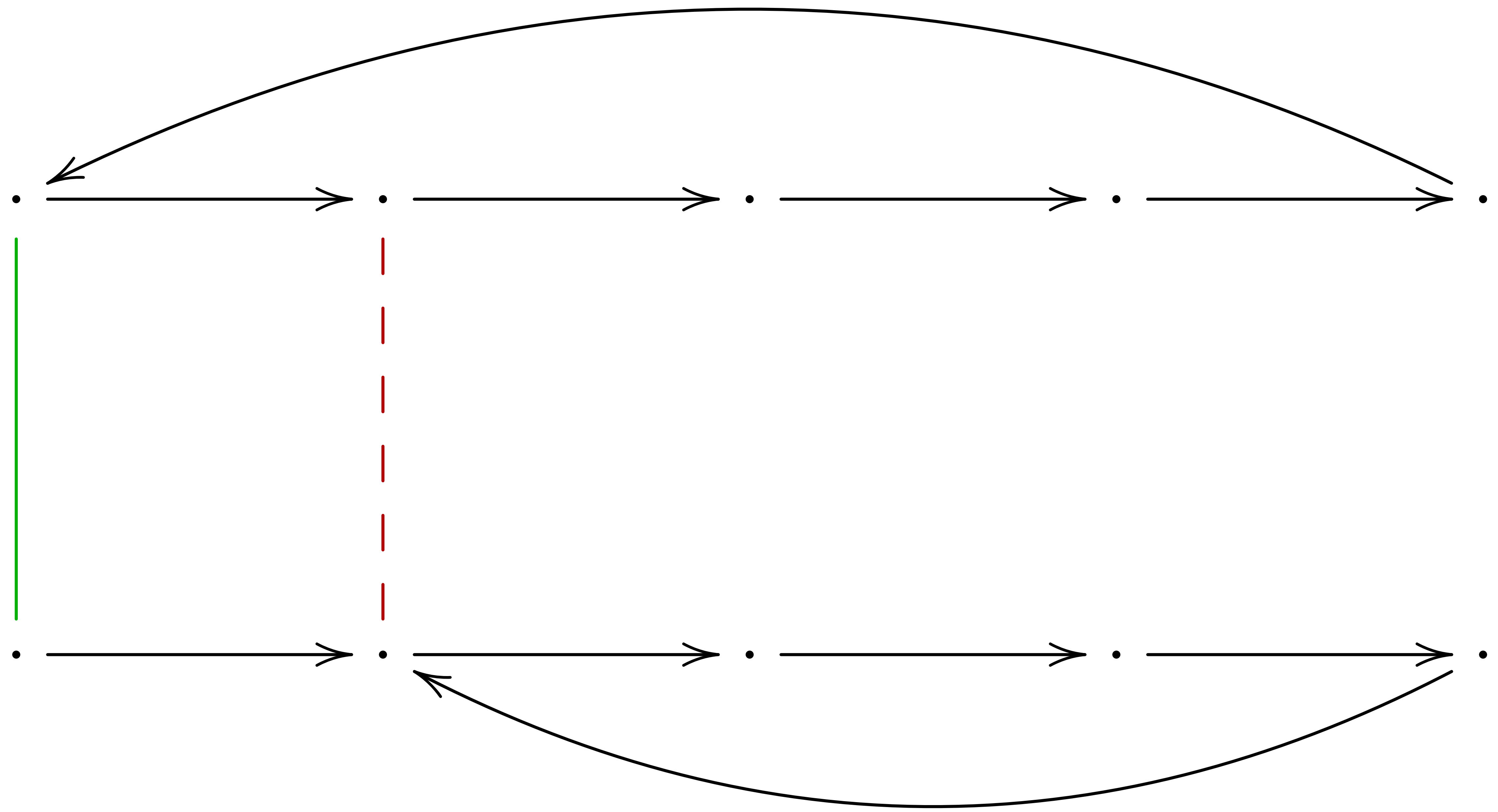
The previous algorithm is quadratic



0 pairs

Complexity

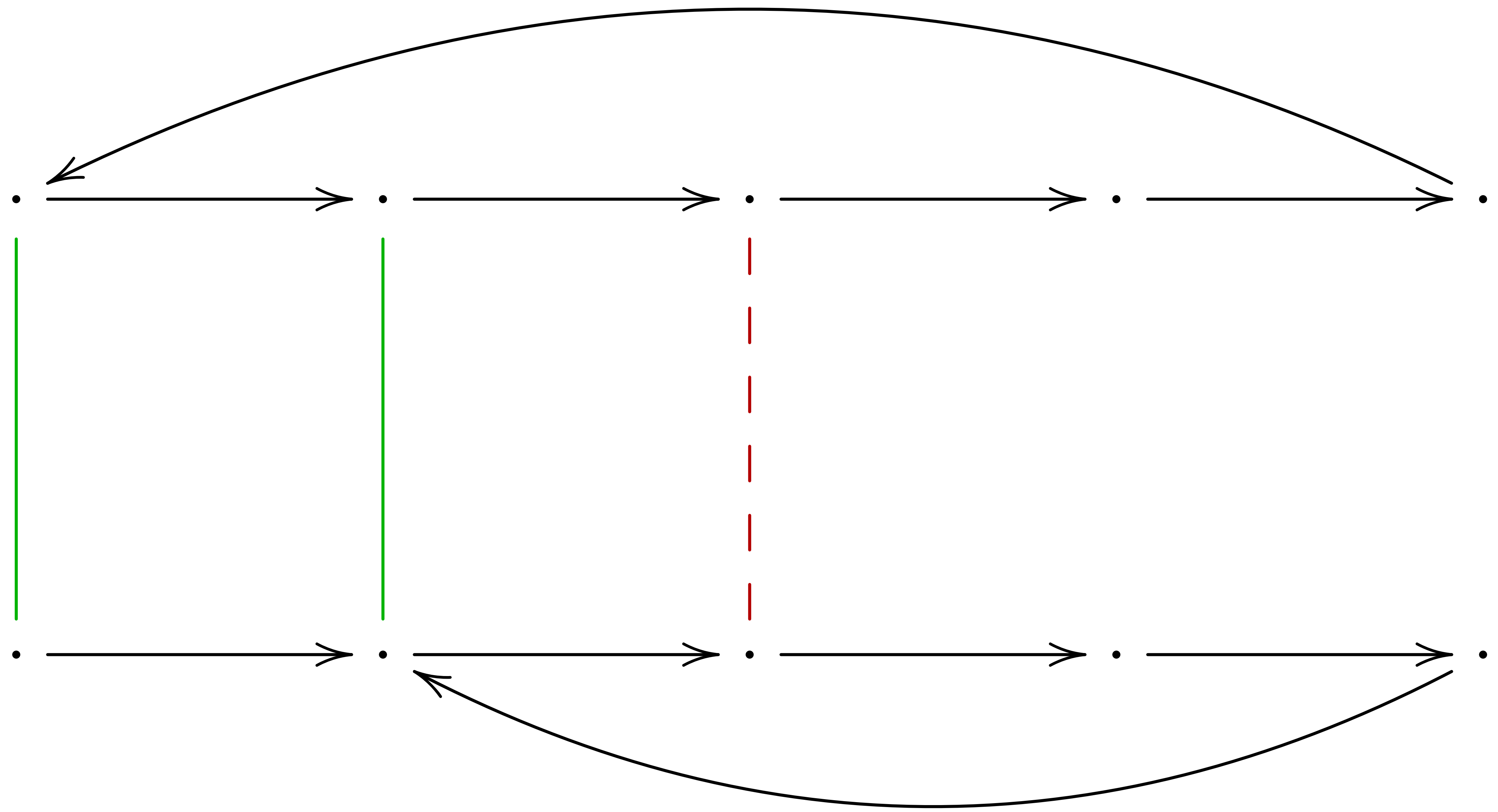
The previous algorithm is quadratic



1 pairs

Complexity

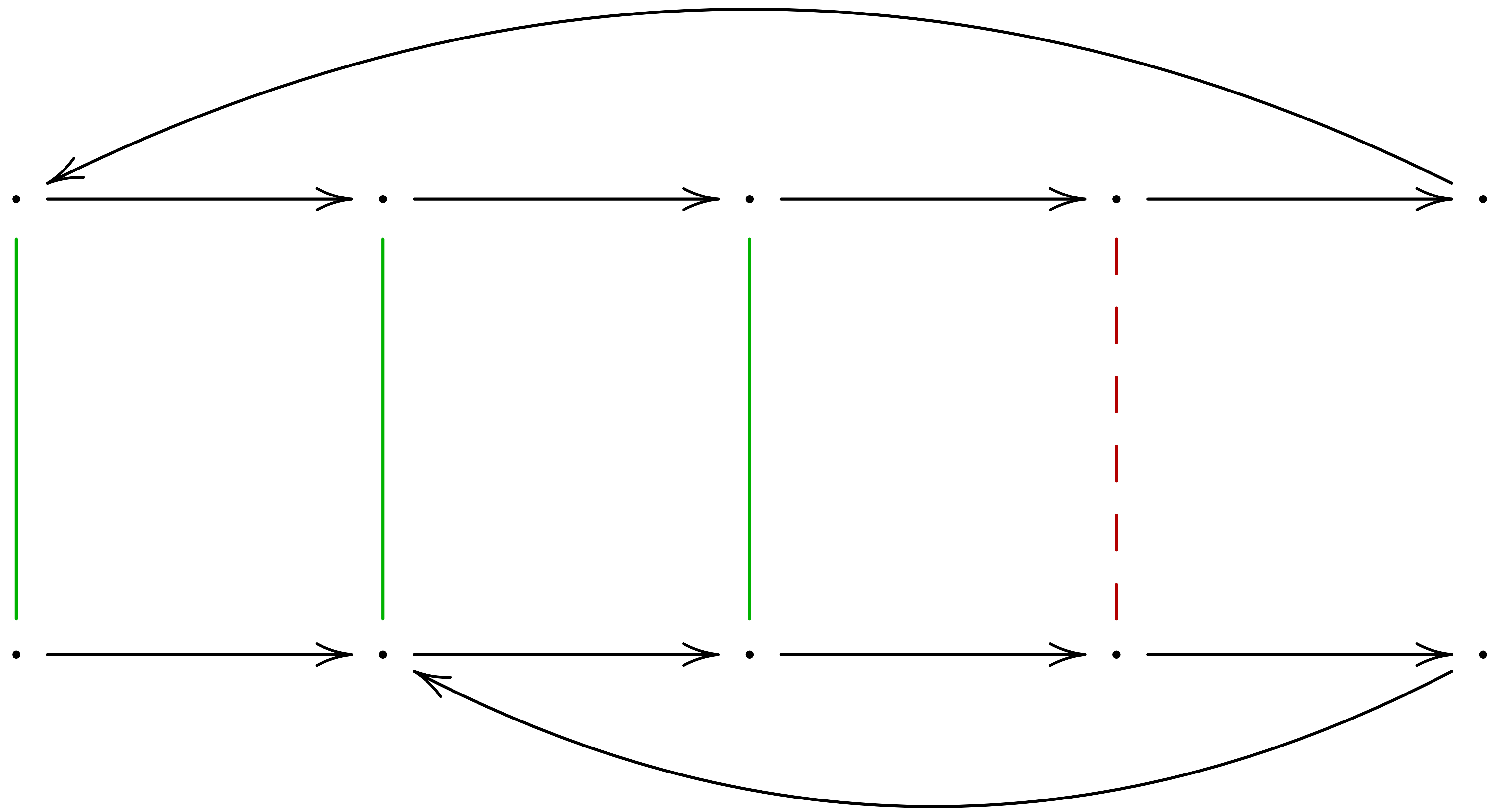
The previous algorithm is quadratic



2 pairs

Complexity

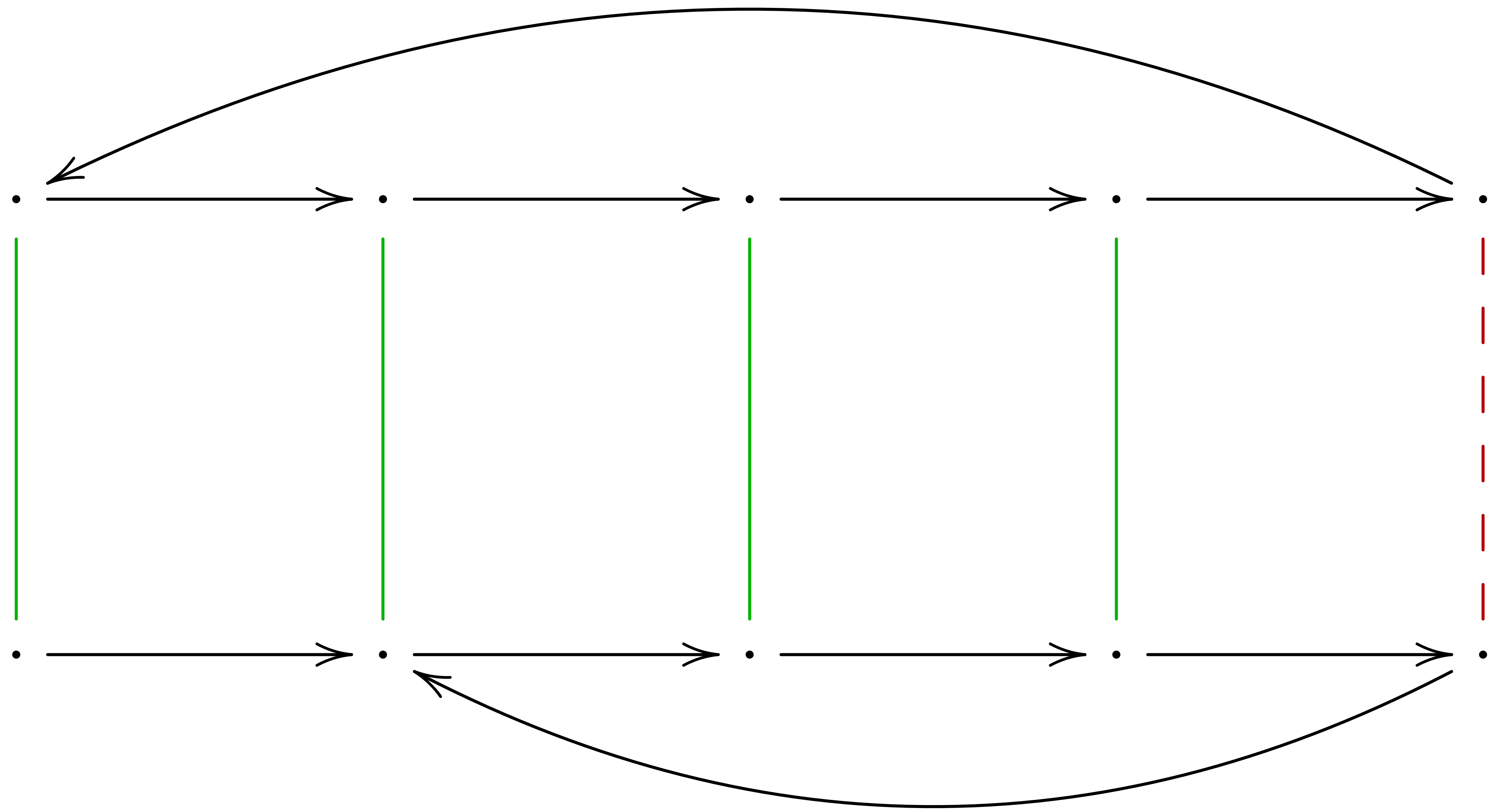
The previous algorithm is quadratic



3 pairs

Complexity

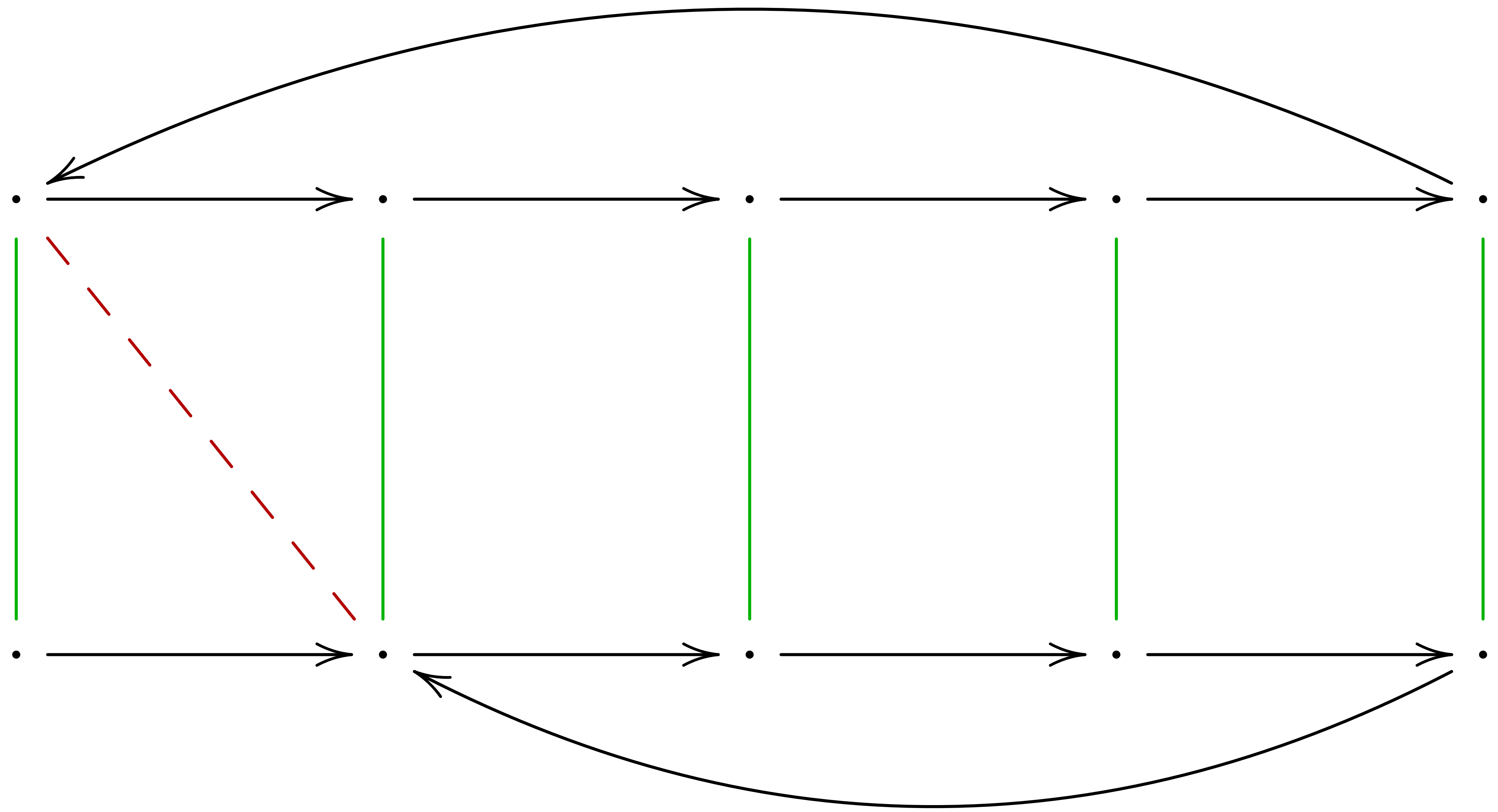
The previous algorithm is quadratic



4 pairs

Complexity

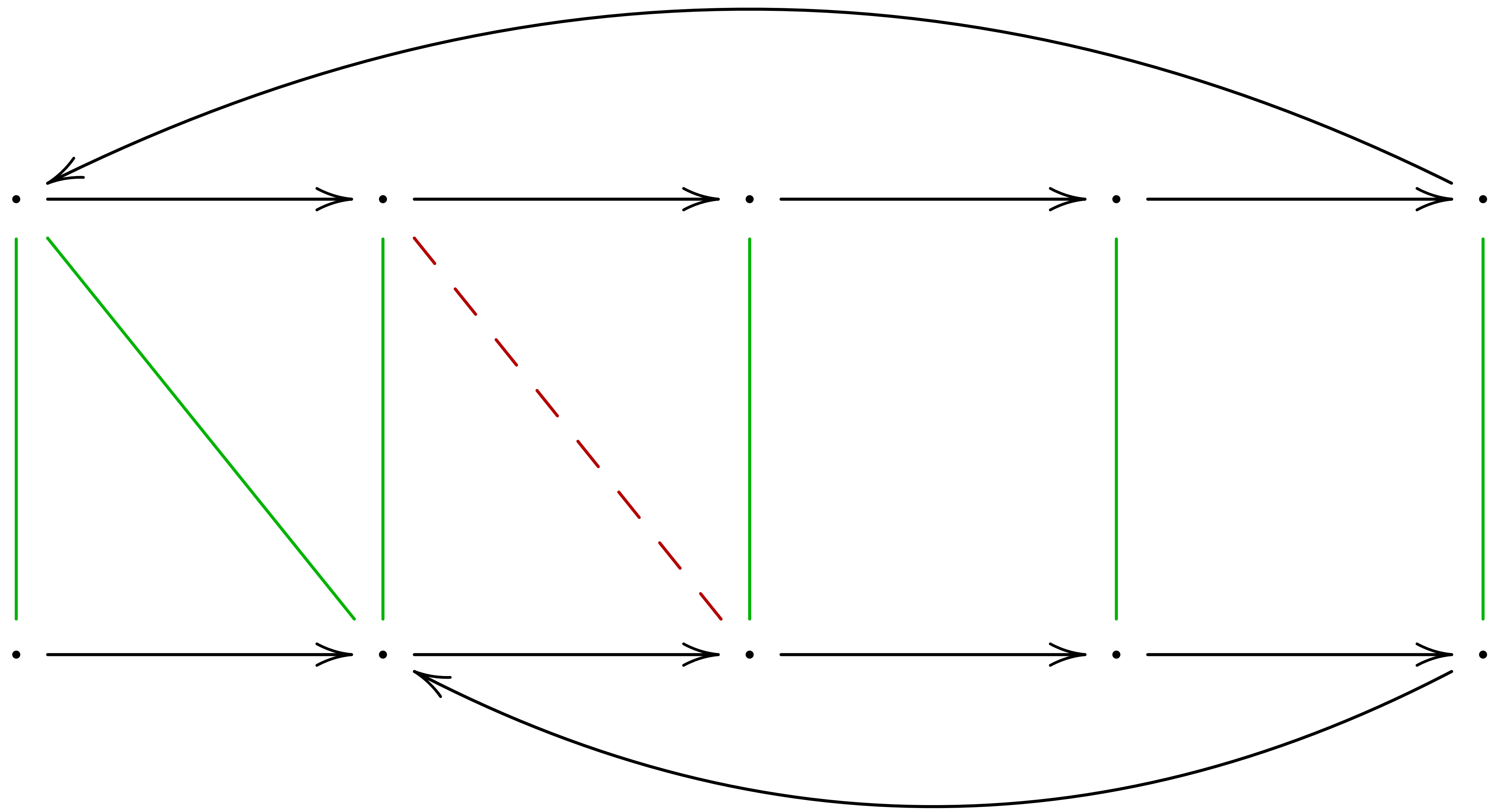
The previous algorithm is quadratic



5 pairs

Complexity

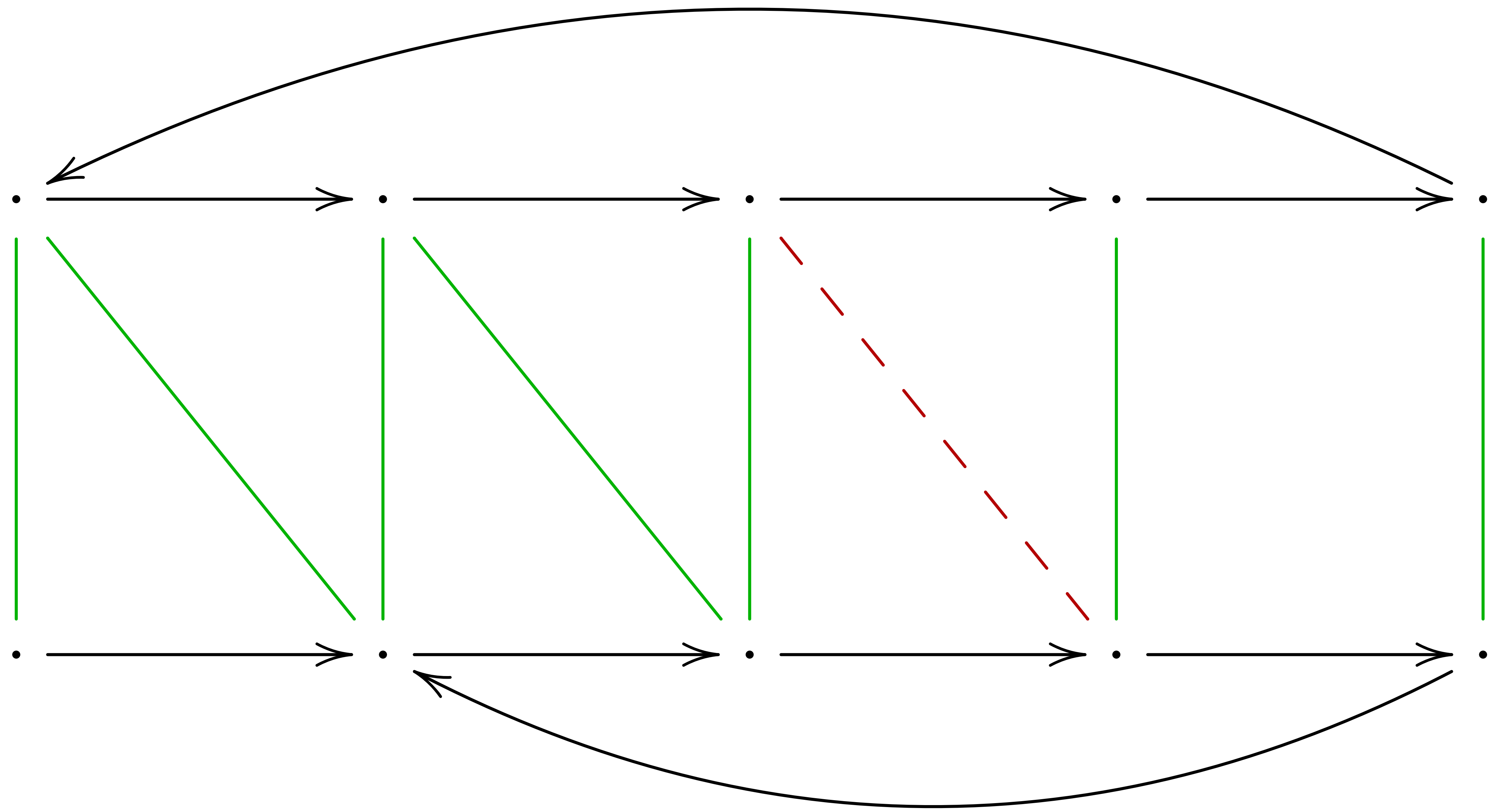
The previous algorithm is quadratic



6 pairs

Complexity

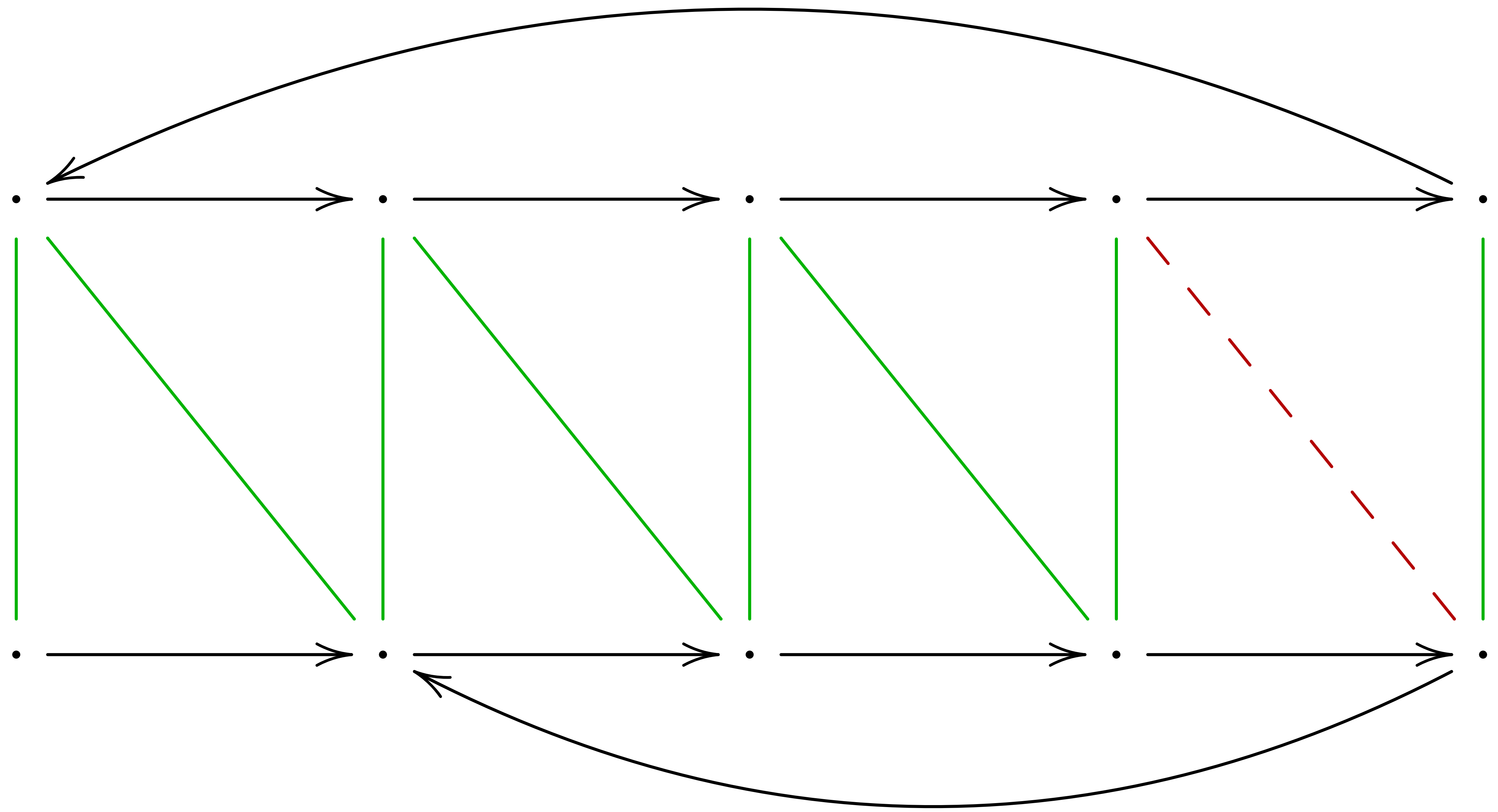
The previous algorithm is quadratic



7 pairs

Complexity

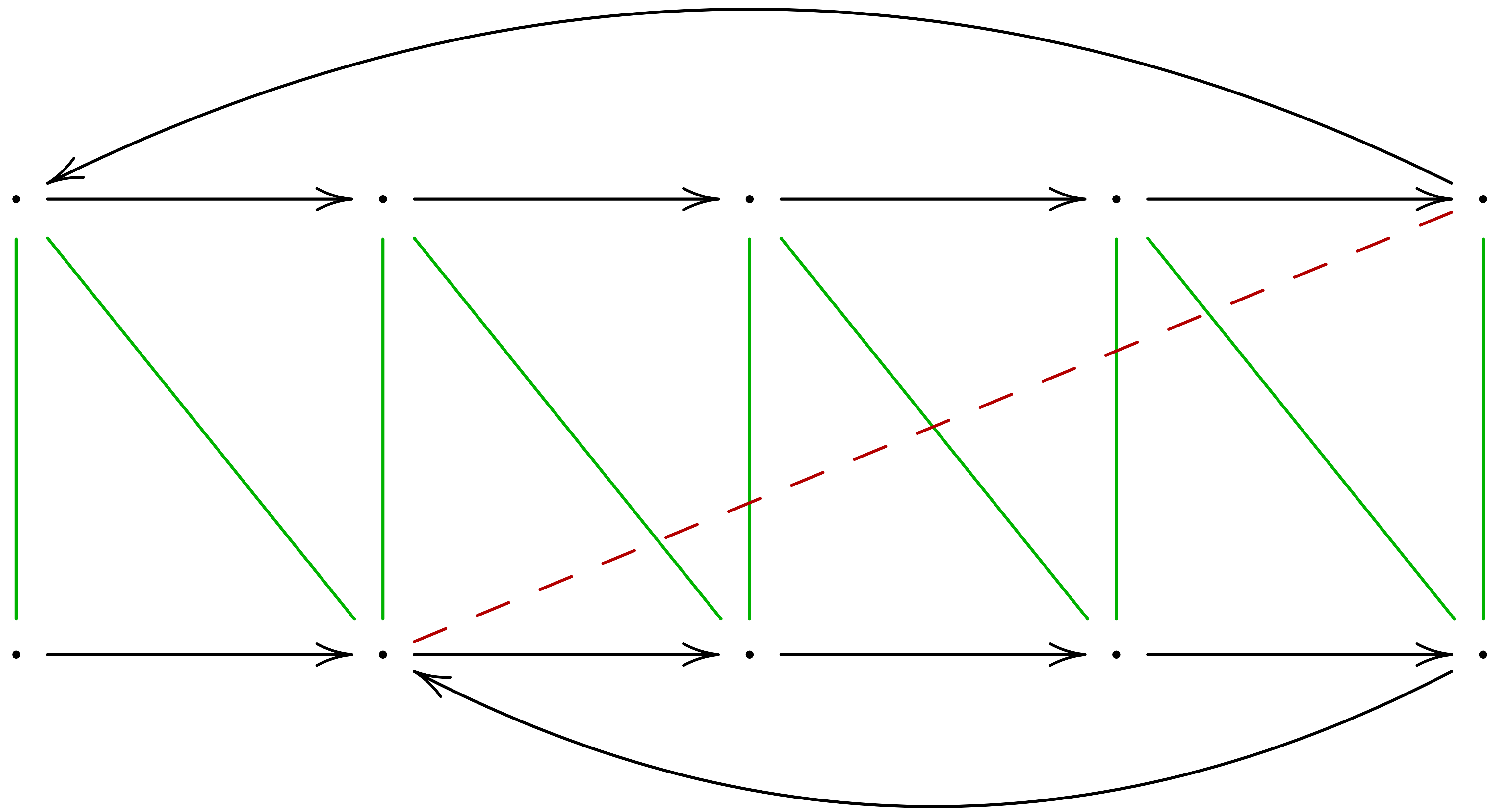
The previous algorithm is quadratic



8 pairs

Complexity

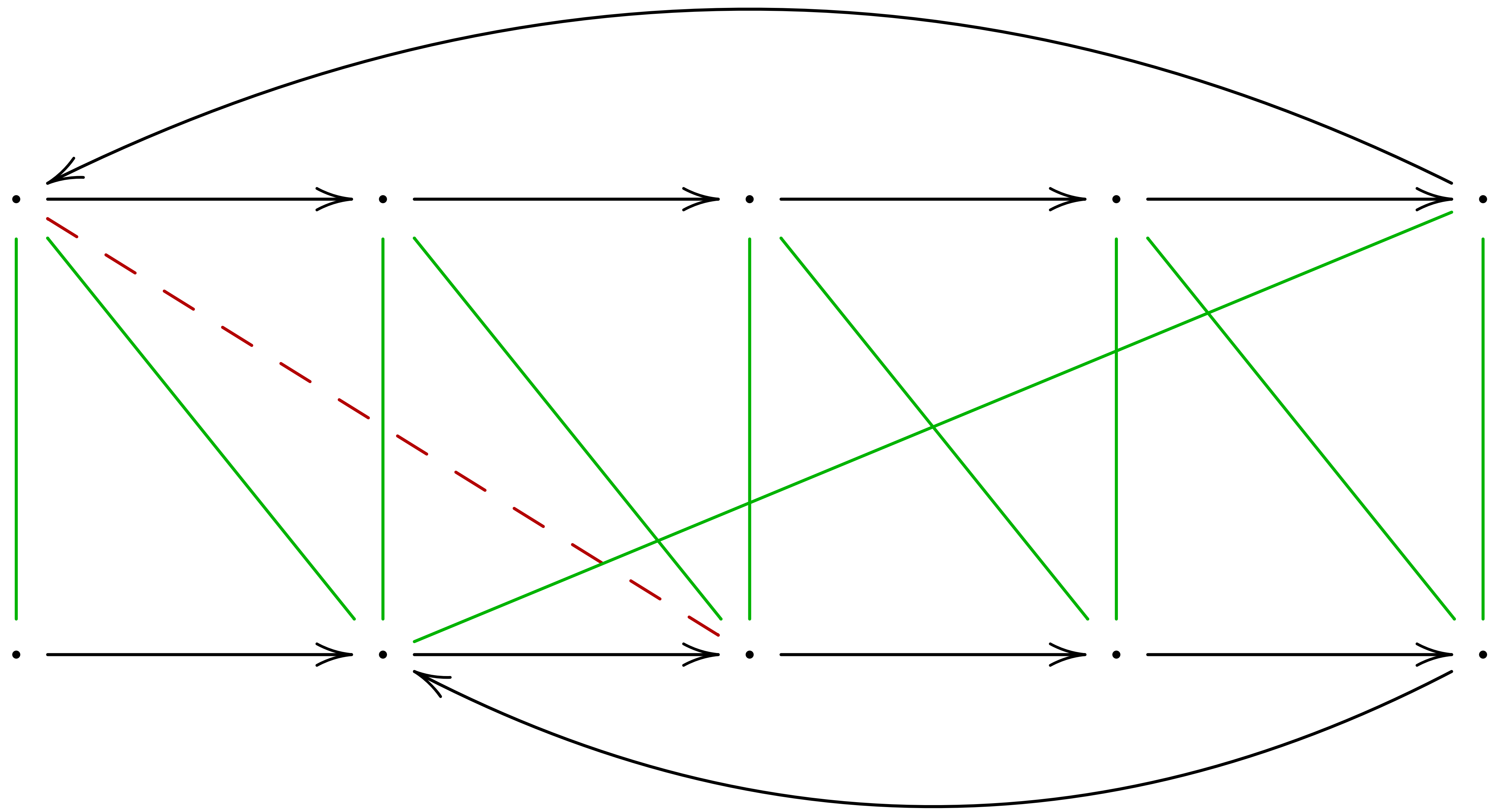
The previous algorithm is quadratic



9 pairs

Complexity

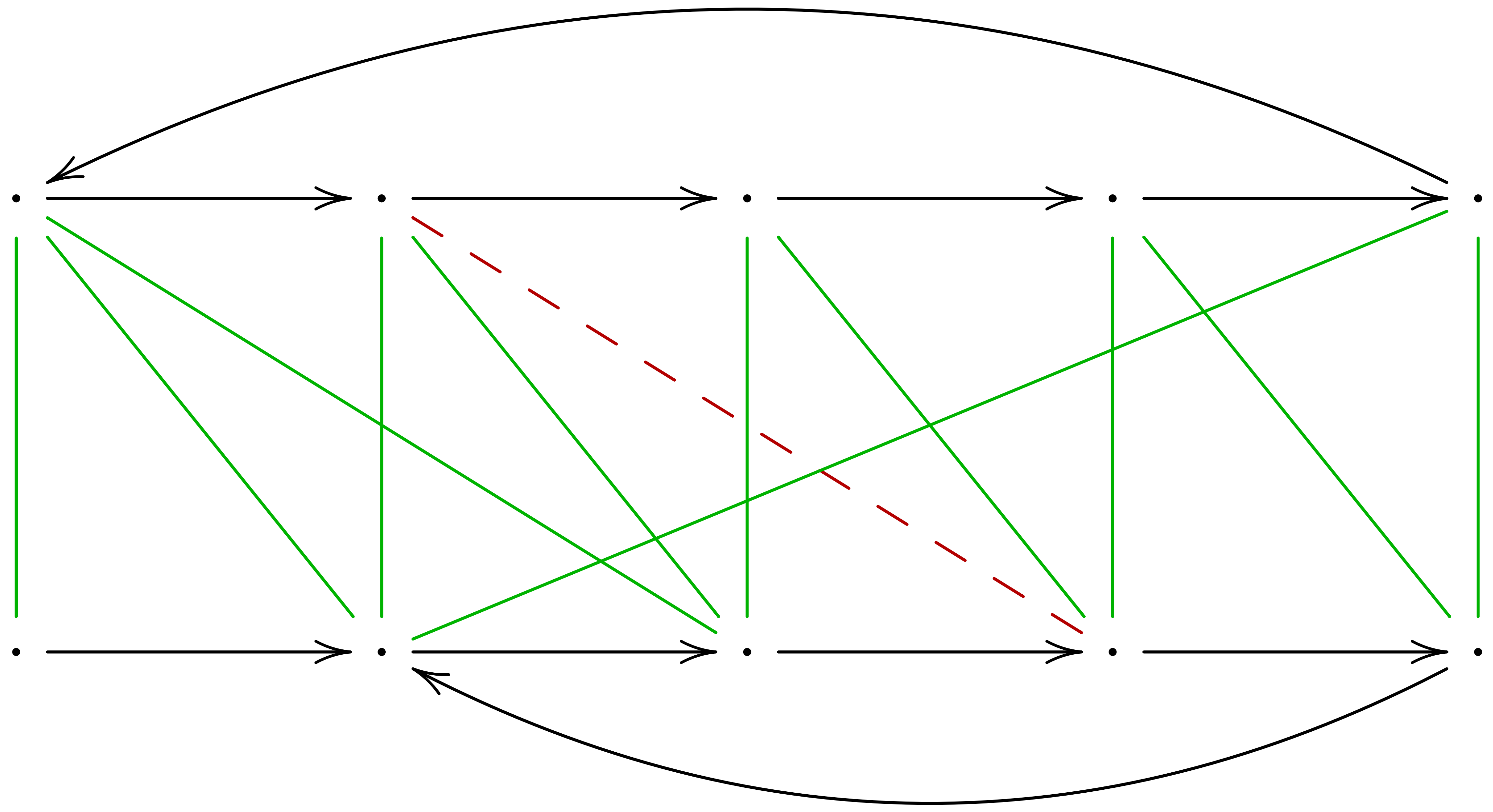
The previous algorithm is quadratic



10 pairs

Complexity

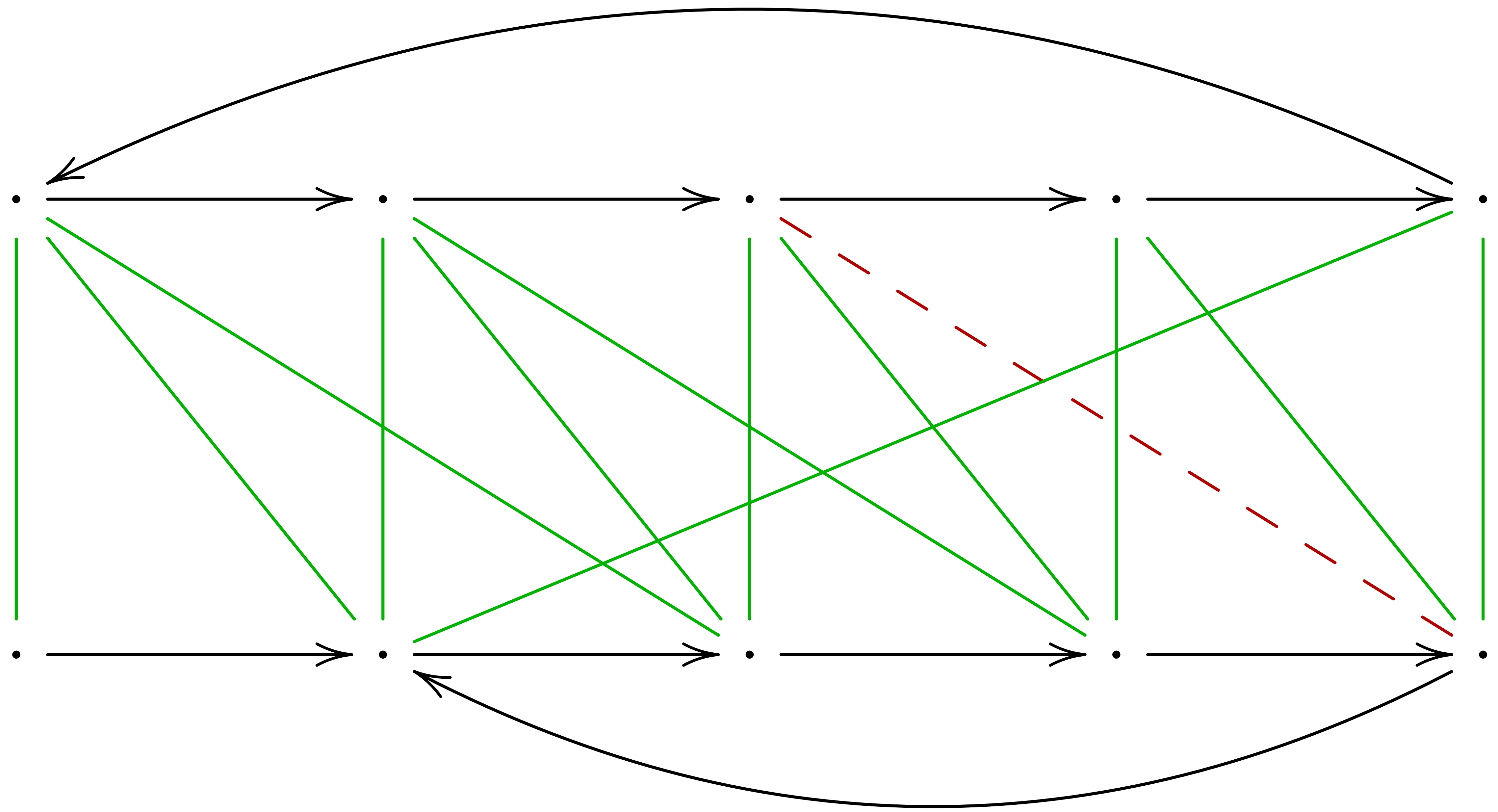
The previous algorithm is quadratic



11 pairs

Complexity

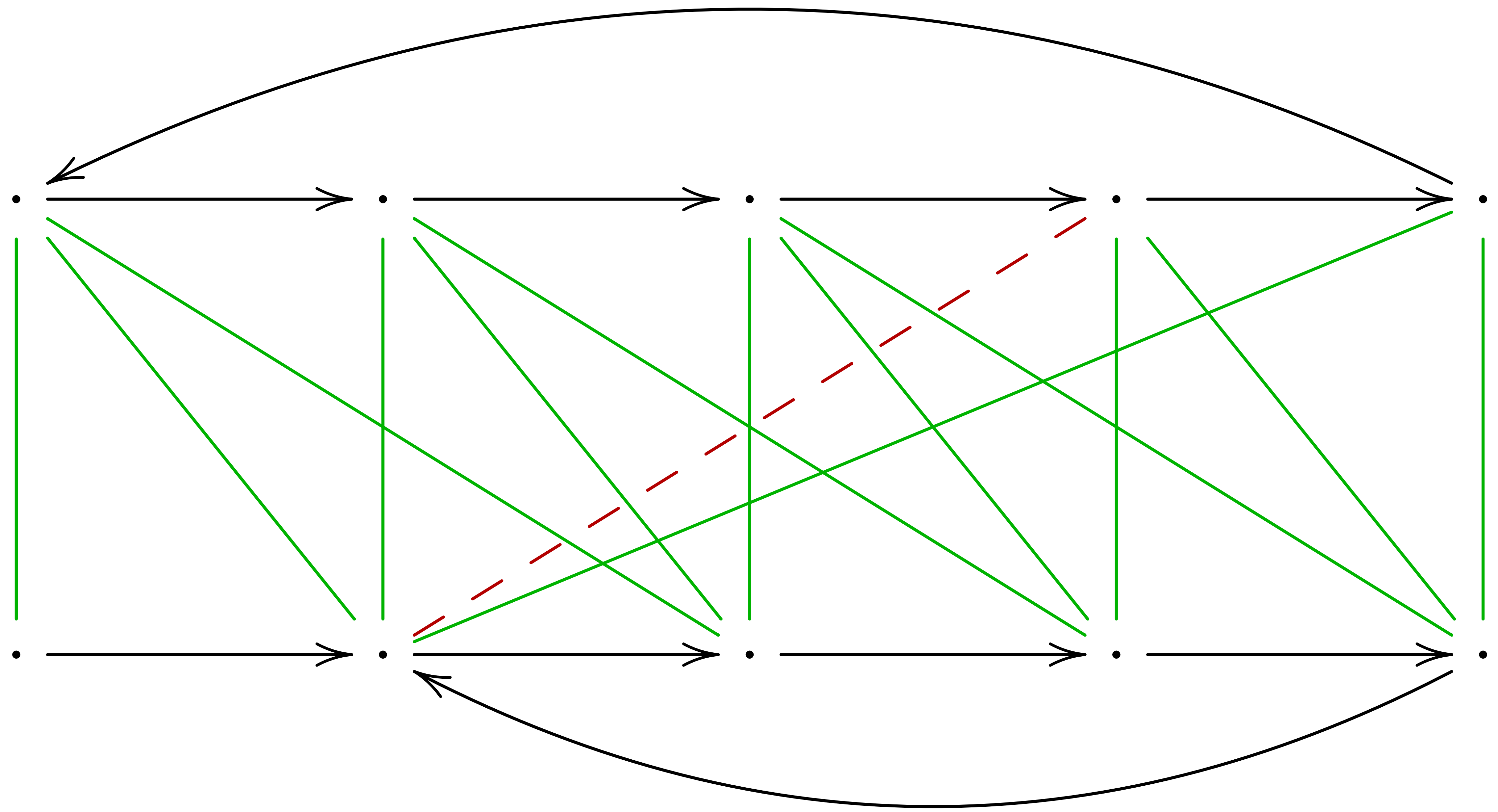
The previous algorithm is quadratic



12 pairs

Complexity

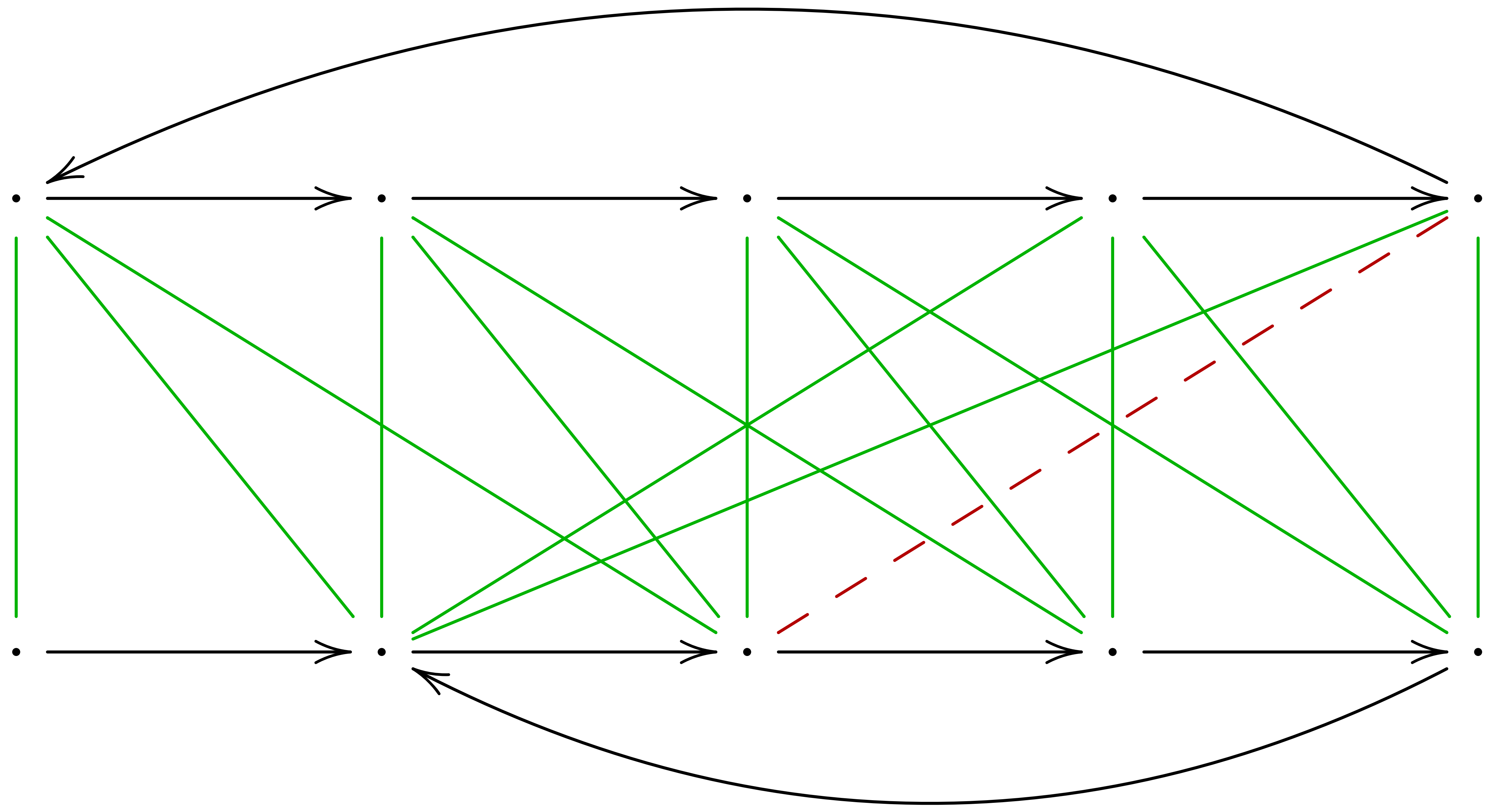
The previous algorithm is quadratic



13 pairs

Complexity

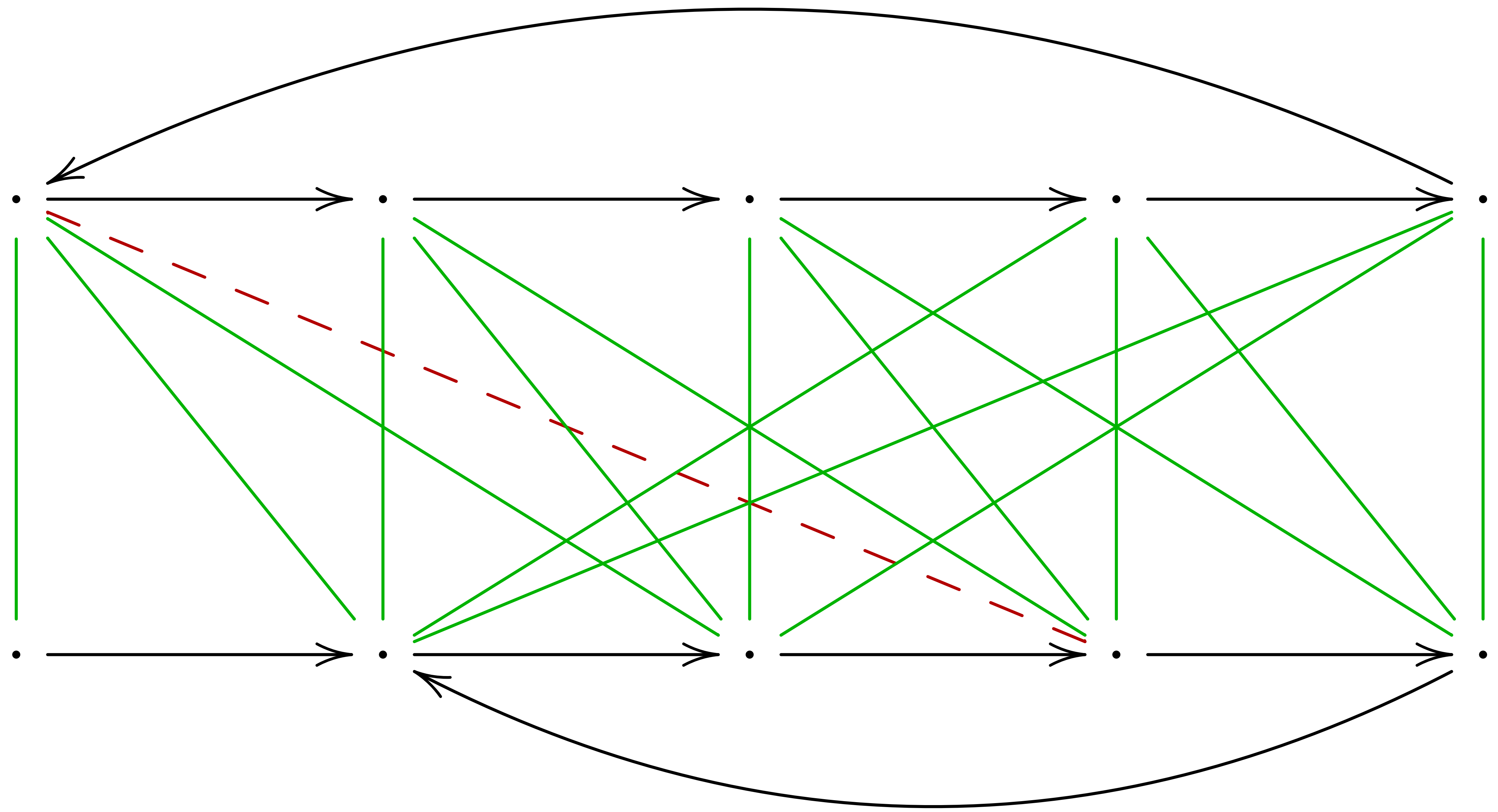
The previous algorithm is quadratic



14 pairs

Complexity

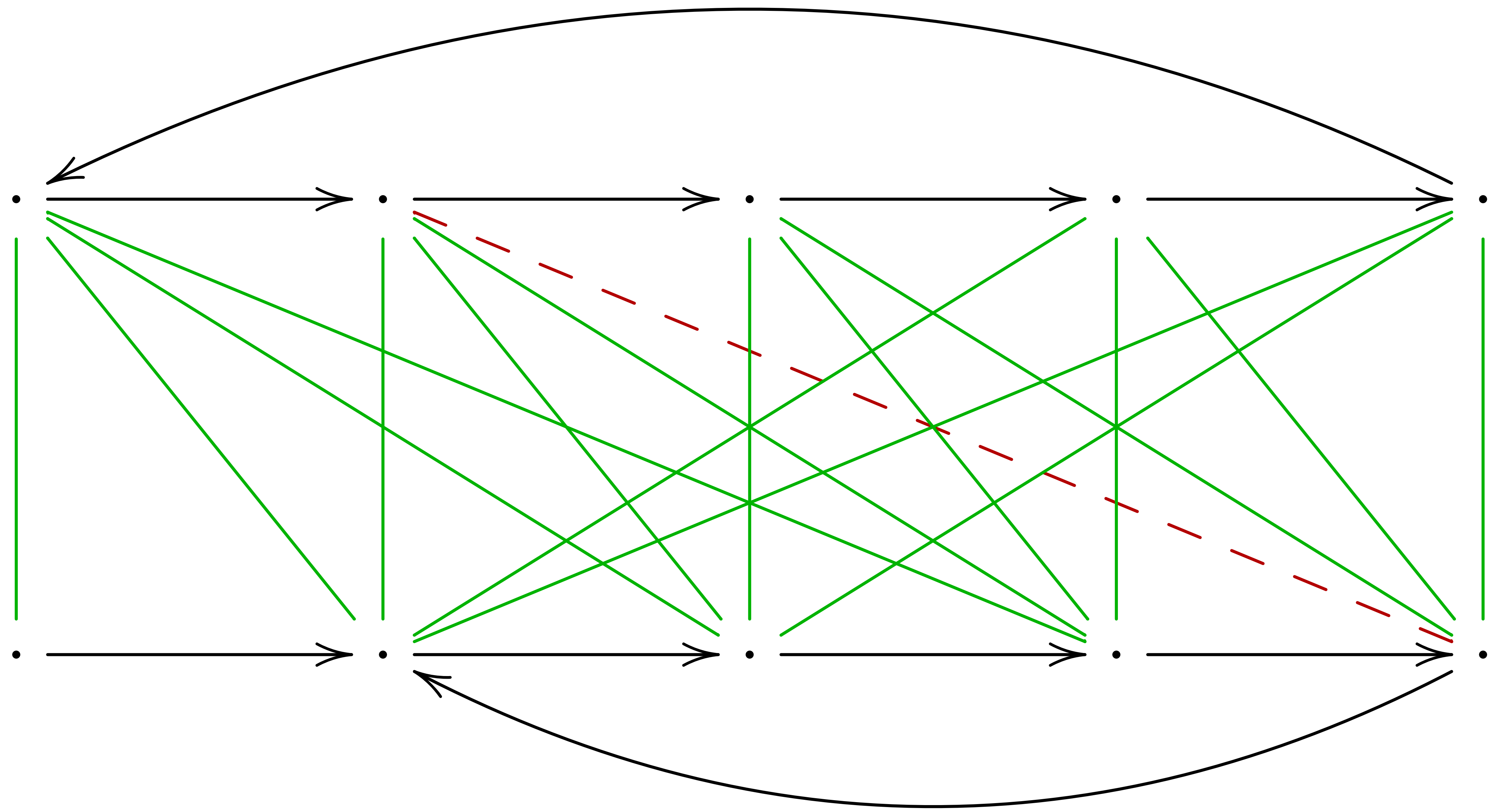
The previous algorithm is quadratic



15 pairs

Complexity

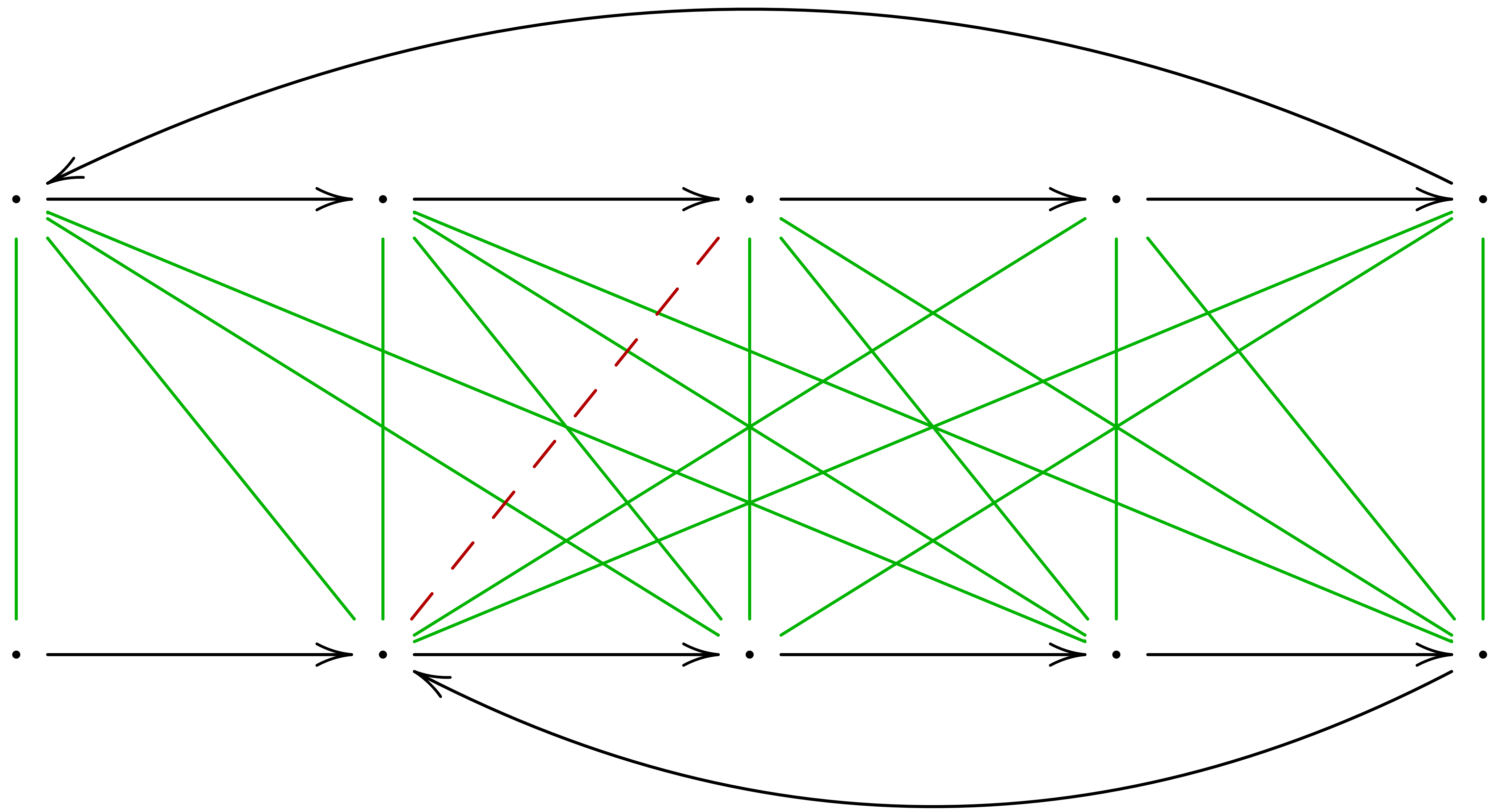
The previous algorithm is quadratic



16 pairs

Complexity

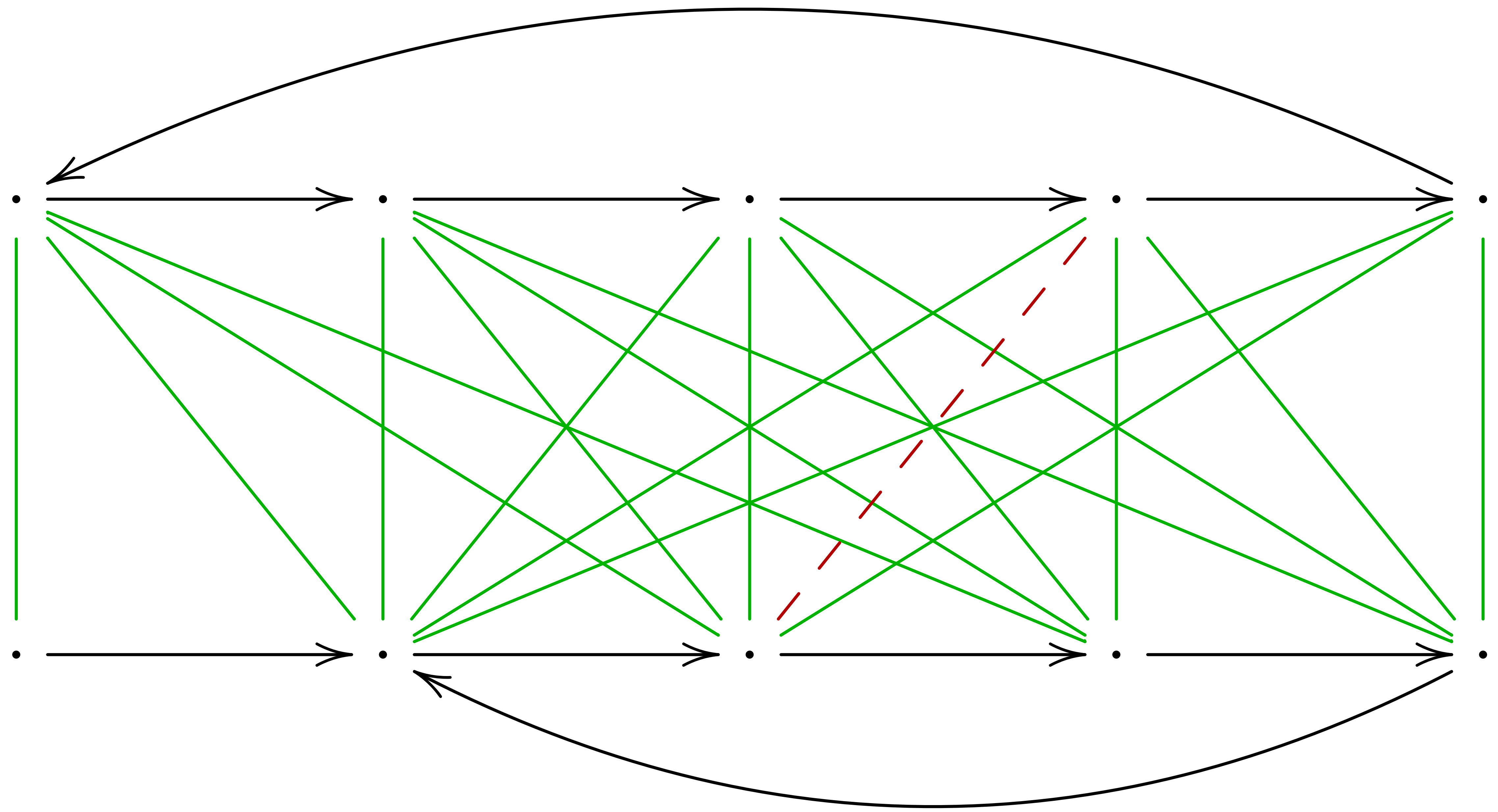
The previous algorithm is quadratic



17 pairs

Complexity

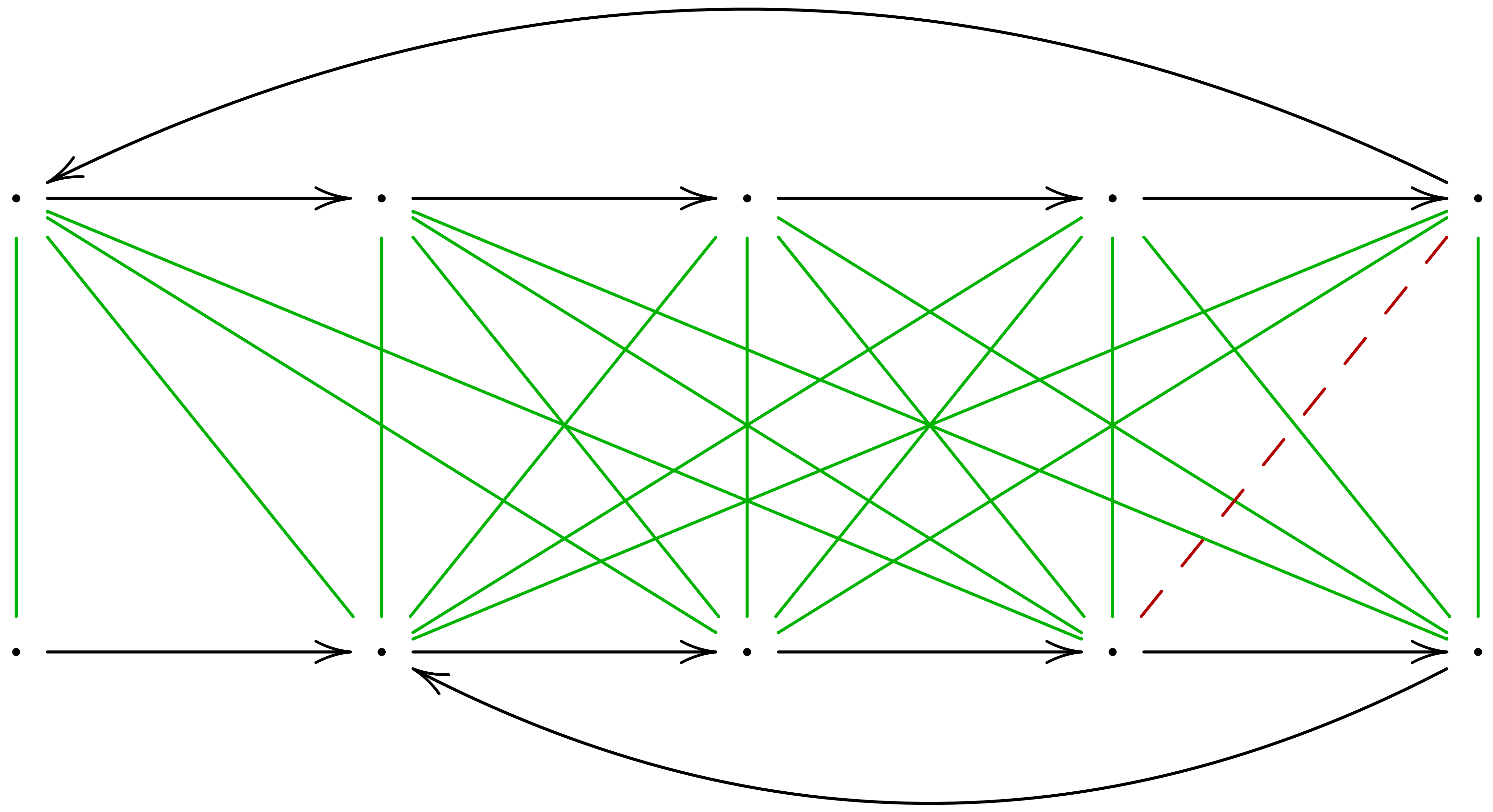
The previous algorithm is quadratic



18 pairs

Complexity

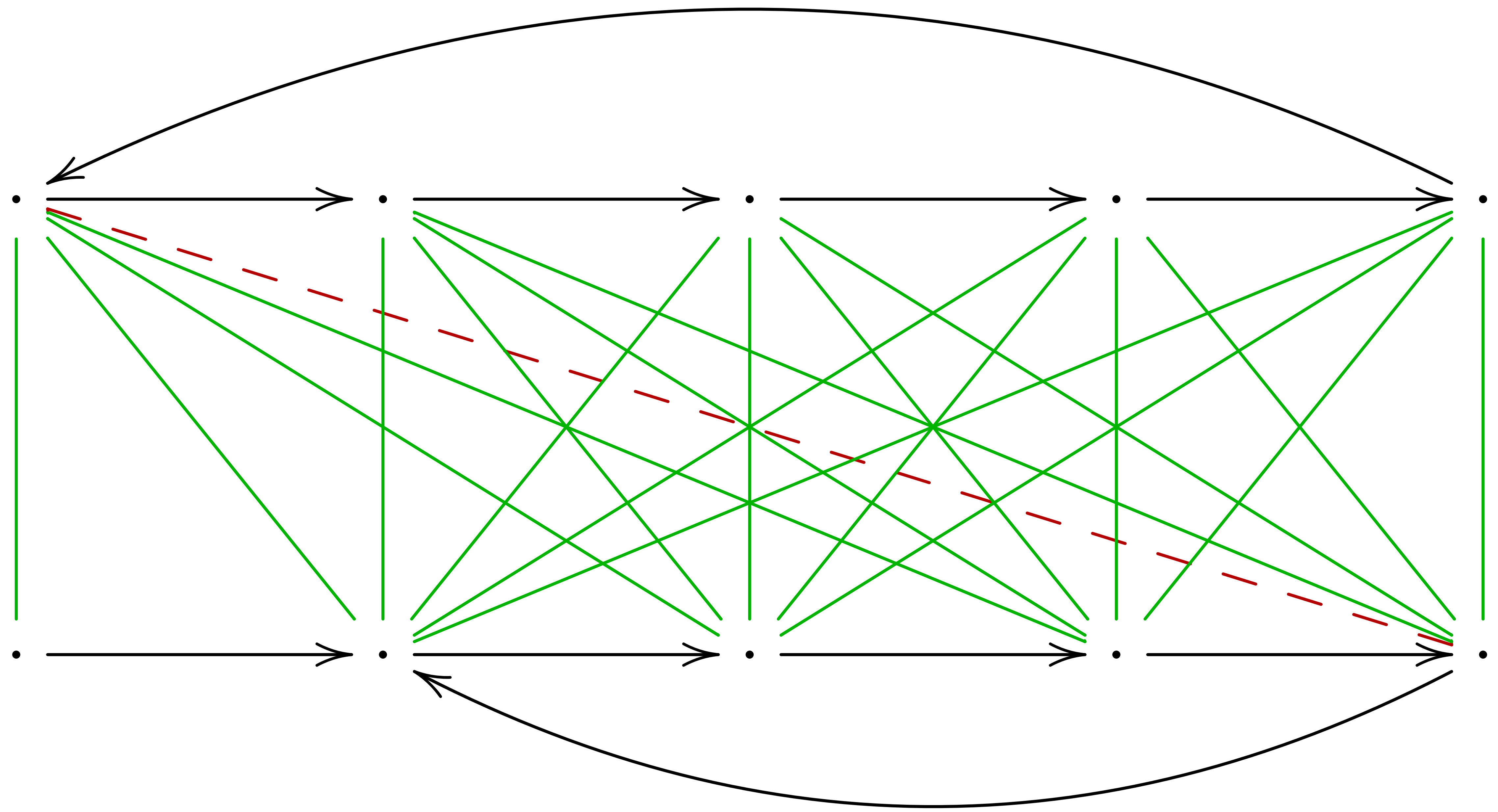
The previous algorithm is quadratic



19 pairs

Complexity

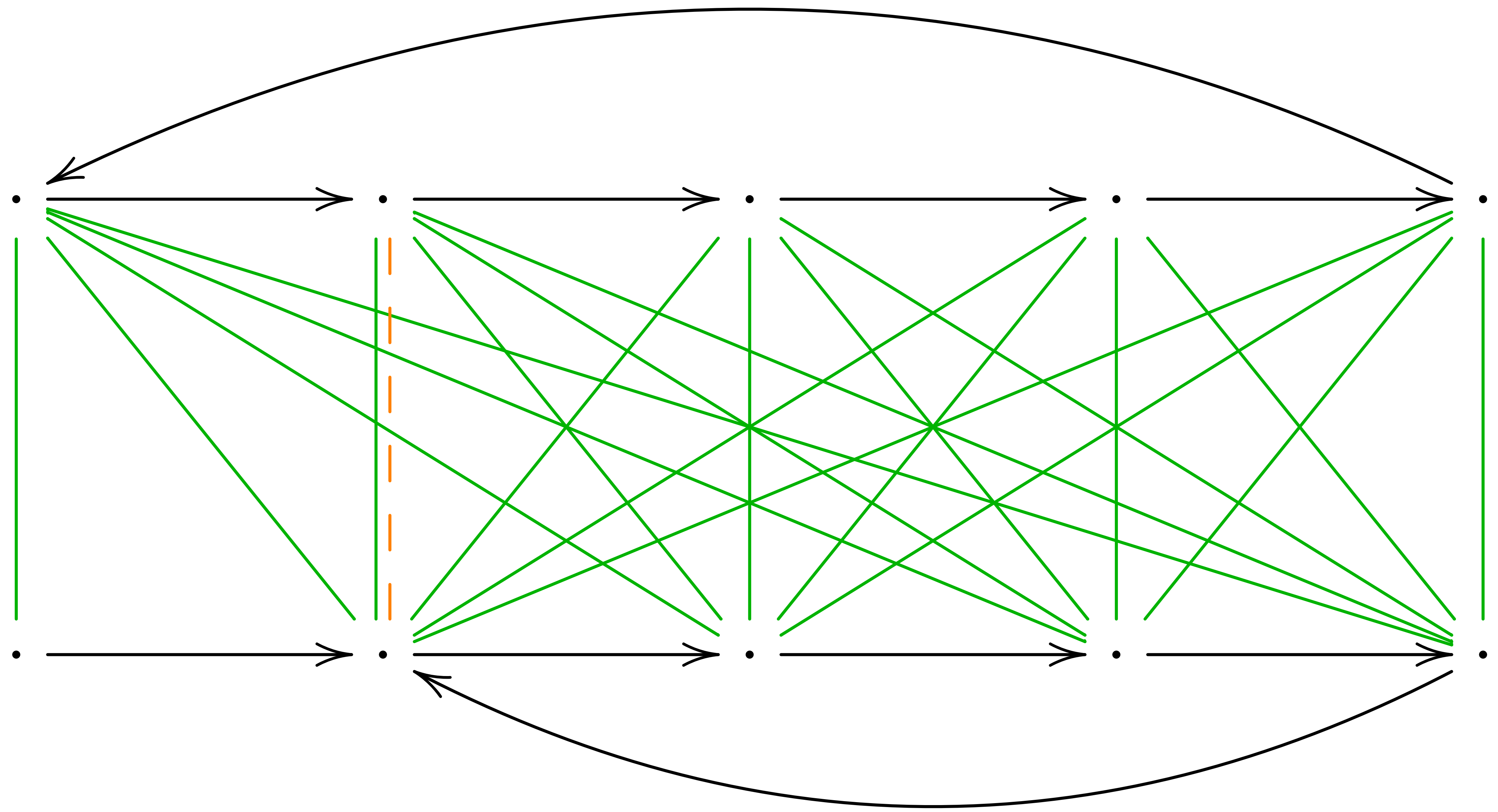
The previous algorithm is quadratic



20 pairs

Complexity

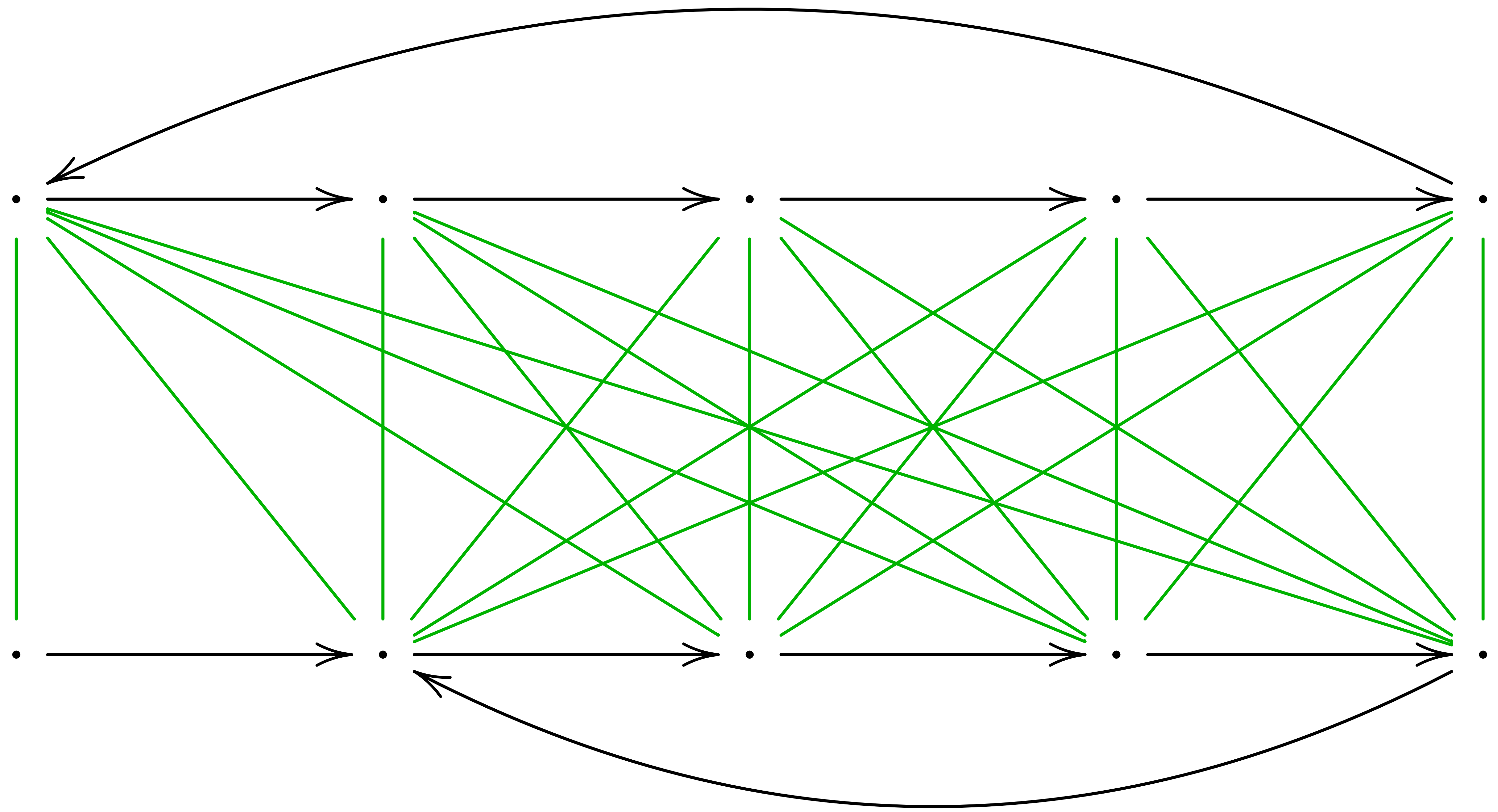
The previous algorithm is quadratic



21 pairs

Complexity

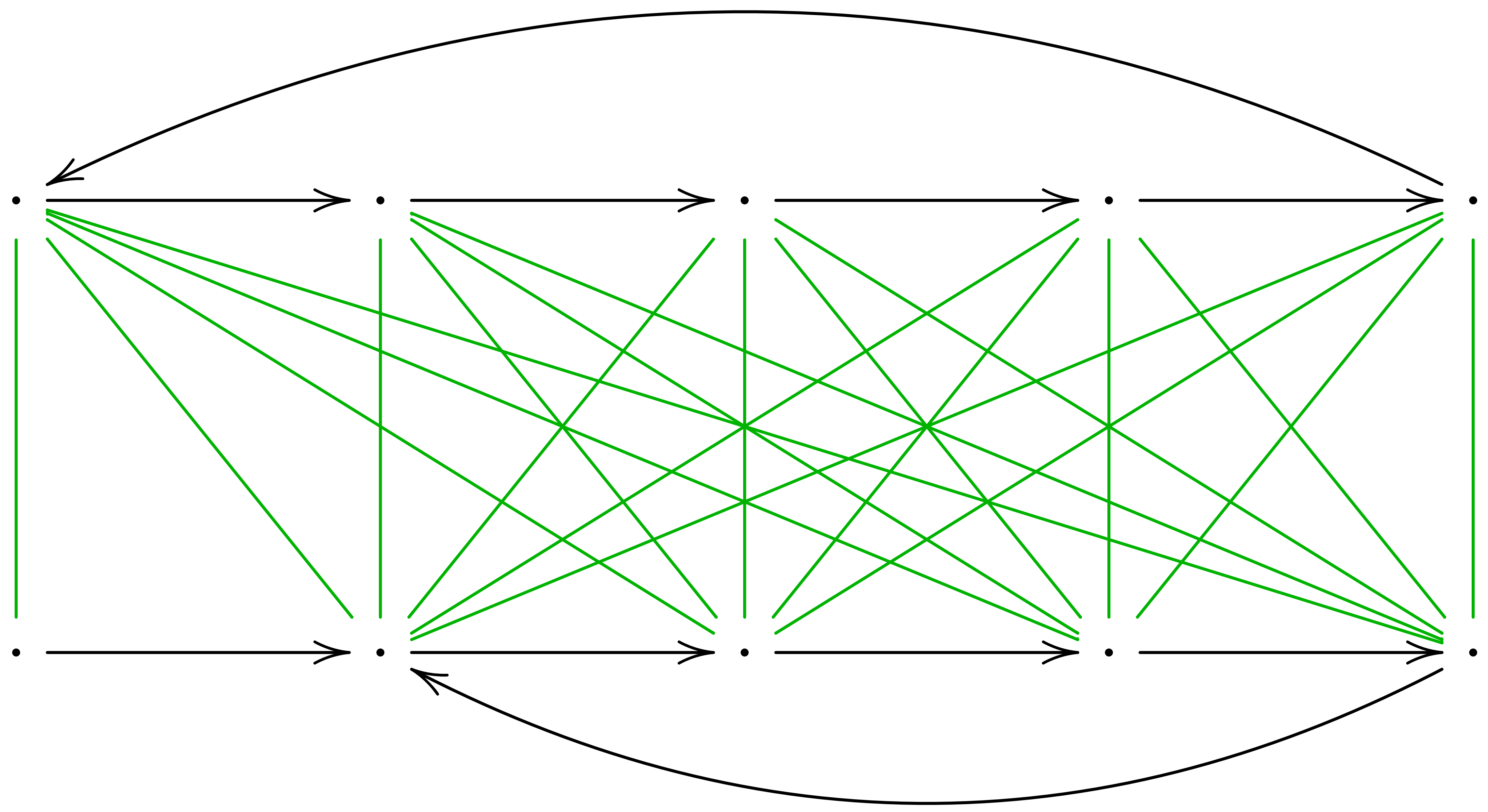
The previous algorithm is quadratic



21 pairs

First improvement

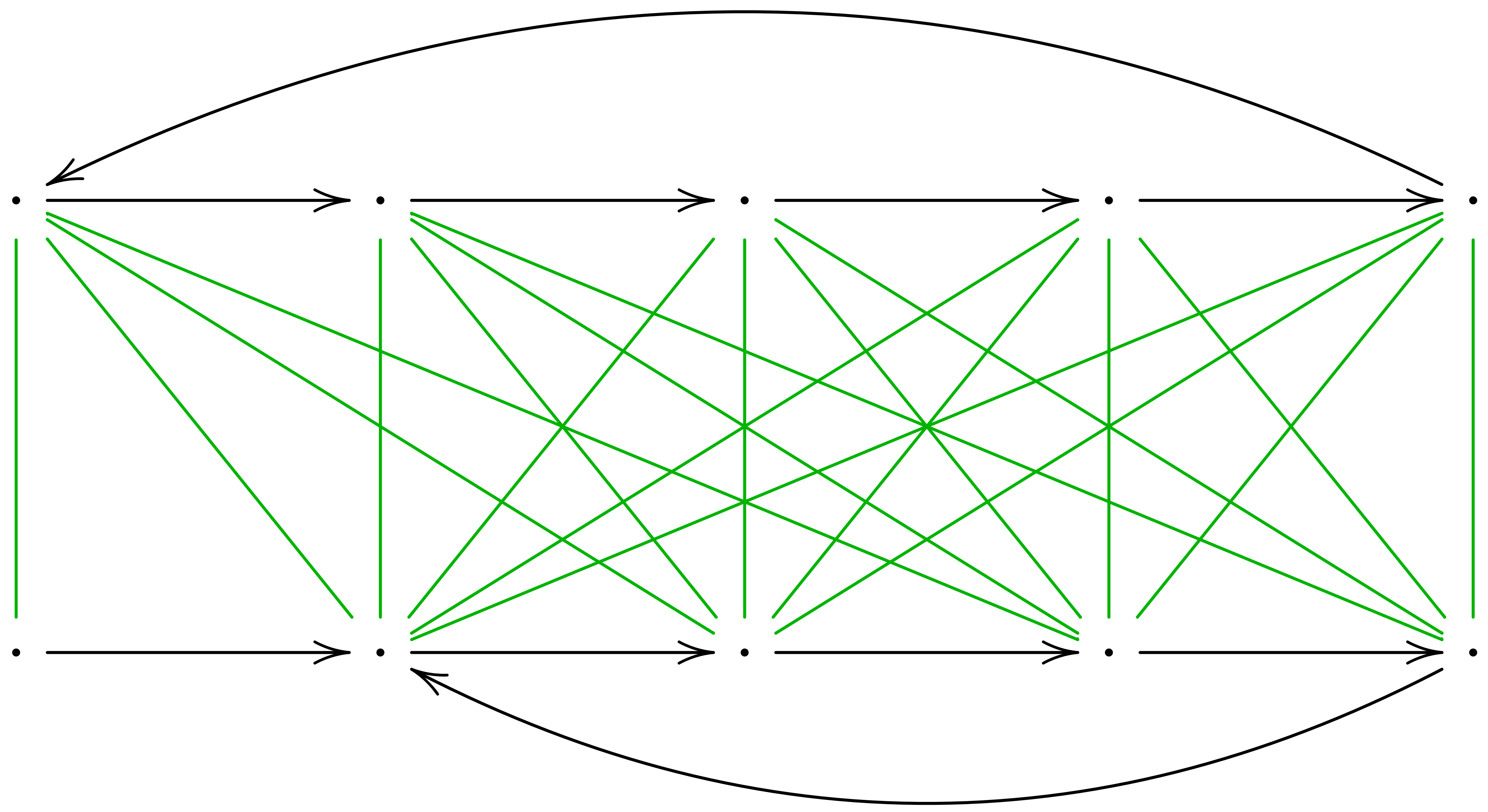
One can stop much earlier



21 pairs

First improvement

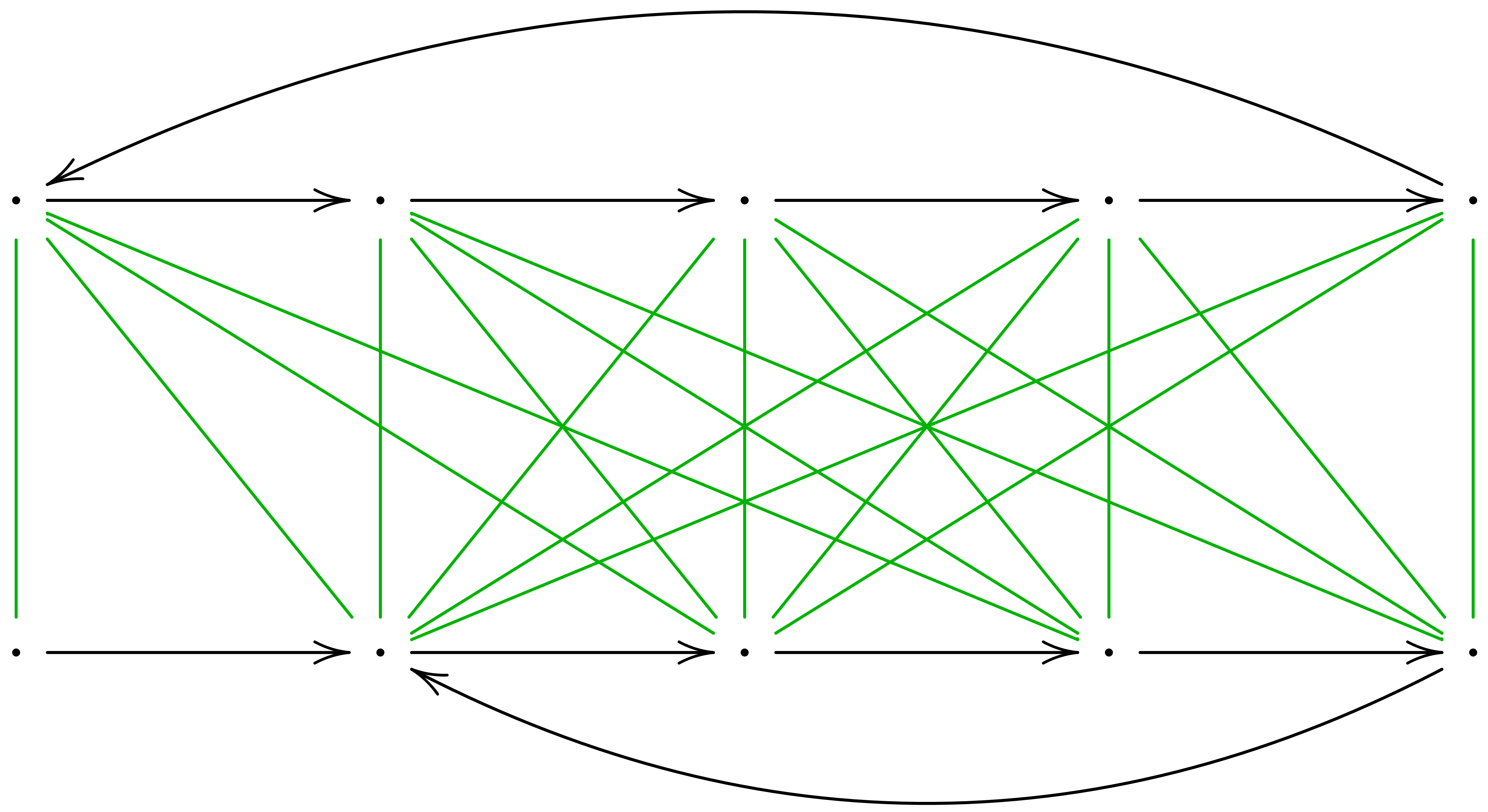
One can stop much earlier



~~21~~ 20 pairs

First improvement

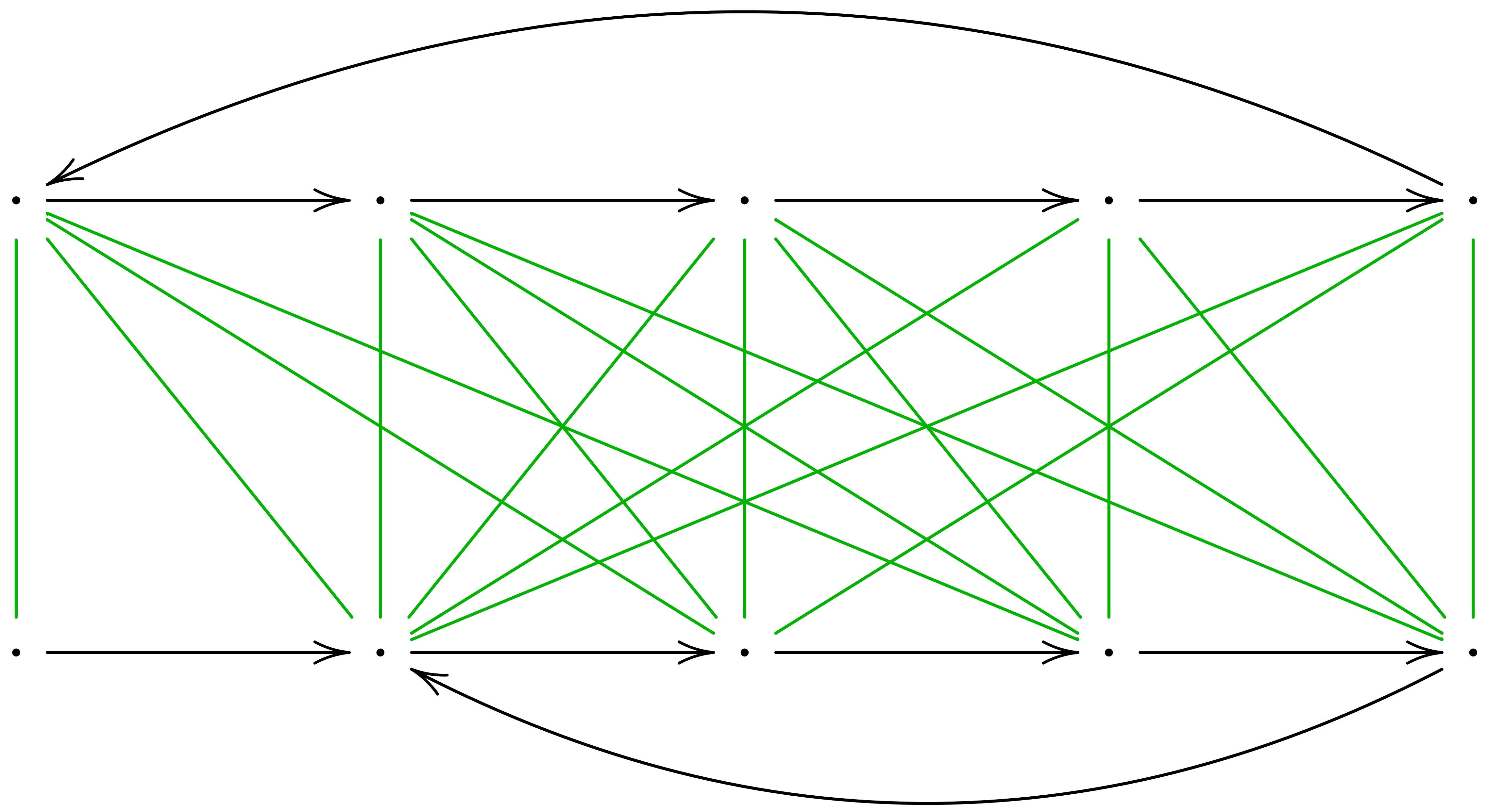
One can stop much earlier



~~21~~ 19 pairs

First improvement

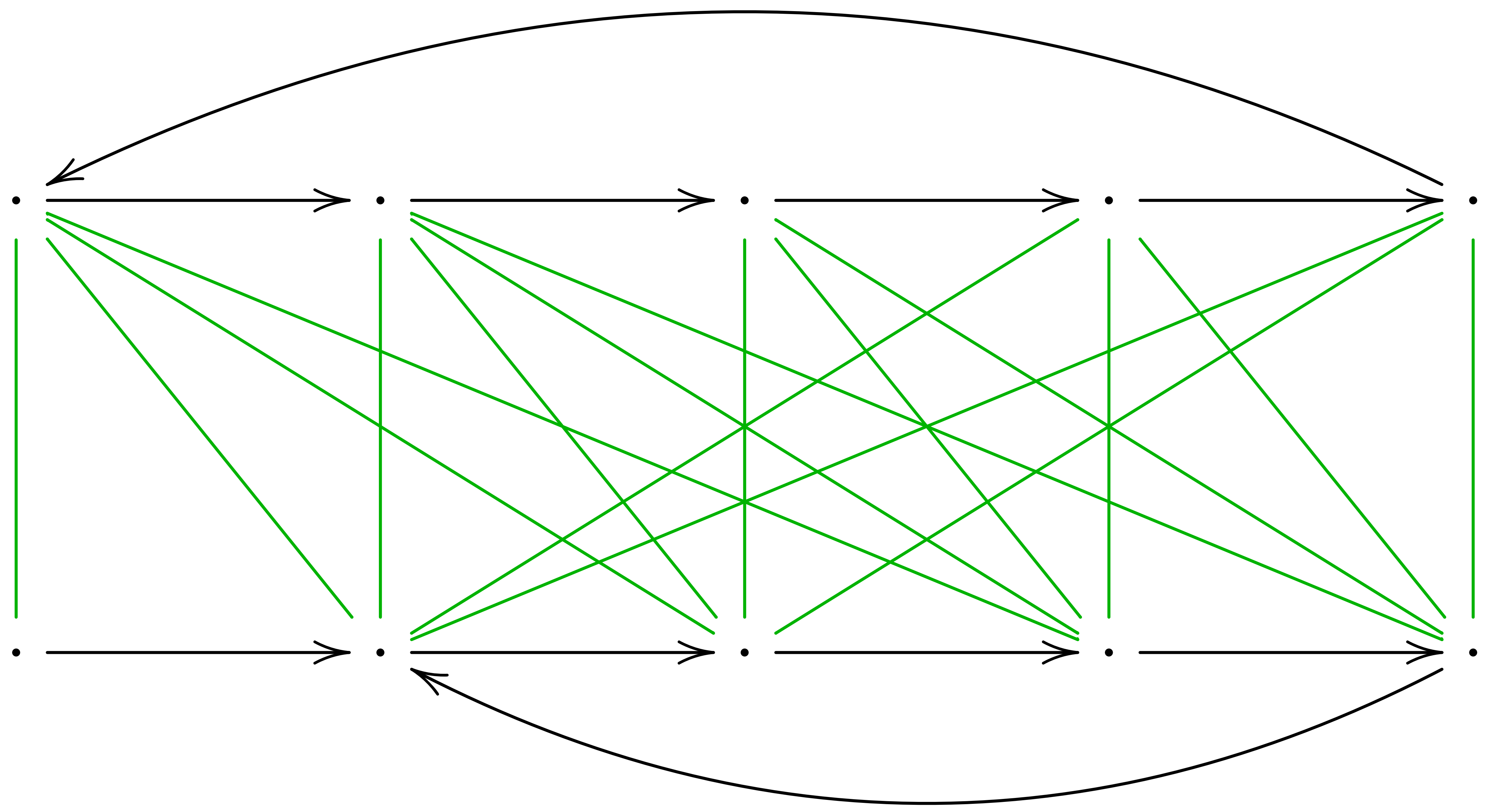
One can stop much earlier



~~21~~ 18 pairs

First improvement

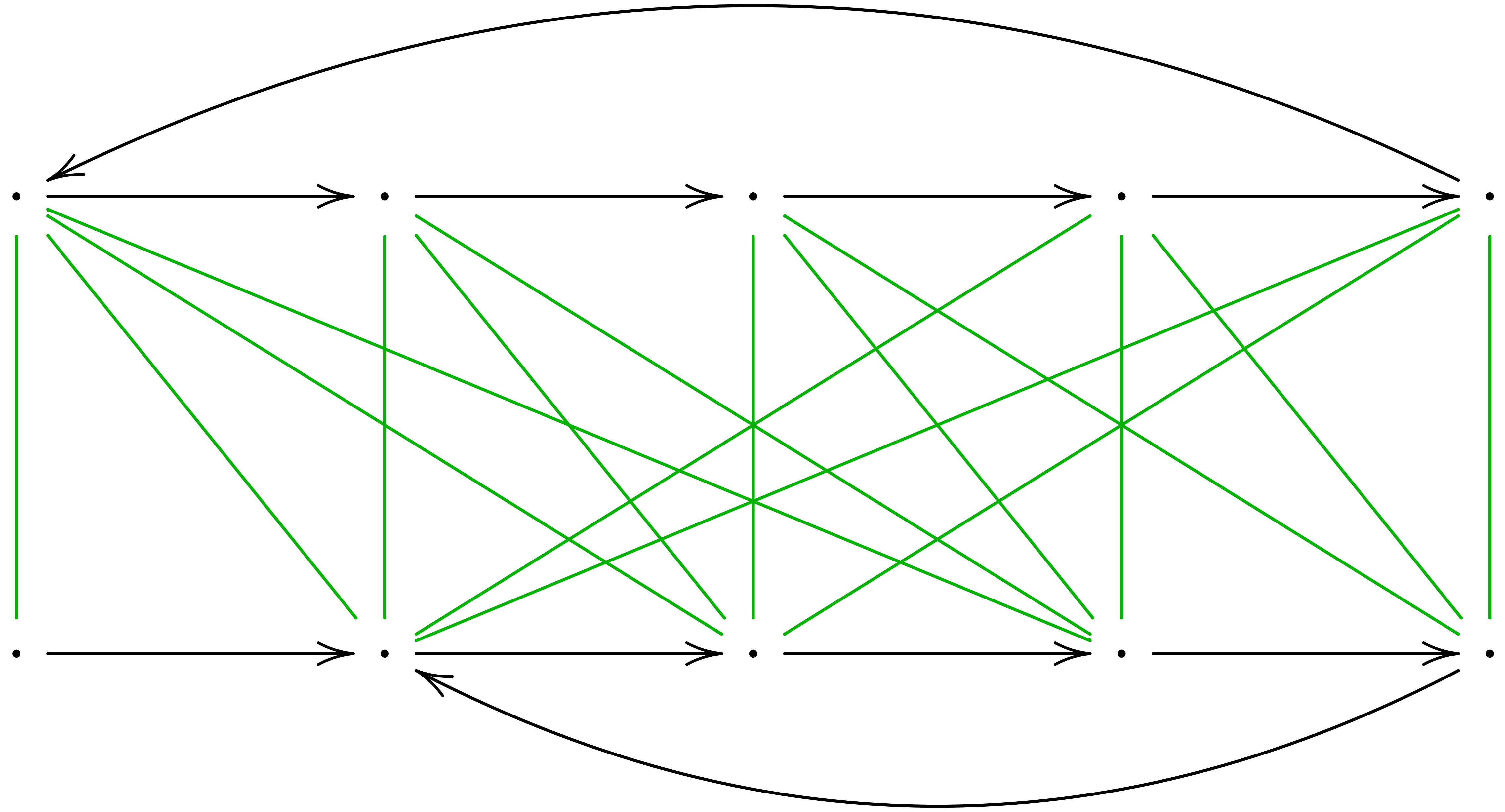
One can stop much earlier



~~21~~ 17 pairs

First improvement

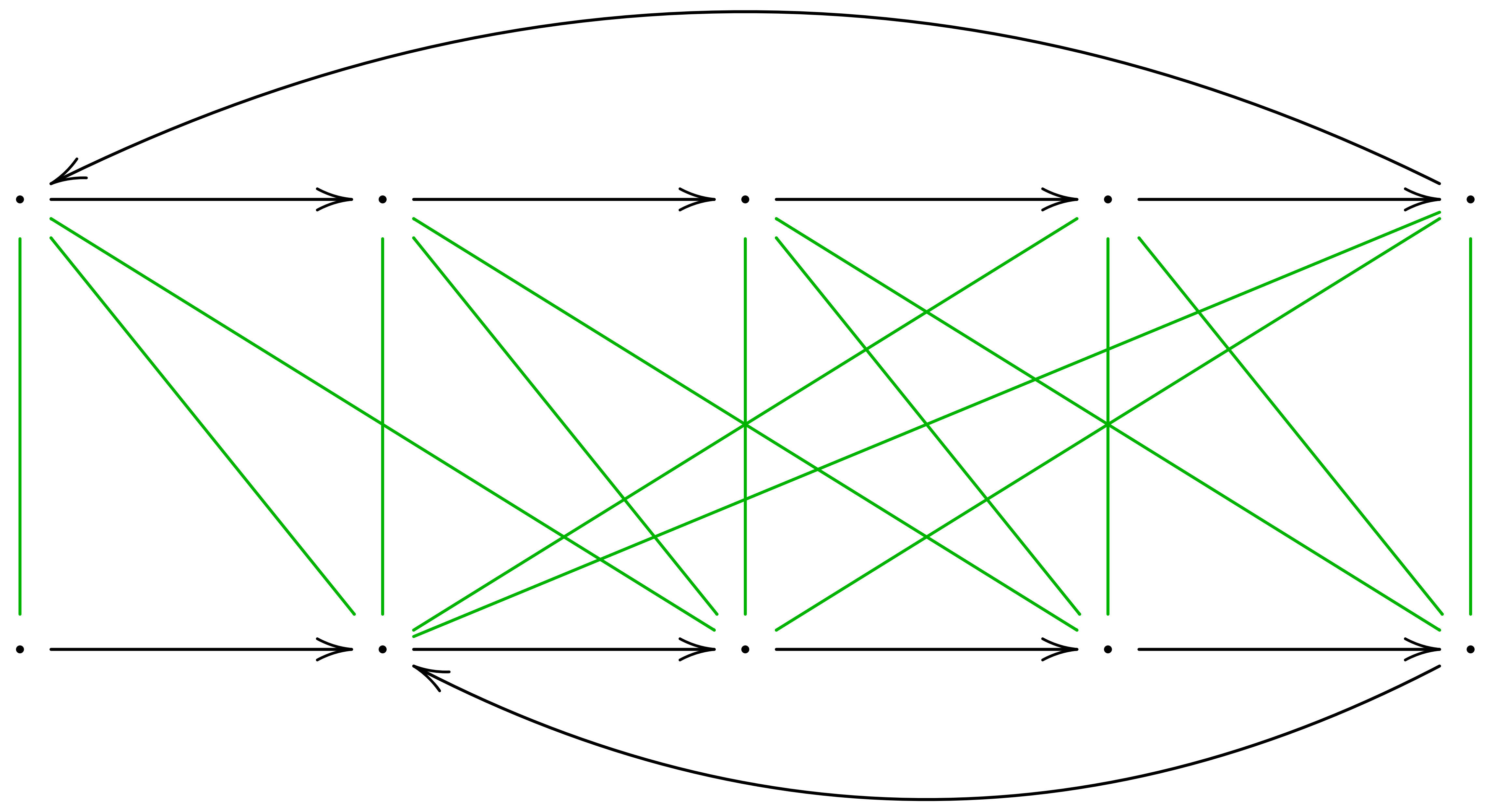
One can stop much earlier



~~21~~ 16 pairs

First improvement

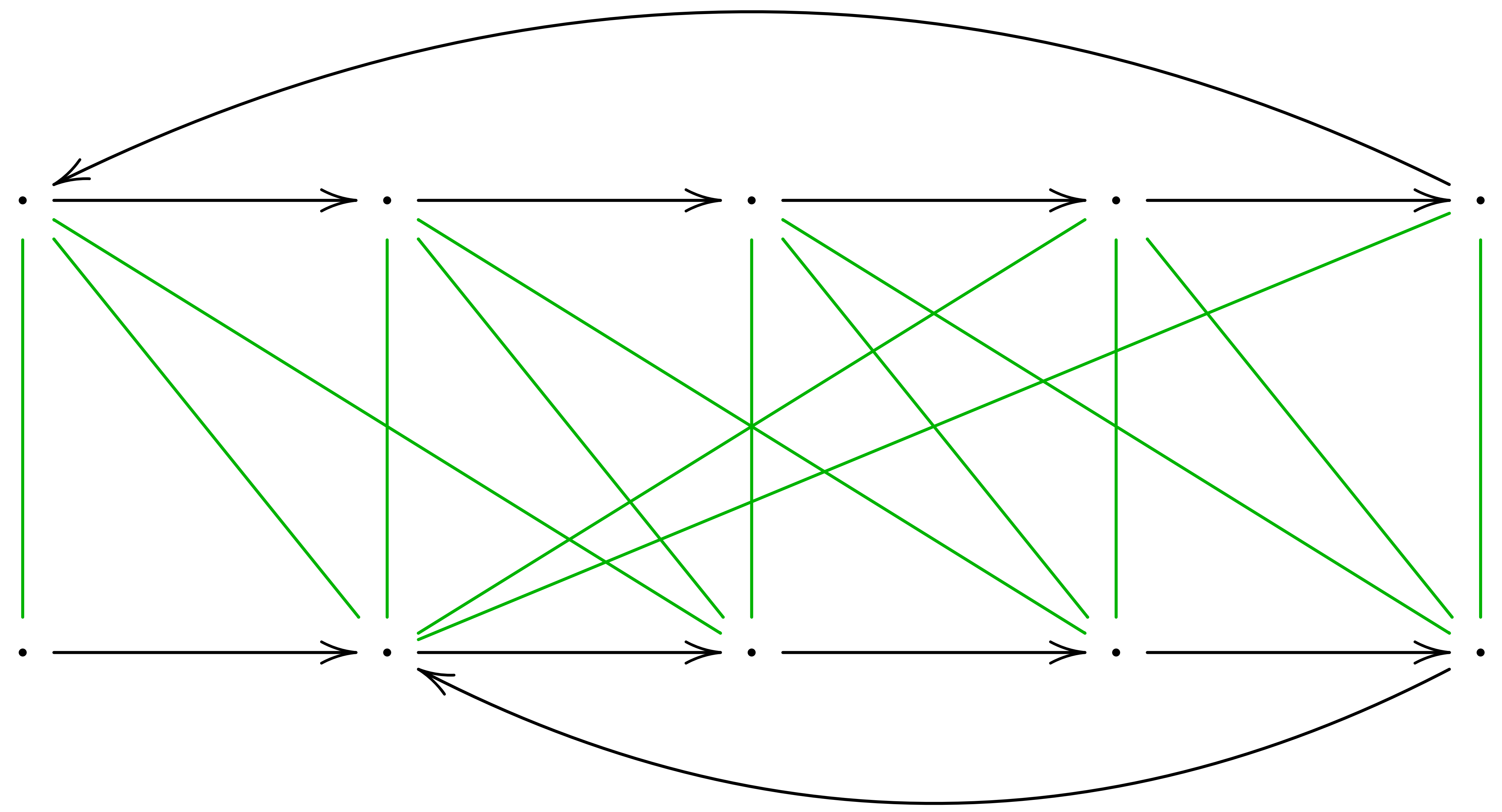
One can stop much earlier



~~21~~ 15 pairs

First improvement

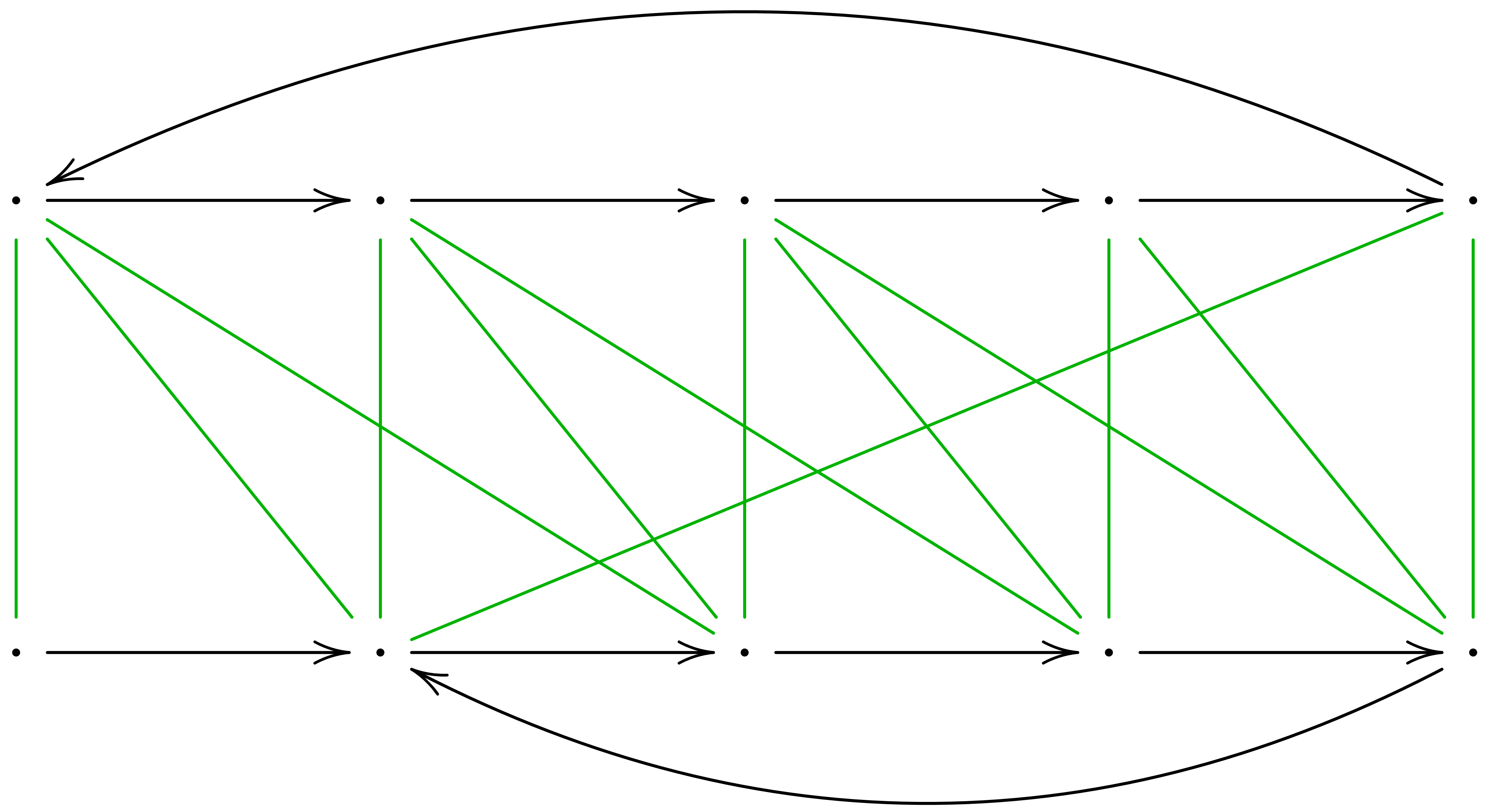
One can stop much earlier



~~21~~ 14 pairs

First improvement

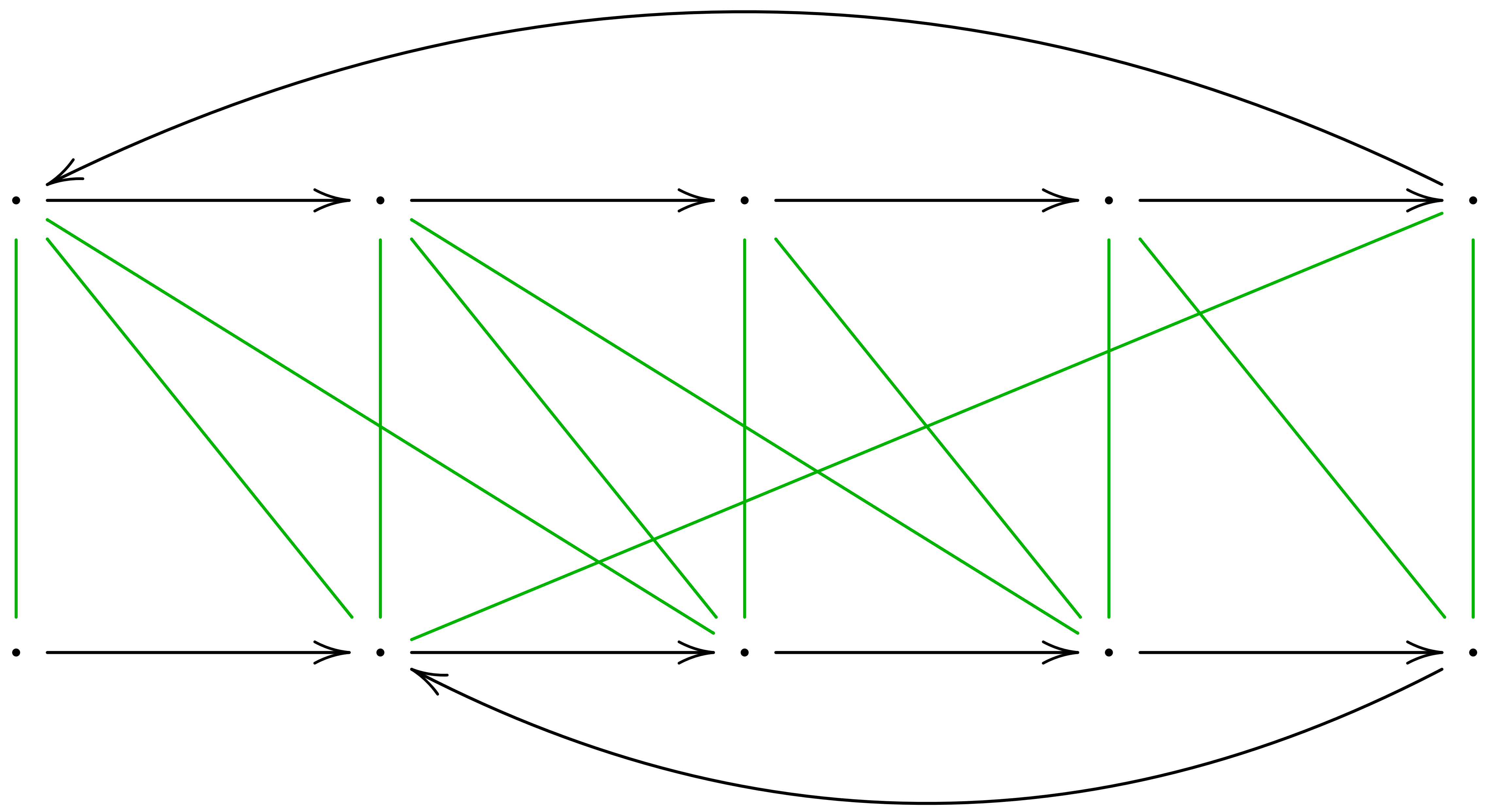
One can stop much earlier



~~21~~ 13 pairs

First improvement

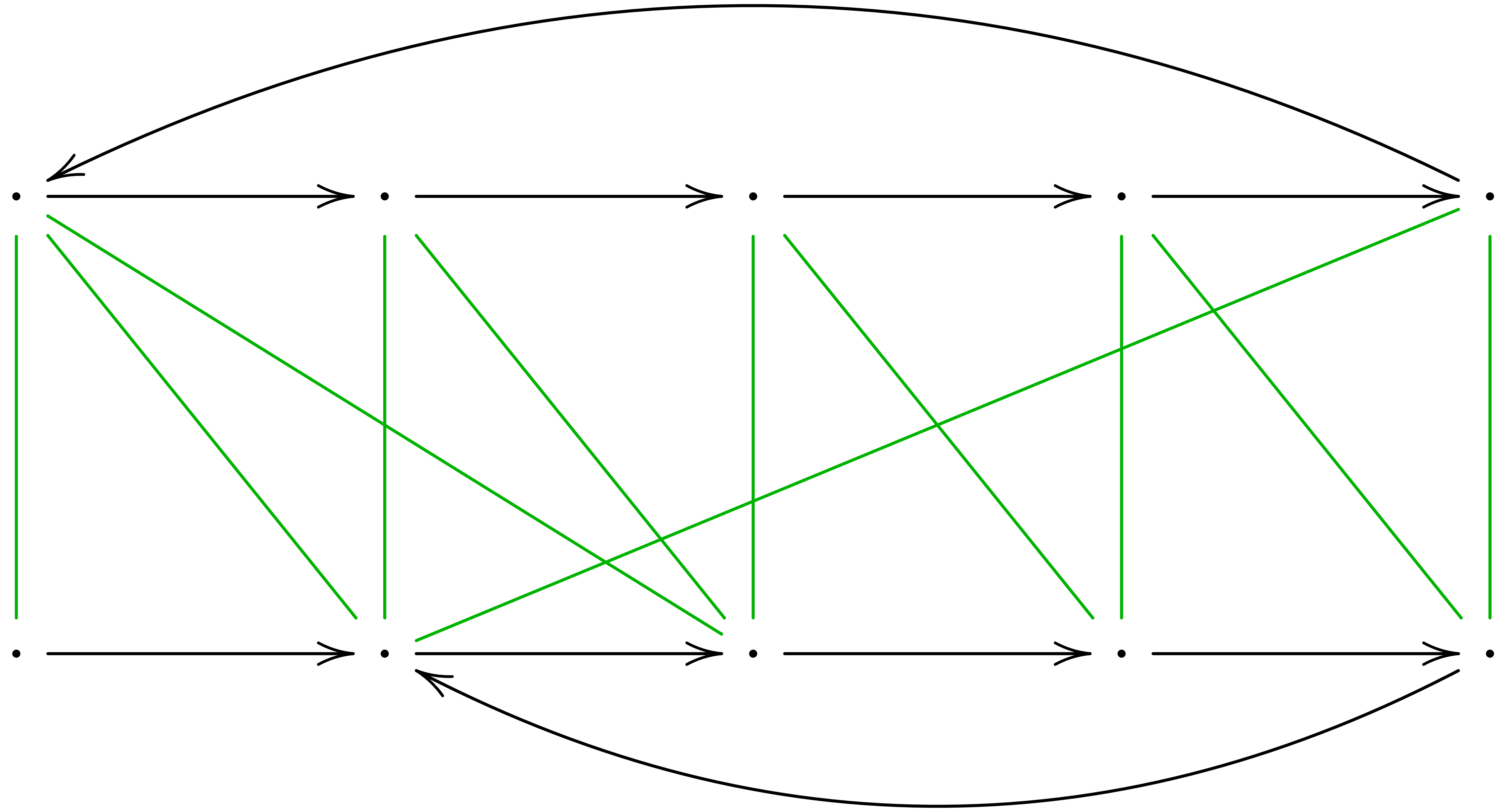
One can stop much earlier



~~21~~ 12 pairs

First improvement

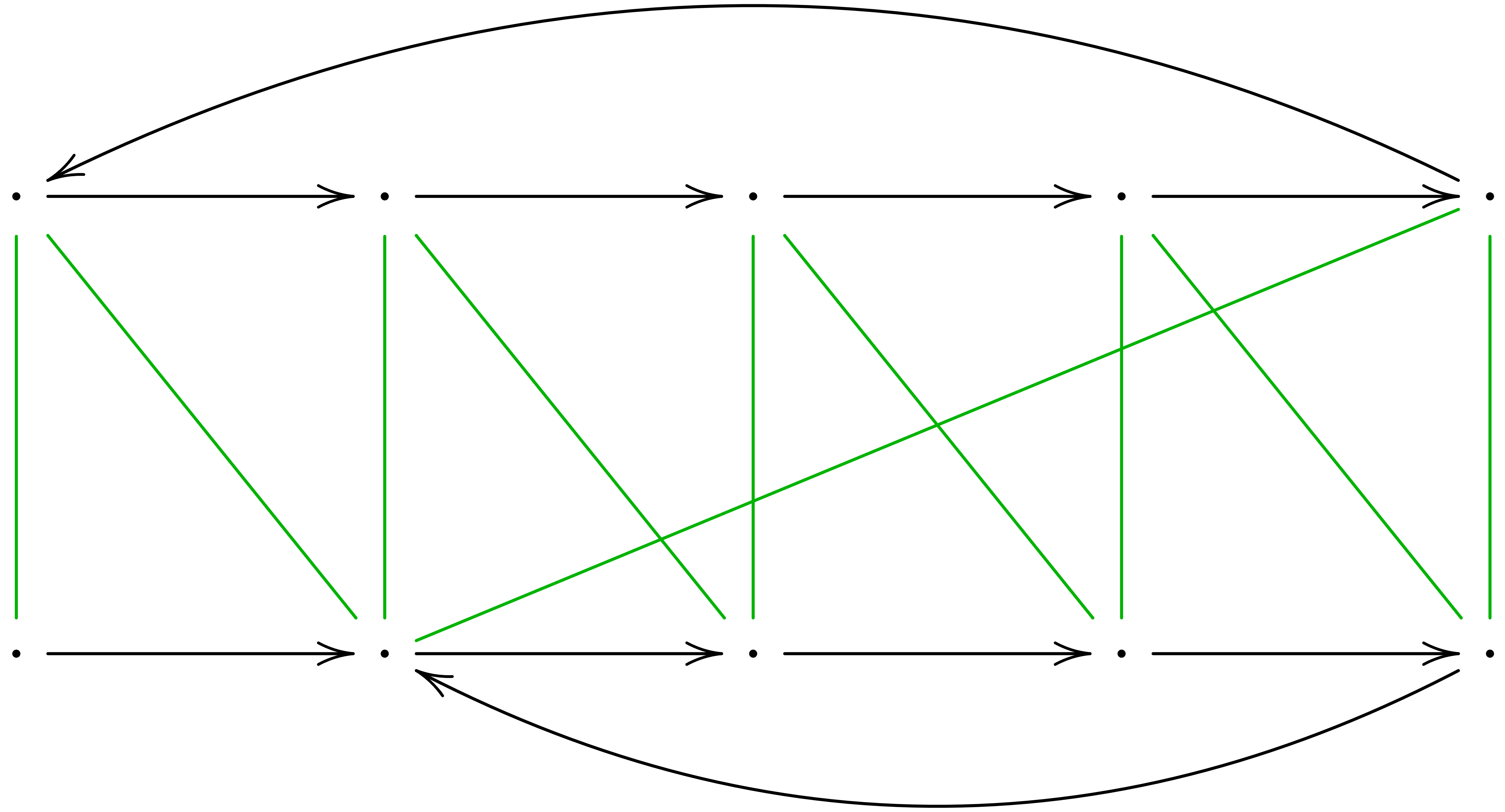
One can stop much earlier



~~21~~ 11 pairs

First improvement

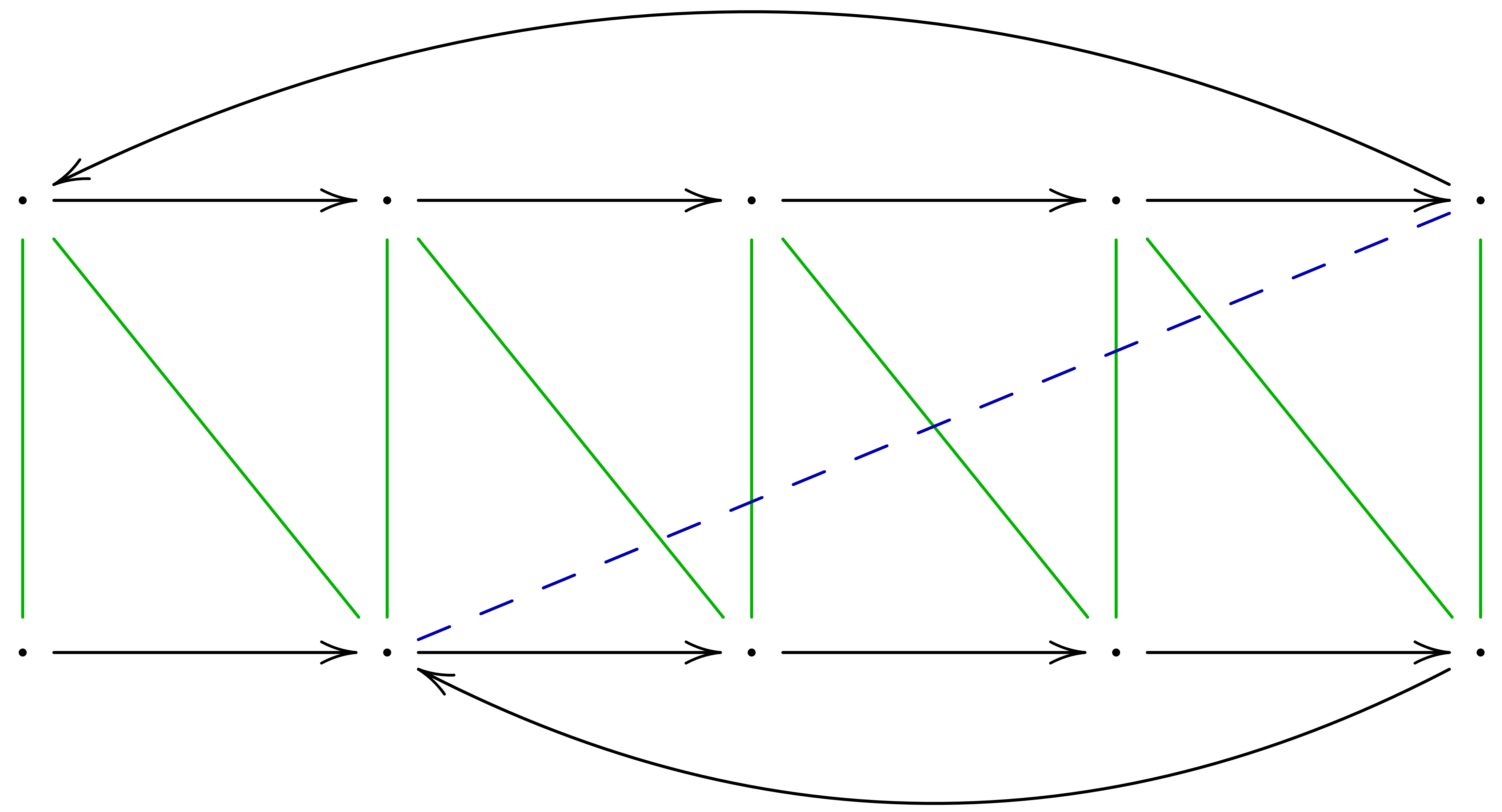
One can stop much earlier



~~21~~ 10 pairs

First improvement

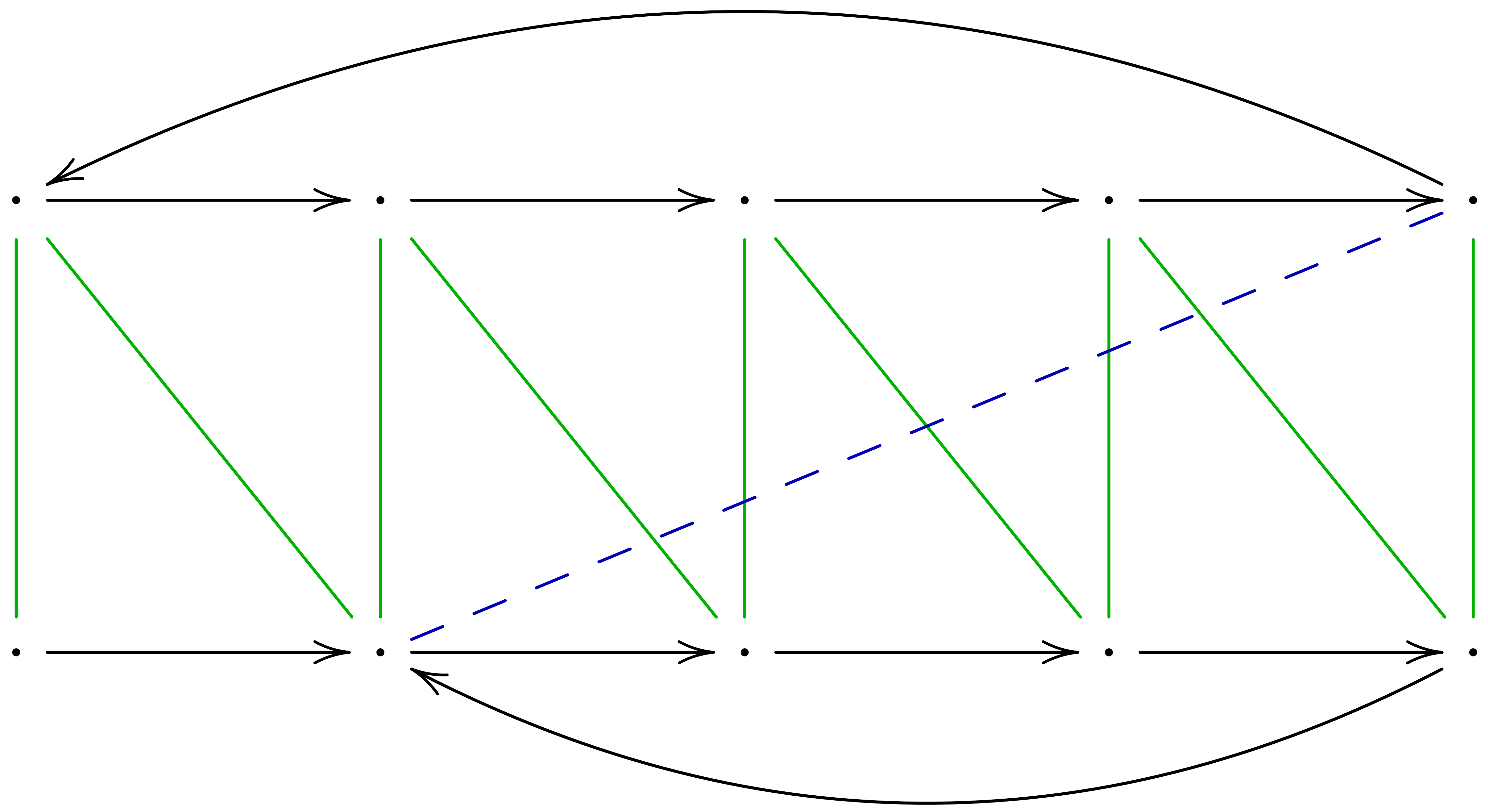
One can stop much earlier



~~21~~ 9 pairs

First improvement

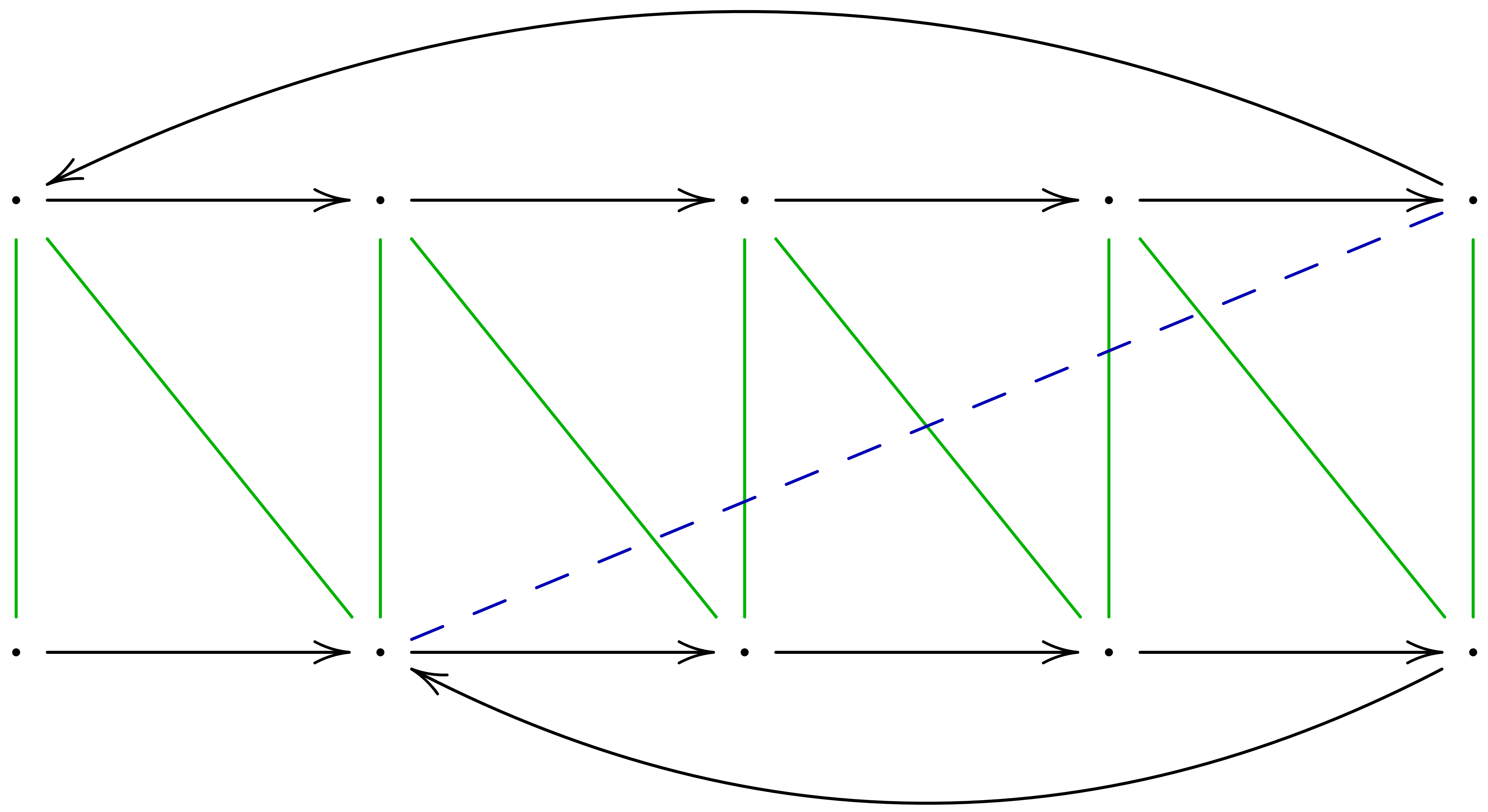
One can stop much earlier



[Hopcroft and Karp '71]

First improvement

One can stop much earlier



[Hopcroft and Karp '71]

Complexity: almost linear

[Tarjan '75]

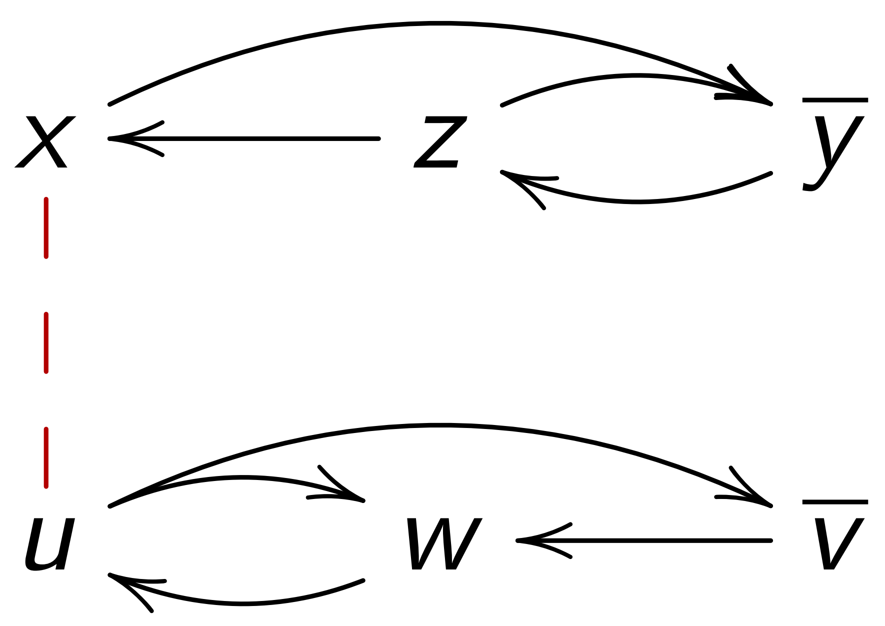
Correctness of the improvement

Correctness of HK algorithm, revisited:

- The previous relation is not a bisimulation - proof of equivalence
- But can be completed to one using equivalence - transitivity
- Hopcroft and Karp's algorithm ('71) attempts to construct a bisimulation up to equivalence

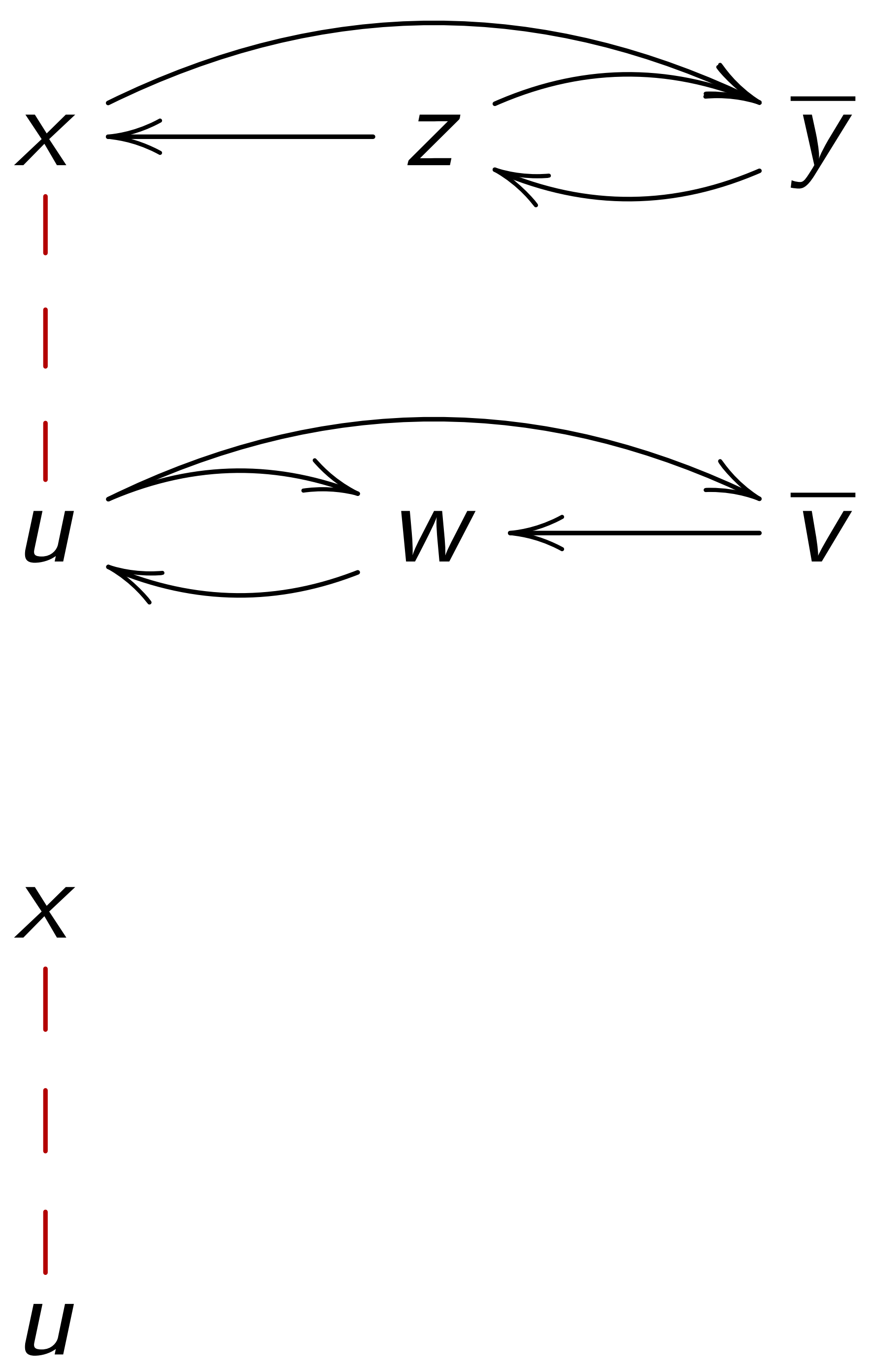
Non-Deterministic Automata

Use Hopcroft and Karp on the fly, through the powerset construction:



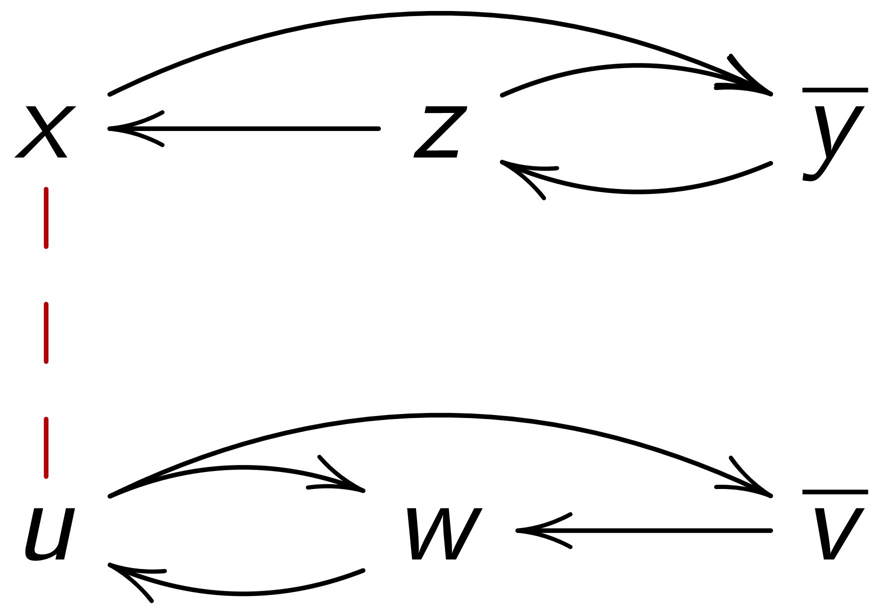
Non-Deterministic Automata

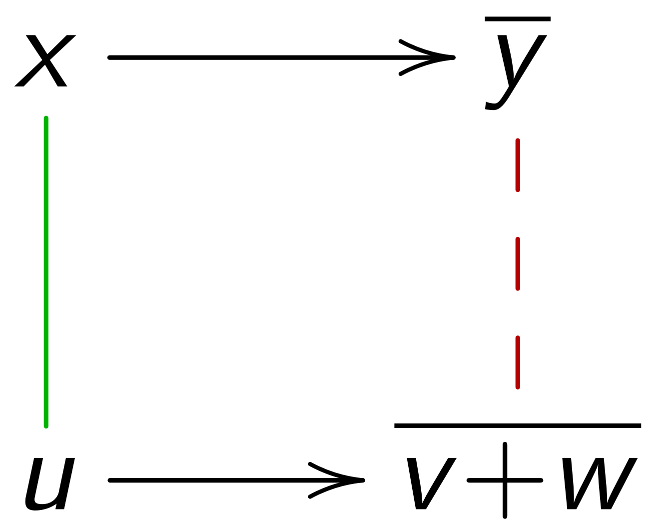
Use Hopcroft and Karp on the fly, through the powerset construction:



Non-Deterministic Automata

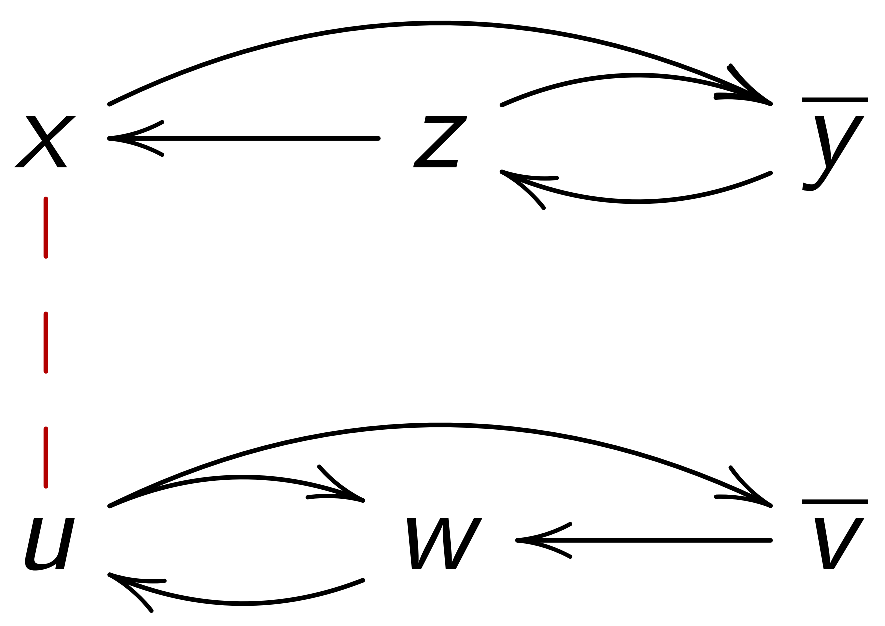
Use Hopcroft and Karp on the fly, through the powerset construction:

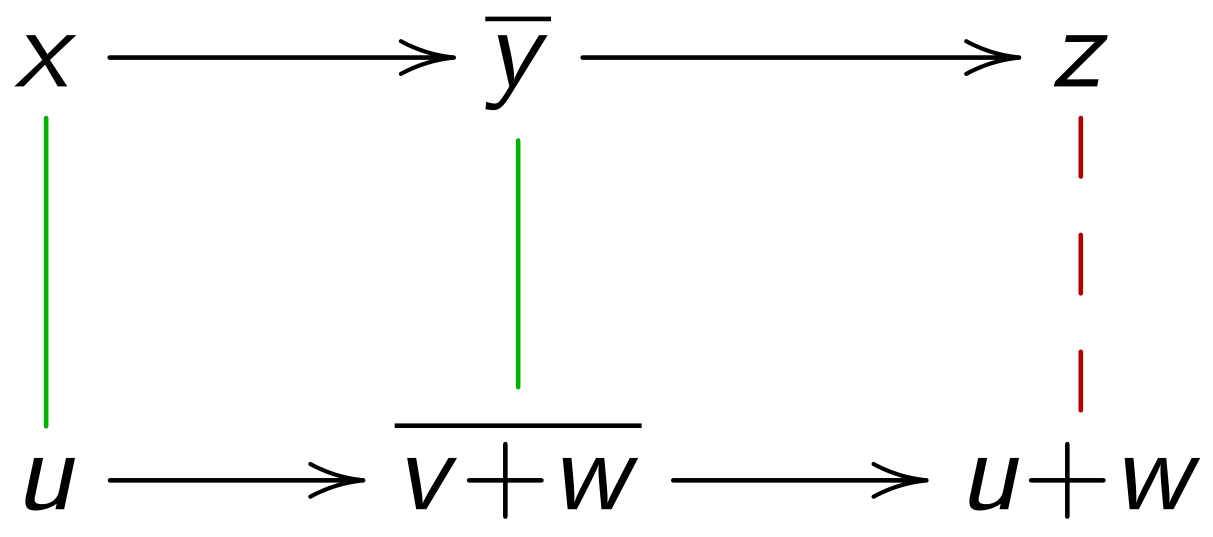




Non-Deterministic Automata

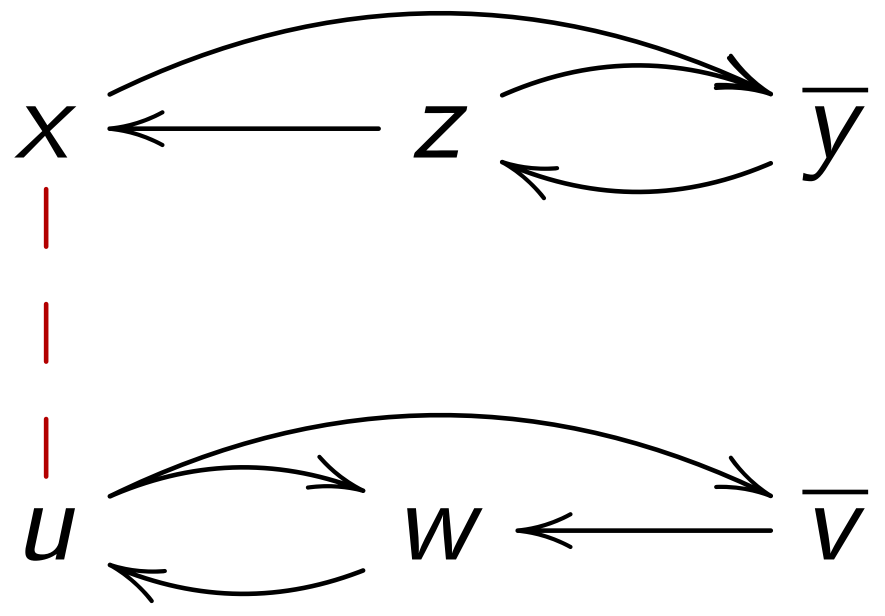
Use Hopcroft and Karp on the fly, through the powerset construction:

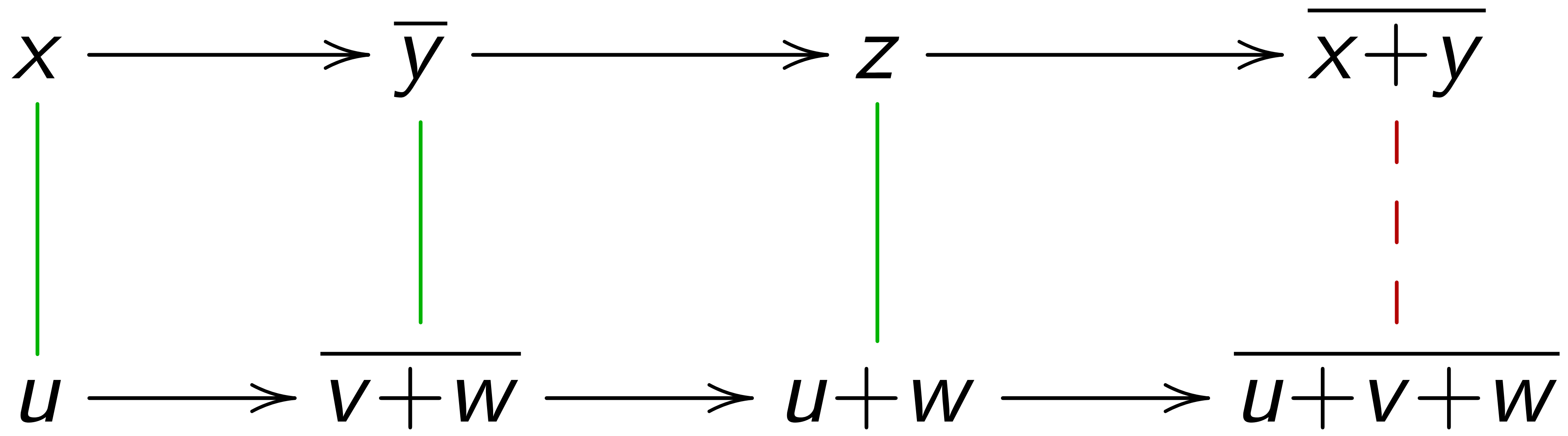




Non-Deterministic Automata

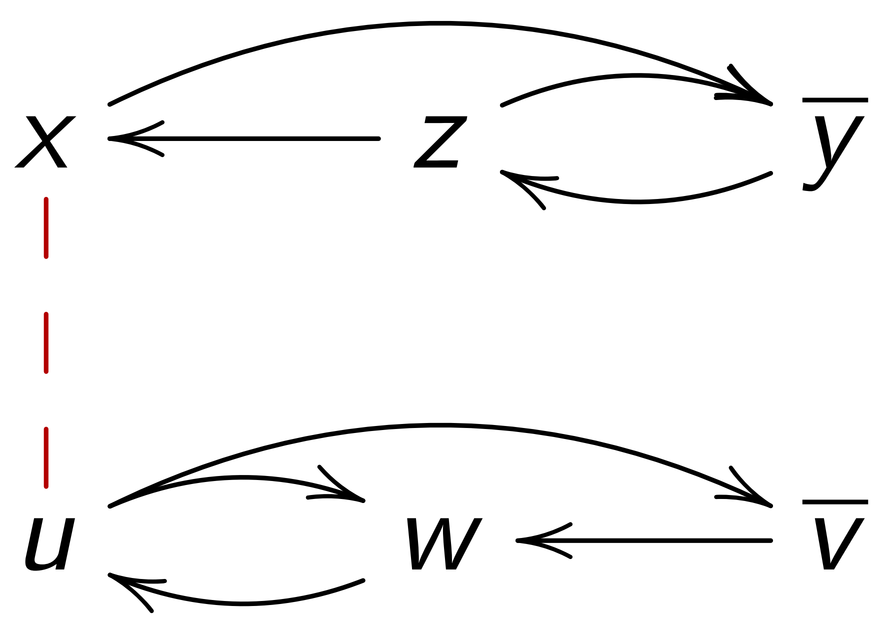
Use Hopcroft and Karp on the fly, through the powerset construction:

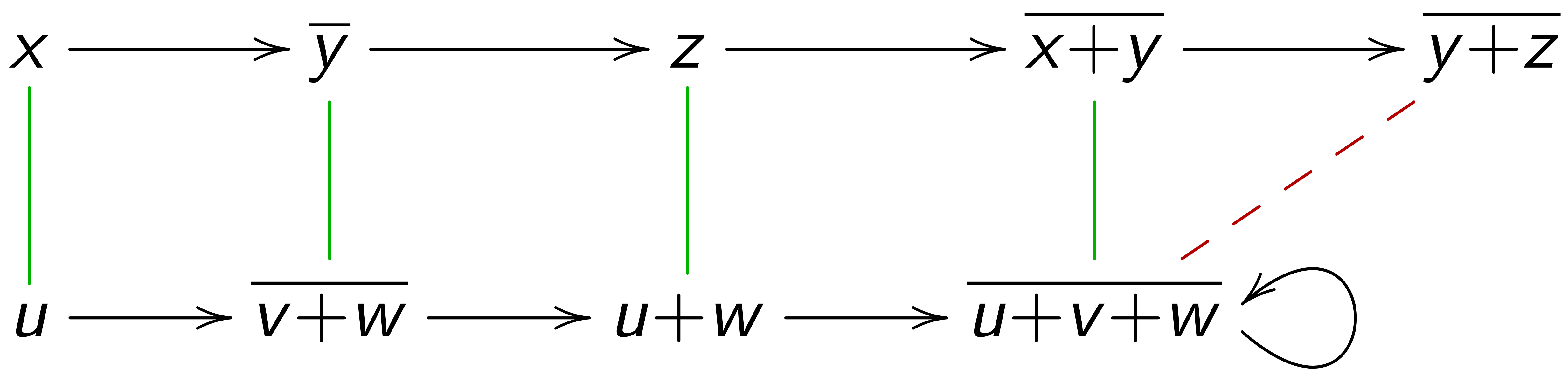




Non-Deterministic Automata

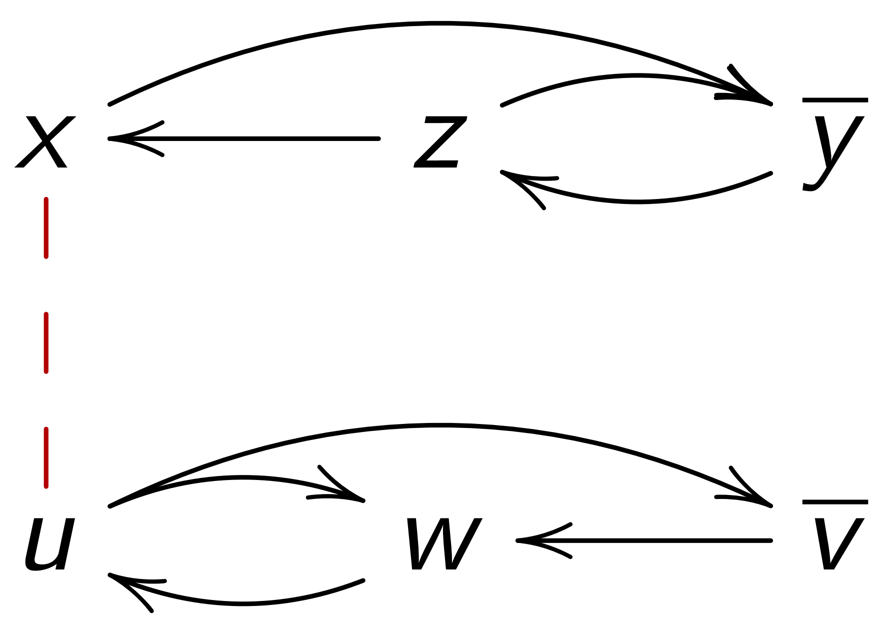
Use Hopcroft and Karp on the fly, through the powerset construction:

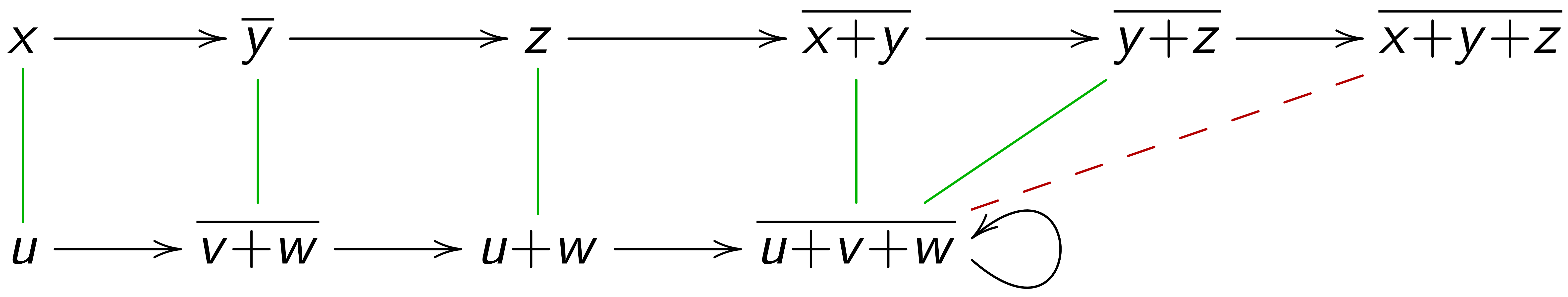




Non-Deterministic Automata

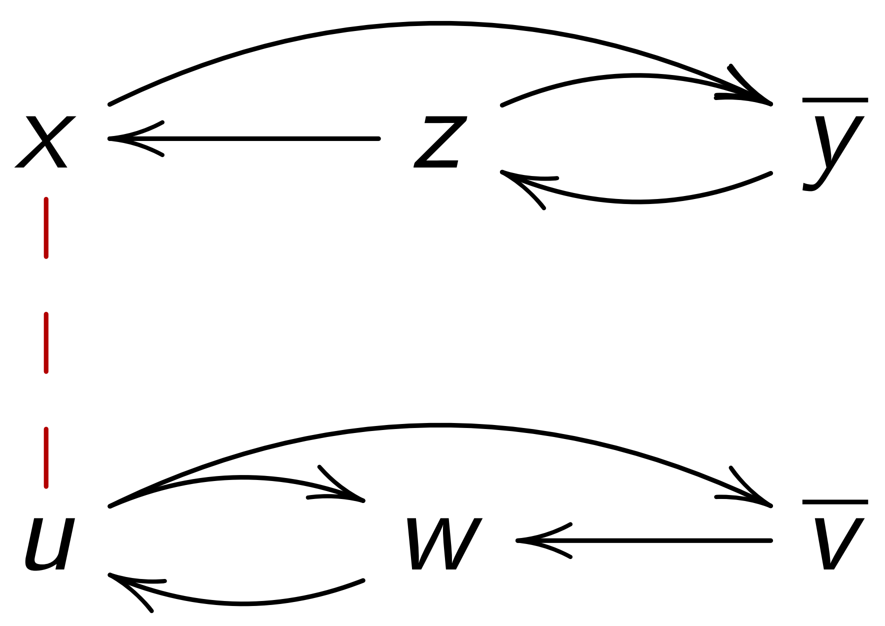
Use Hopcroft and Karp on the fly, through the powerset construction:

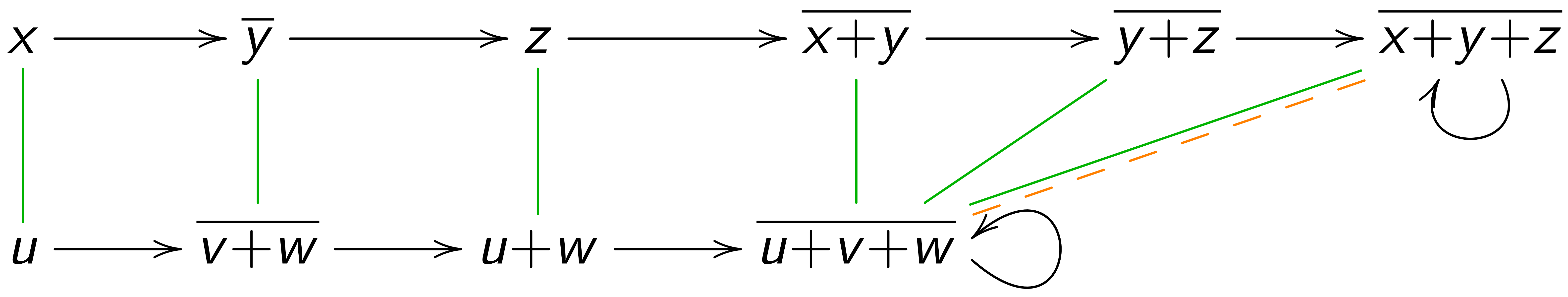




Non-Deterministic Automata

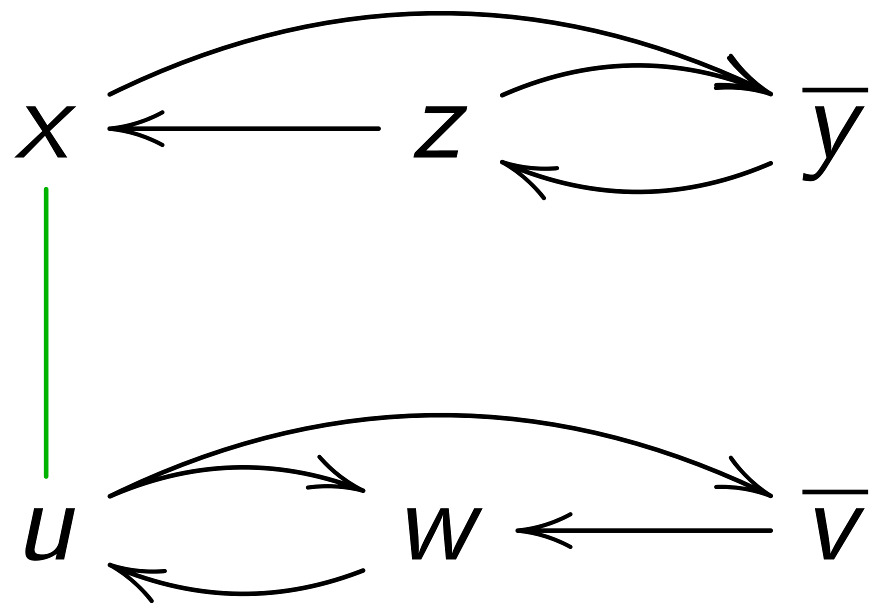
Use Hopcroft and Karp on the fly, through the powerset construction:

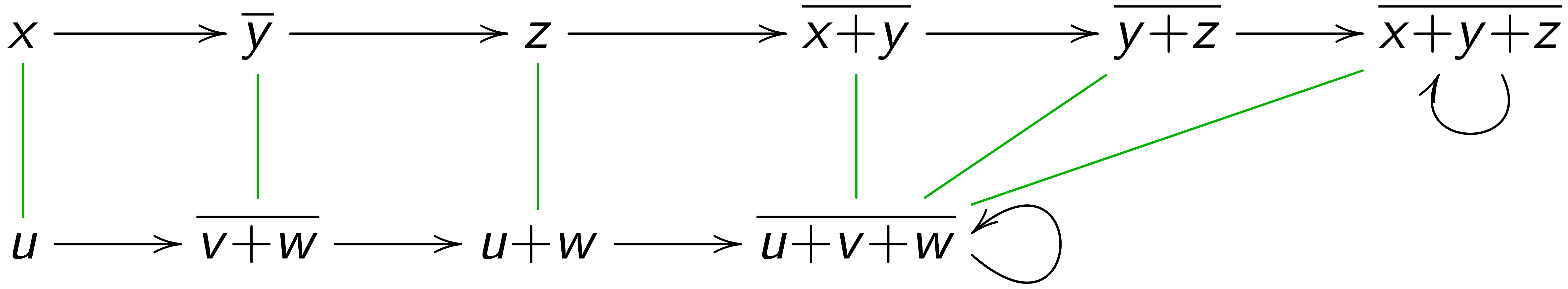




Non-Deterministic Automata

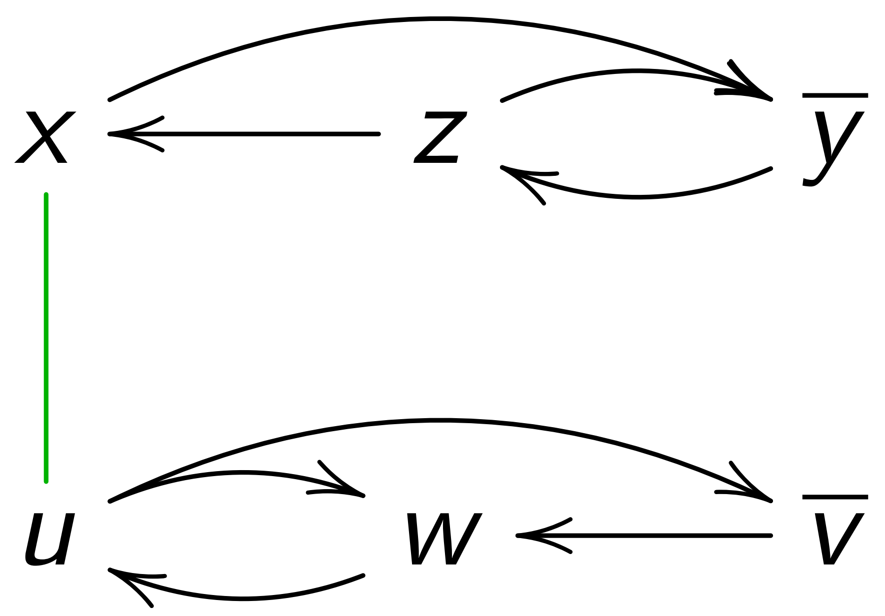
Use Hopcroft and Karp on the fly, through the powerset construction:

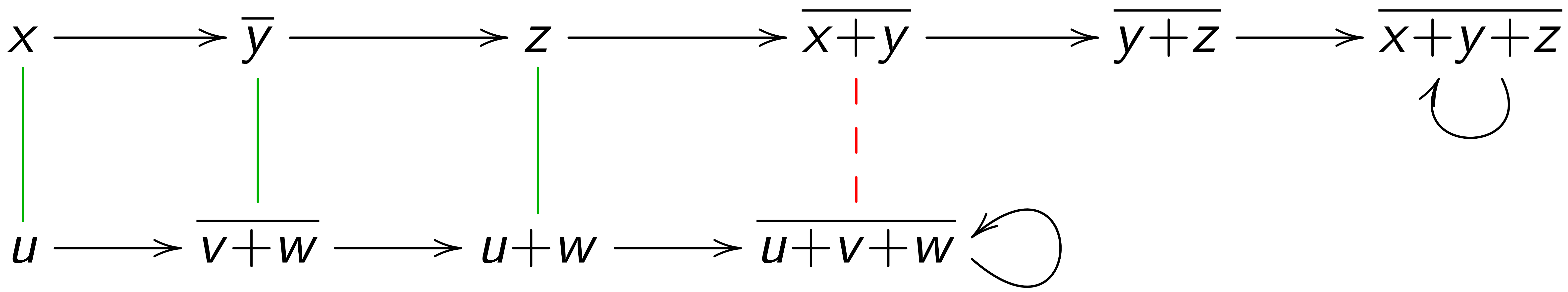




Non-Deterministic Automata

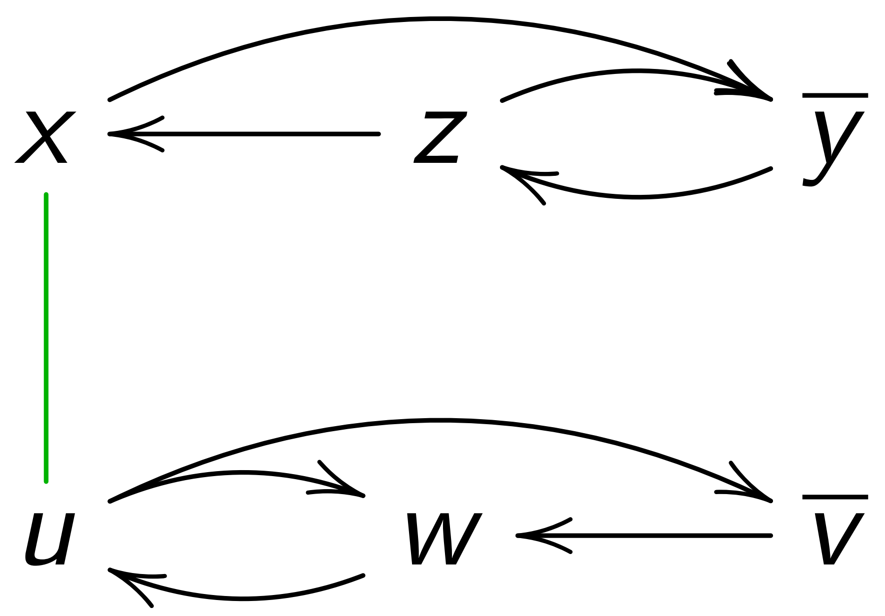
One can do better:



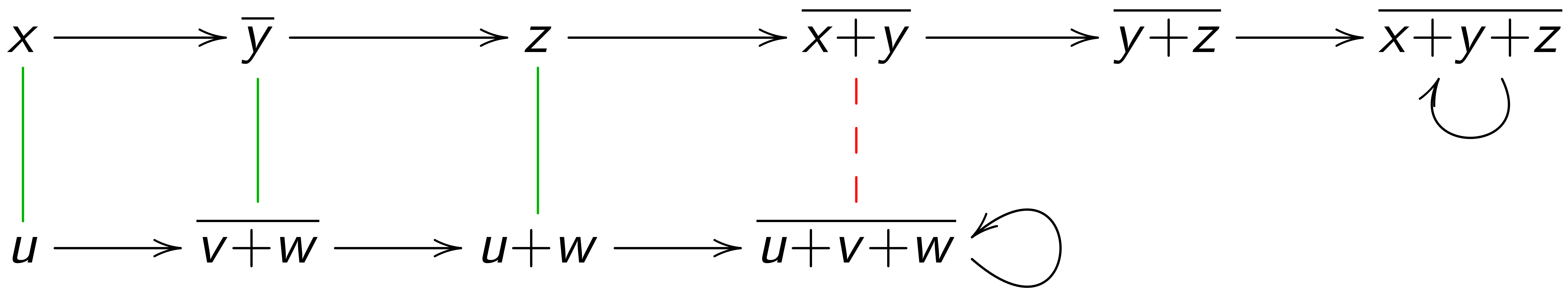


Non-Deterministic Automata

One can do better:

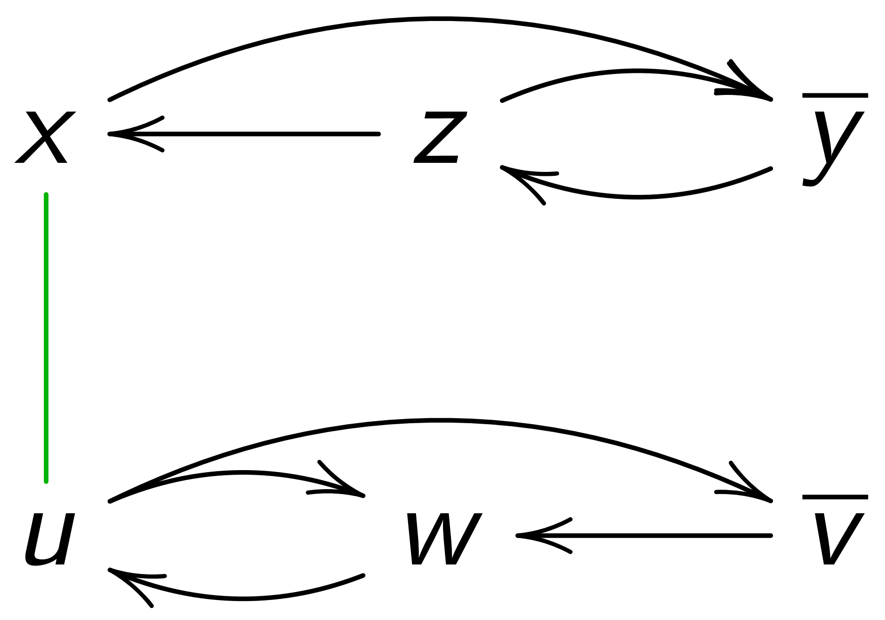


$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v+w) \\ \hline = & (x+y,\ u+v+w) \end{array}$$

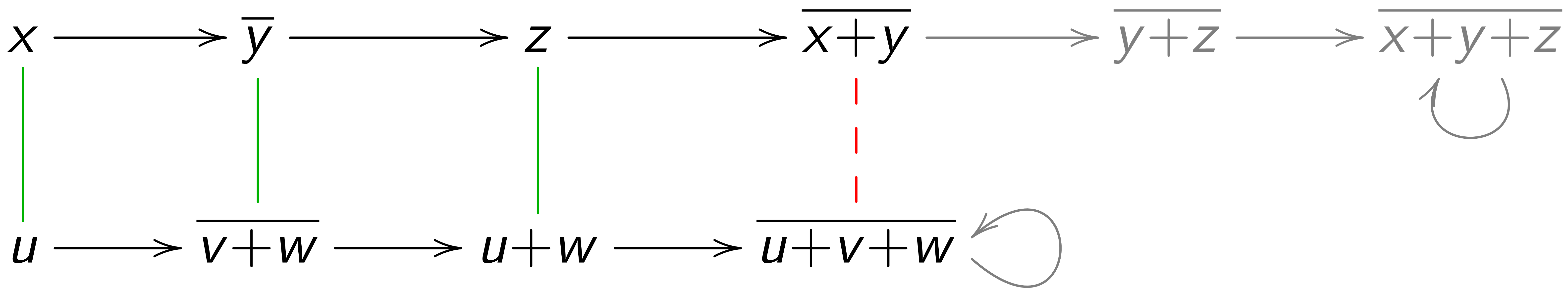


Non-Deterministic Automata

One can do better:



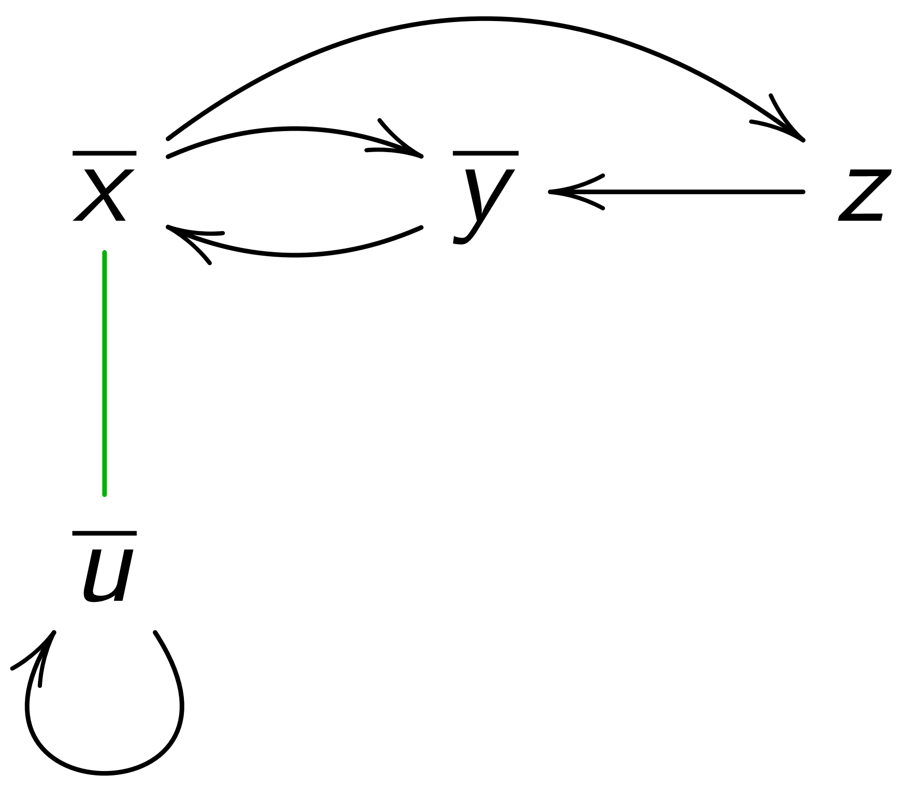
$$
\begin{array}{rl}
 & (x,\ u) \\
+ & (y,\ v+w) \\
\hline
= & (x+y,\ u+v+w)
\end{array}
$$

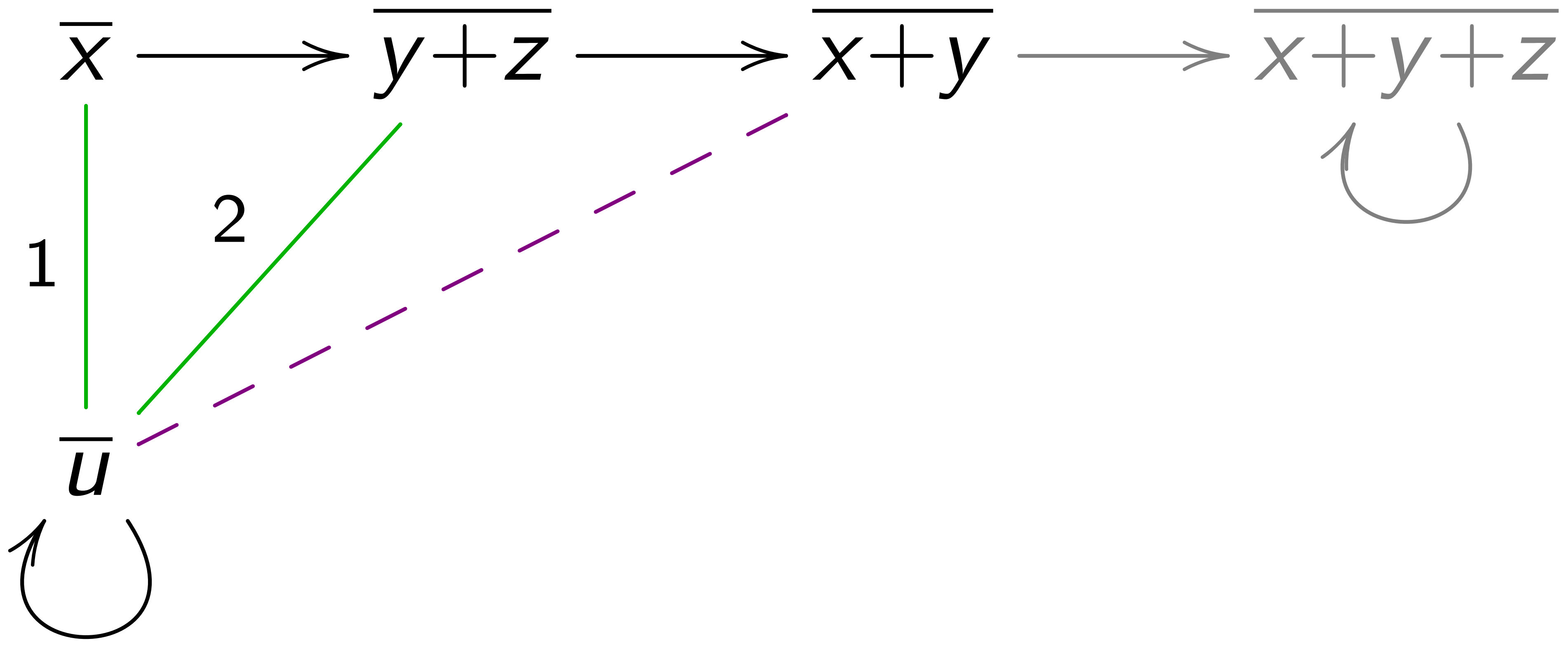


using bisimulations up to union

Non-Deterministic Automata

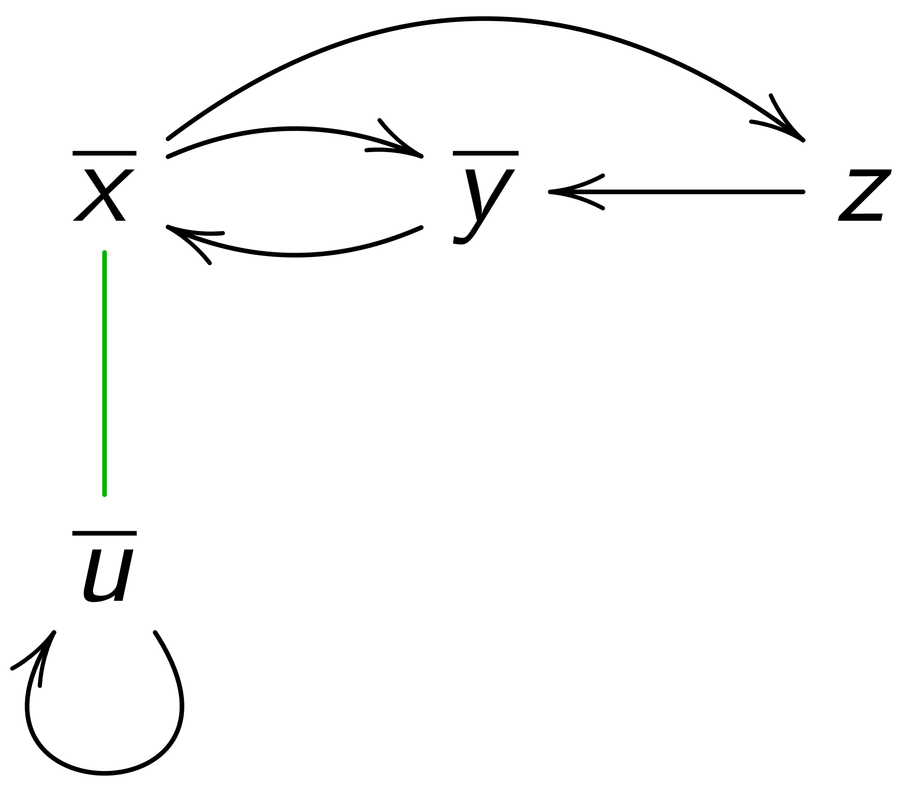
One can do even better:



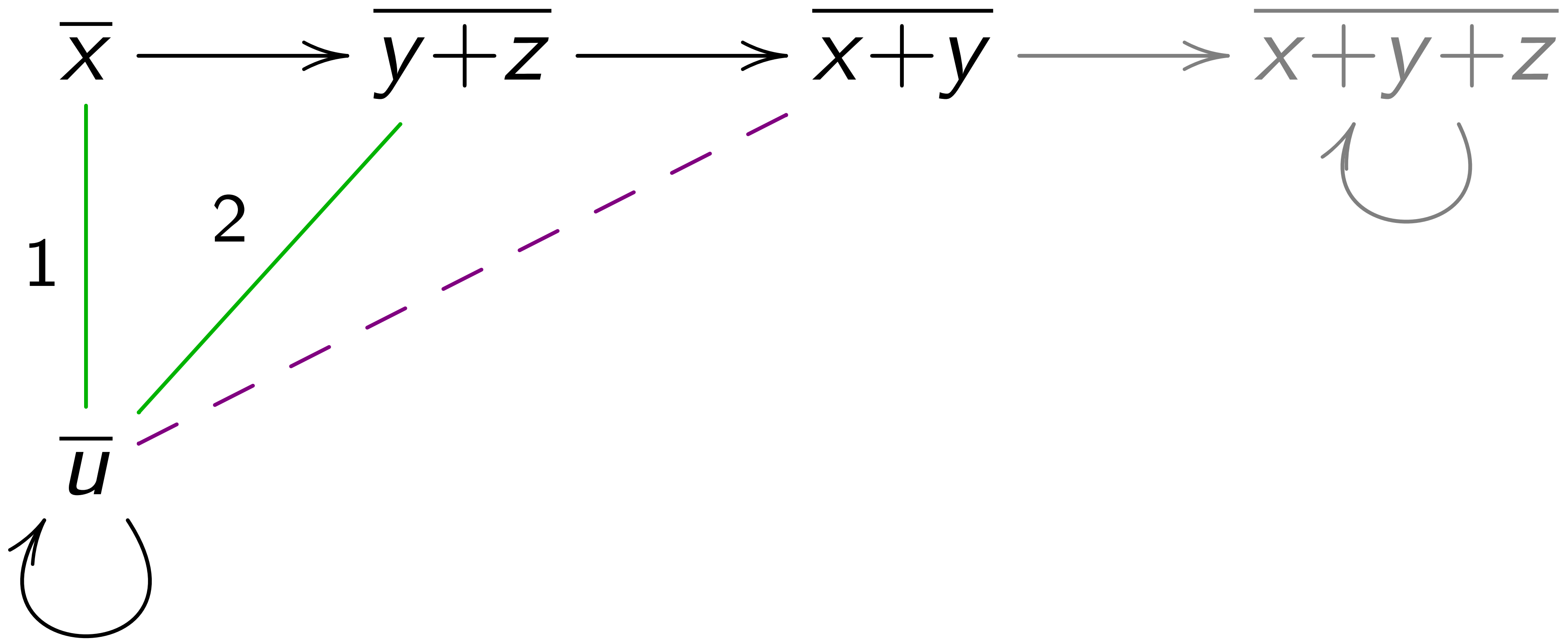


Non-Deterministic Automata

One can do even better:

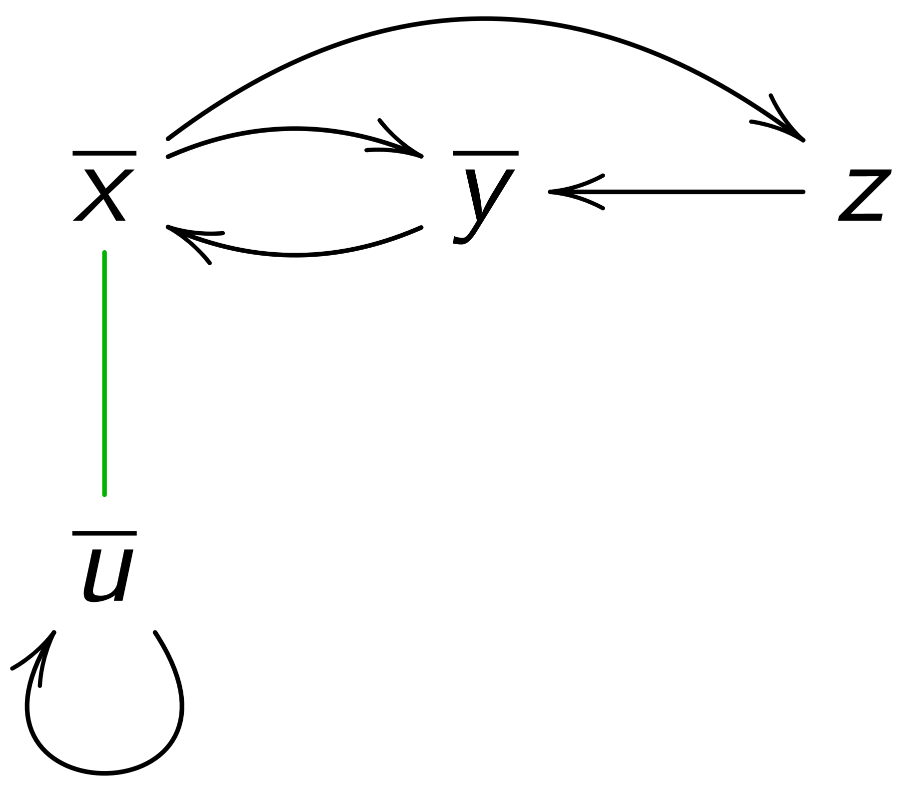


$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z &&\\
&= u && (2)
\end{aligned}
$$

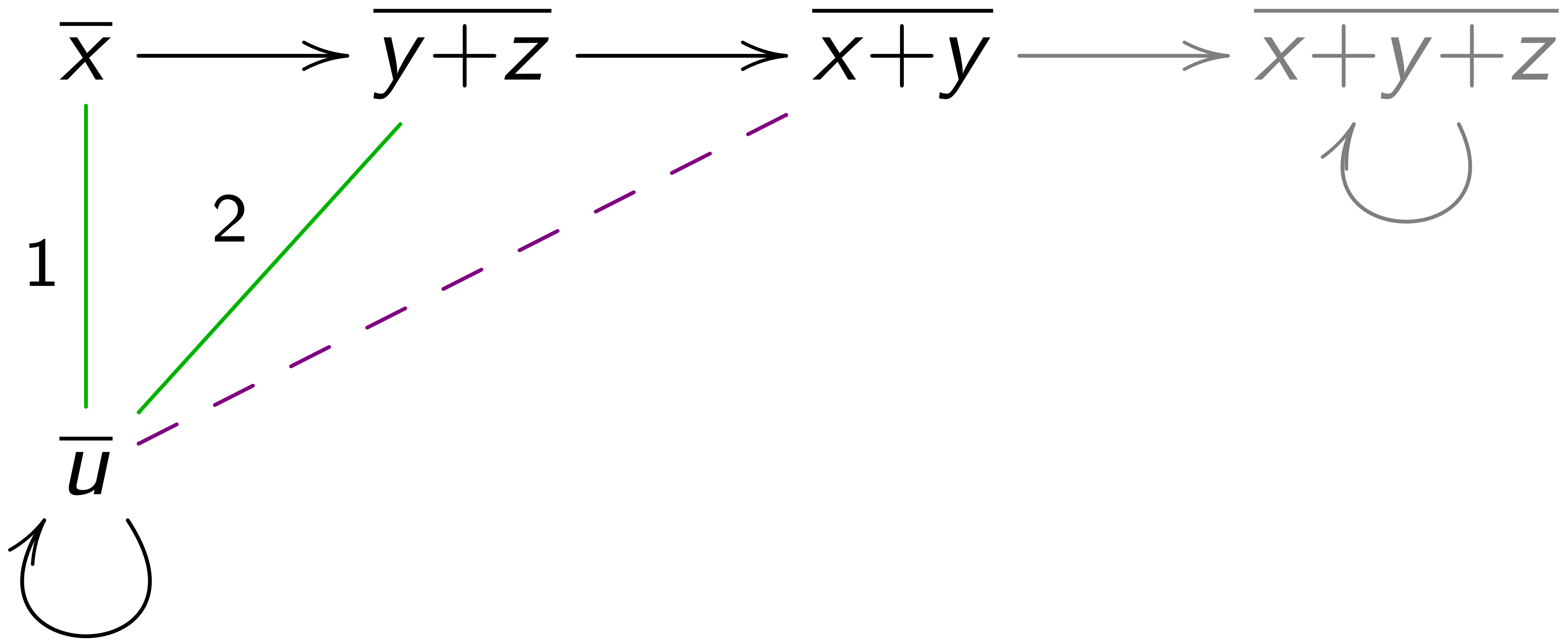


Non-Deterministic Automata

One can do even better:

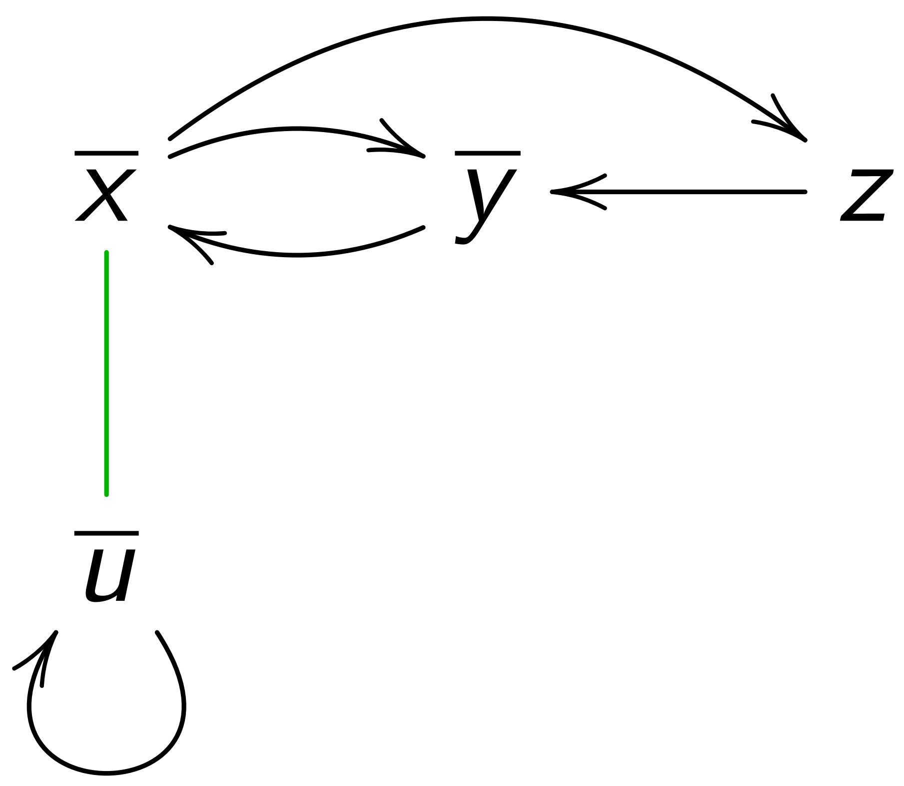


$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z &&\\
&= u && (2)
\end{aligned}
$$

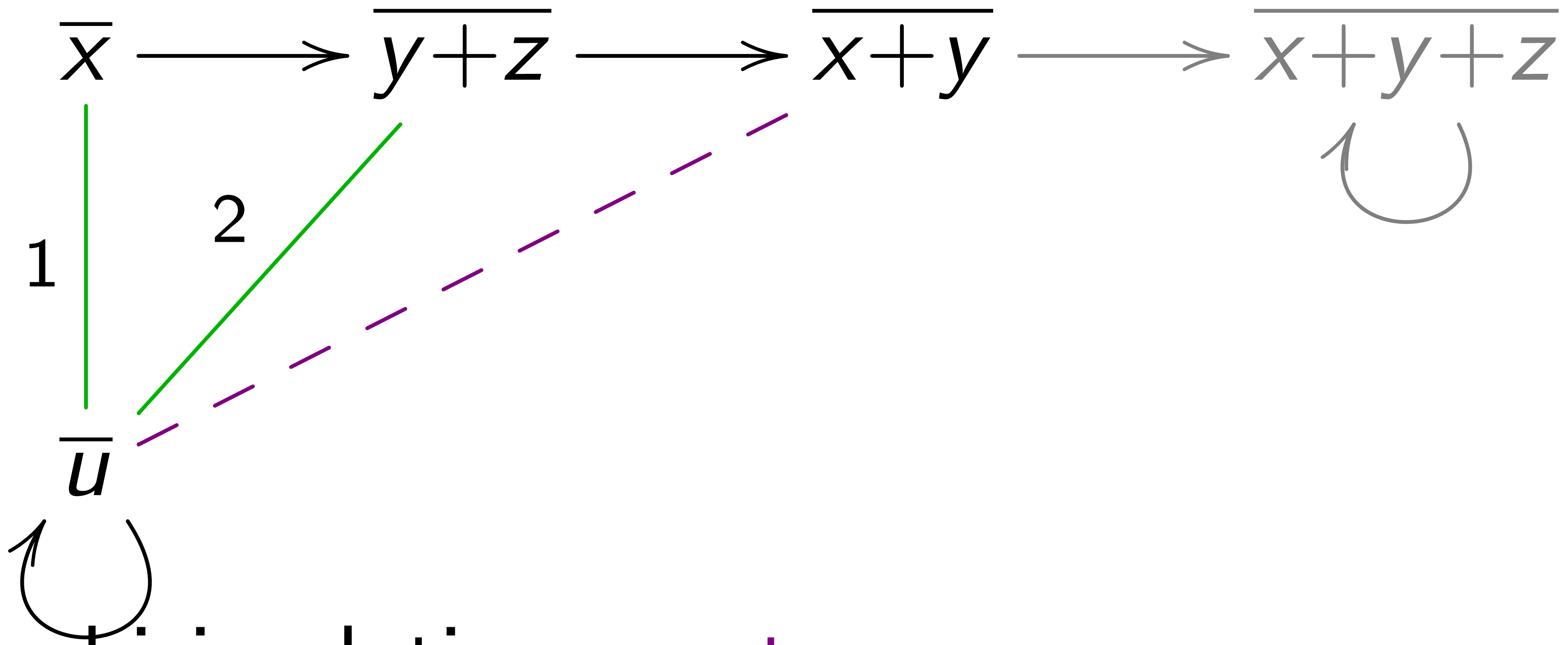


Non-Deterministic Automata

One can do even better:



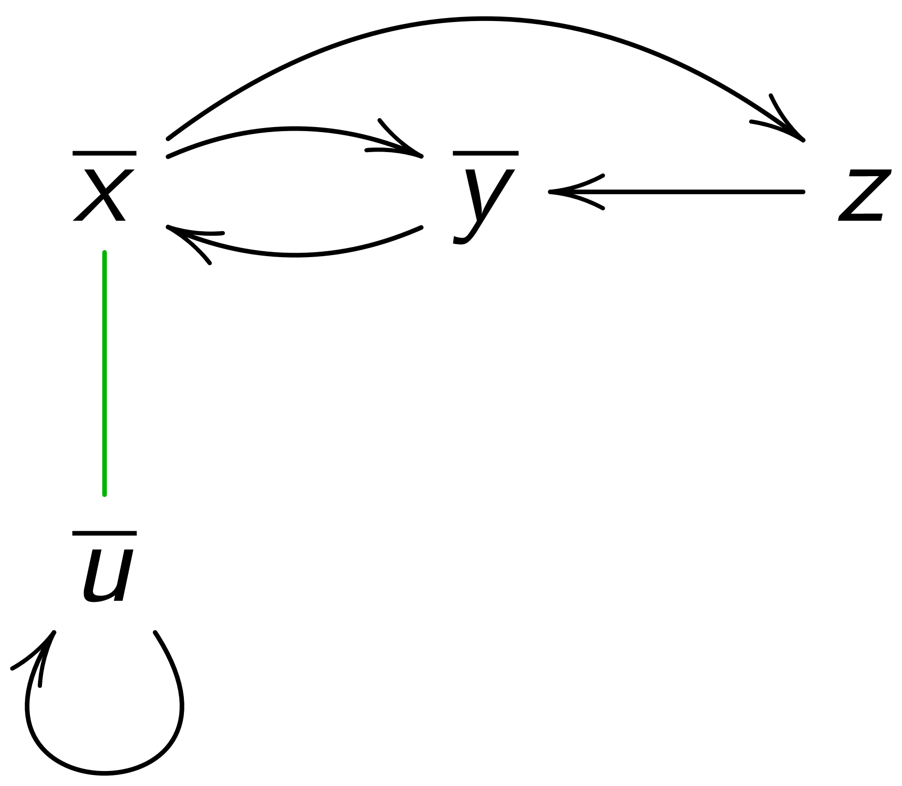
$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z &&\\
&= u && (2)
\end{aligned}
$$



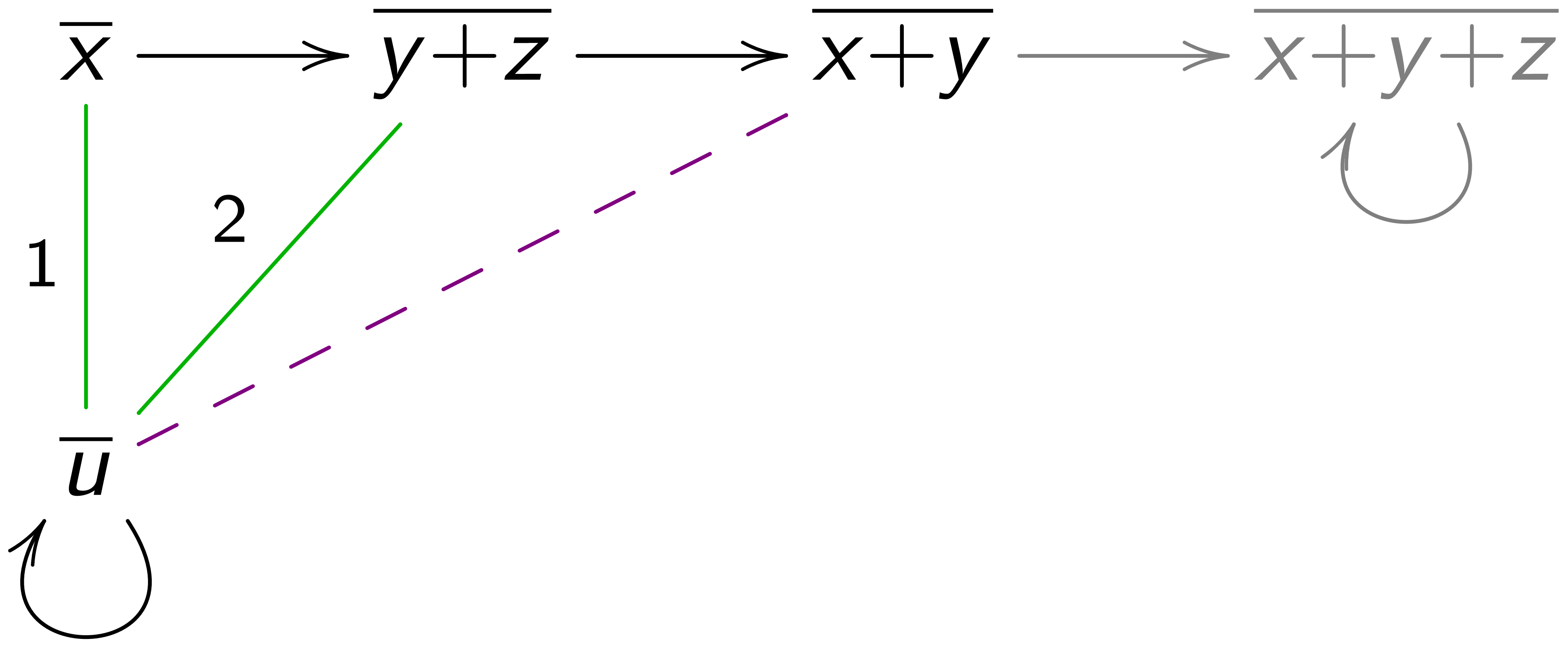
using bisimulations up to congruence

Non-Deterministic Automata

One can do even better:



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



this yield to the HKC algorithm [Bonchi, Pous'13]

Intermezzo: conduction up-to with coalgebra

Next time

Lecture 3

Equivalence via axioms
Completeness