Coalgebras for Concurrency

– or –

A bridge between automata and concurrency theory.

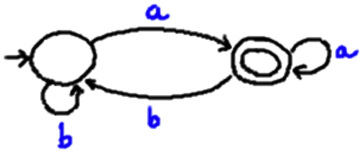
Alexandra Silva

Radboud University Nijmegen
Centrum Wiskunde & Informatica

September 6, 2014
TRENDS 2014
Rome, Italy

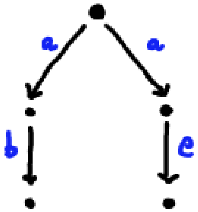
Context

- Automata are basic structures in Computer Science.
- Language equivalence: well-studied, several algorithms.
- Renewed attention (POPL'11, '13, '14).



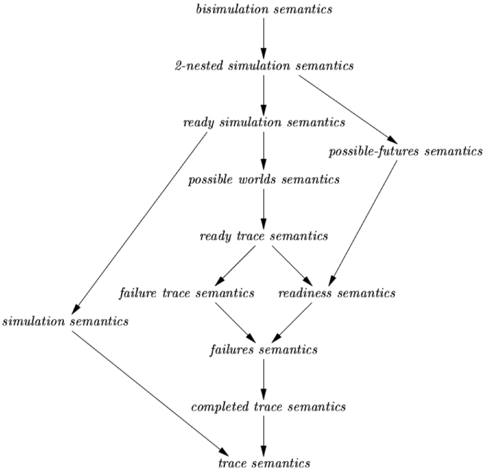
Context

- Concurrency: a spectrum of equivalences.
- Checking usually done by reducing to bisimilarity.



An alternative road

- Many efficient algorithms for equivalence of automata.
- Applications in concurrency?



From automata to concurrency

Various spectrum equivalences

=

Language equivalence of a *transformed* system

=

Automaton with outputs and structured state space (Moore automata).

Bonsangue, Bonchi, Caltais, Rutten, S. MFPS 12

From automata to concurrency

- Generalization of existing algorithms to Moore automata.
- Brzozowski's and Hopcroft/Karp algorithms for van Glabbeek's spectrum.
- Cleaveland and Hennessy's acceptance graphs for must/may testing = Moore automata.
- Brzozowski's and Hopcroft/Karp algorithms algorithm for must/may testing.

Bonchi, Caltais, Pous, Silva. APLAS 2013

From automata to concurrency

- Generalization of existing algorithms to Moore automata.
- Brzozowski's and Hopcroft/Karp algorithms for van Glabbeek's spectrum.
- Cleaveland and Hennessy's acceptance graphs for must/may testing = Moore automata.
- Brzozowski's and Hopcroft/Karp algorithms algorithm for must/may testing.

Bonchi, Caltais, Pous, Silva. APLAS 2013

The approach

Roadmap

1. Brief introduction to coalgebra.
2. Two algorithms for language equivalence and generalizations.
3. Trends and opportunities.

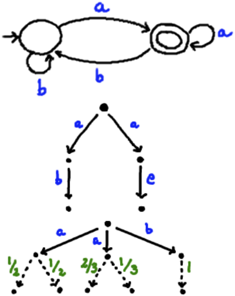
(Co)algebra

Specify and reason about systems.

(Co)algebra

Specify and reason about systems.

state-machines
e.g. DFA, LTS, PA, . . .



(Co)algebra

Specify and **reason** about **systems**.

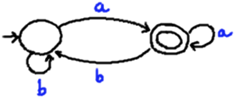
Syntax
RE, CCS, . . .

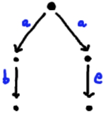
$b^*a(b^*a)^*$

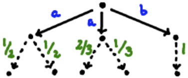
$a.b.0 + a.c.0$

$a.(\frac{1}{2}\cdot 0 \oplus \frac{1}{2}\cdot 0) + \cdots$

state-machines
e.g. DFA, LTS, PA, . . .

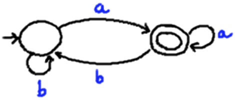
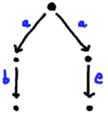
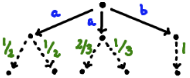






(Co)algebra

Specify and **reason** about **systems**.

Specify	reason	systems
Syntax RE, CCS, . . .	Axiomatization KA,. . .	state-machines e.g. DFA, LTS, PA, . . .
$b^*a(b^*a)^*$	$1 + aa^* = a^*$ ⋮	
$a.b.0 + a.c.0$	$P + 0 = P$ ⋮	
$a.(\frac{1}{2} \cdot 0 \oplus \frac{1}{2} \cdot 0) + \cdots$	$p.P \oplus p'.P = (p+p').P$ ⋮	

(Co)algebra

Specify and **reason** about **systems**.

Syntax
RE, CCS, . . .

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

$a.b.0 + a.c.0$

$a.(\frac{1}{2}\cdot 0 \oplus \frac{1}{2}\cdot 0) + \cdots$

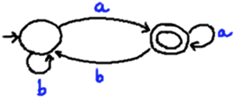
Axiomatization
KA,. . .

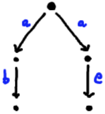
$1 + a\,a^* = a^*$
⋮

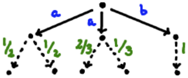
$P + 0 = P$
⋮

$p.P \oplus p'.P = (p+p').P$
⋮

state-machines
e.g. DFA, LTS, PA, . . .

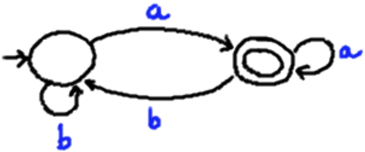




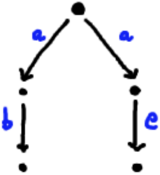


Can we do all of this uniformly in a single framework?

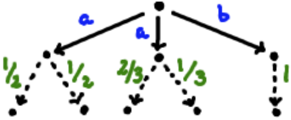
What do this things have in common?



$(S, t : S \to 2 \times S^A)$



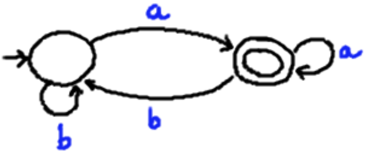
$(S, t : S \to \mathcal{P}S^A)$



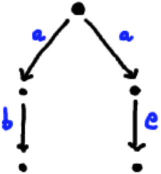
$(S, t : S \to \mathcal{P}\mathcal{D}_\omega(S)^A)$

$(S, t : S \to TS)$ T-coalgebras

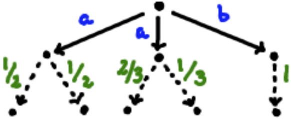
What do this things have in common?



$(S, t : S \to 2 \times S^A)$



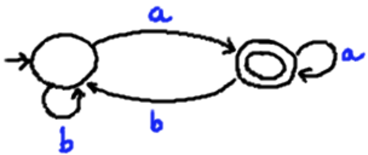
$(S, t : S \to \mathcal{P}S^A)$



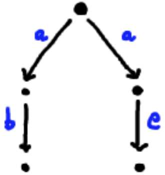
$(S, t : S \to \mathcal{P}\mathcal{D}_\omega(S)^A)$

$(S, t : S \to TS)$ T-coalgebras

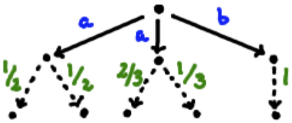
What do this things have in common?



$$(S, t : S \to 2 \times S^A)$$



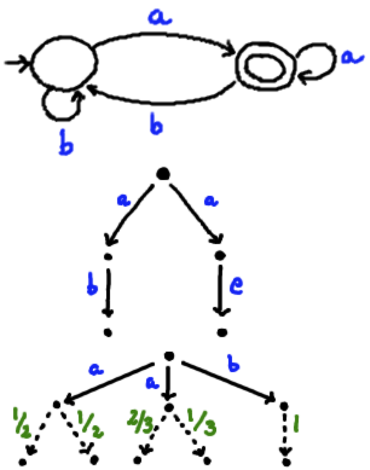
$$(S, t : S \to \mathcal{P} S^A)$$



$$(S, t : S \to \mathcal{P}\mathcal{D}_\omega(S)^A)$$

$(S, t : S \to TS)$ T-coalgebras

What do this things have in common?



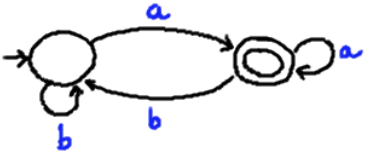
$(S, t : S \to 2 \times S^A)$

$(S, t : S \to \mathcal{P}S^A)$

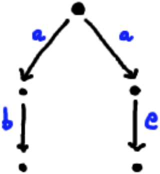
$(S, t : S \to \mathcal{P}\mathcal{D}_\omega(S)^A)$

$(S, t : S \to TS)$ T-coalgebras

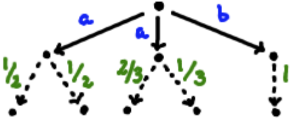
What do this things have in common?



$$(S, t : S \to 2 \times S^A)$$



$$(S, t : S \to \mathcal{P}S^A)$$



$$(S, t : S \to \mathcal{P}\mathcal{D}_\omega(S)^A)$$

$(S, t : S \to TS)$ T-coalgebras

The power of T

$$(S, t : S \to TS)$$

The functor T determines:

1. notion of observational equivalence (coalg. bisimulation)
   E.g. $T = 2 \times (-)^A$: language equivalence
2. behaviour (final coalgebra)
   E.g. $T = 2 \times (-)^A$: languages over A – 2^{A^*}
3. set of expressions describing finite systems
4. axioms to prove bisimulation equivalence of expressions

The power of T

$$(S, t : S \to TS)$$

The functor T determines:

1. notion of observational equivalence (coalg. bisimulation)
   E.g. $T = 2 \times (-)^A$: language equivalence
2. behaviour (final coalgebra)
   E.g. $T = 2 \times (-)^A$: languages over A – 2^{A^*}
3. set of expressions describing finite systems
4. axioms to prove bisimulation equivalence of expressions

The power of T

$$(S, t : S \to TS)$$

The functor T determines:

1. notion of observational equivalence (coalg. bisimulation)
   E.g. $T = 2 \times (-)^A$: language equivalence
2. behaviour (final coalgebra)
   E.g. $T = 2 \times (-)^A$: languages over A – 2^{A^*}
3. set of expressions describing finite systems
4. axioms to prove bisimulation equivalence of expressions

The power of T

$$(S, t : S \to TS)$$

The functor T determines:

1. notion of observational equivalence (coalg. bisimulation)
   E.g. $T = 2 \times (-)^A$: language equivalence
2. behaviour (final coalgebra)
   E.g. $T = 2 \times (-)^A$: languages over A – 2^{A^*}
3. set of expressions describing finite systems
4. axioms to prove bisimulation equivalence of expressions

1 + 2 are classic coalgebra; 3 + 4 are recent work.

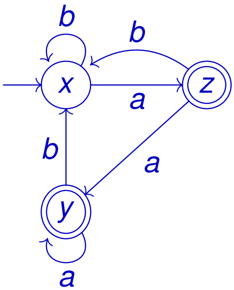
Current state of affairs

- Coalgebra/coinduction – semantic side of the world: operational/denotational semantics, logics, . . .
- Key role in current development of functional languages, type theory, . . .
- This talk: uniform derivation of algorithms and applications to concurrency.

Brzozowski's algorithm

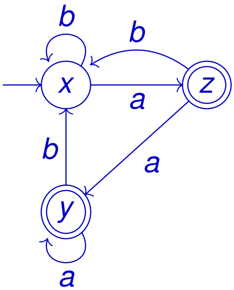
Brzozowski's algorithm, (co)algebraically – Kozen's festschrift 2012

Brzozowski's algorithm (by example)



- initial state: x
- final states: y and z
- $L(x) = \{a, b\}^* a$
- X is reachable but not minimal: $L(y) = \varepsilon + \{a, b\}^* a = L(z)$

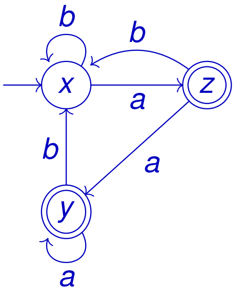
Brzozowski's algorithm (by example)



- initial state: x
- final states: y and z
- $L(x) = \{a, b\}^* a$
- X is reachable but not minimal: $L(y) = \varepsilon + \{a, b\}^* a = L(z)$

Reversing the automaton: *rev*(*X*)

$X =$

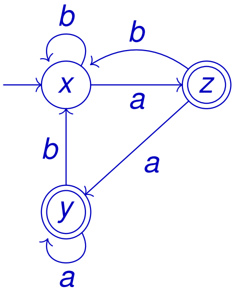


$rev(X) =$

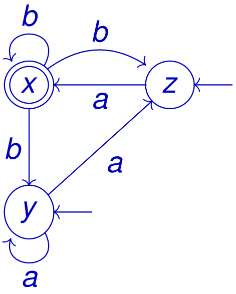
- transitions are reversed
- initial states $\Leftrightarrow$ final states
- *rev*(*X*) is non-deterministic

Reversing the automaton: $rev(X)$

$X =$



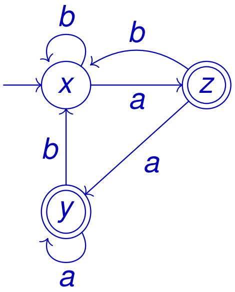
$rev(X) =$



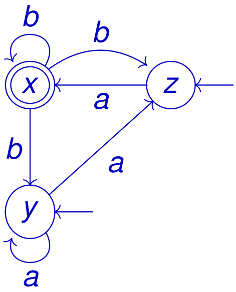
- transitions are reversed
- initial states $\Leftrightarrow$ final states
- $rev(X)$ is non-deterministic

Reversing the automaton: $rev(X)$

$X =$

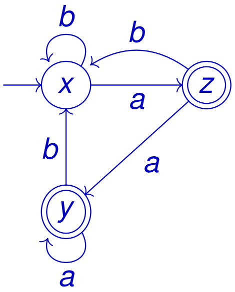


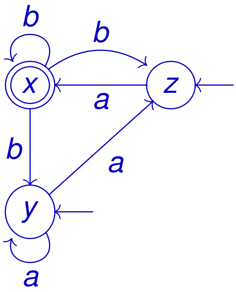
$rev(X) =$



- transitions are reversed
- initial states $\Leftrightarrow$ final states
- $rev(X)$ is non-deterministic

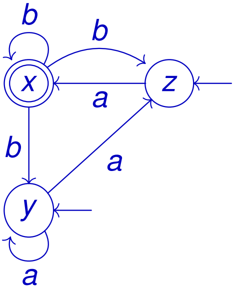
Reversing the automaton: $rev(X)$

$X =$ 

$rev(X) =$ 

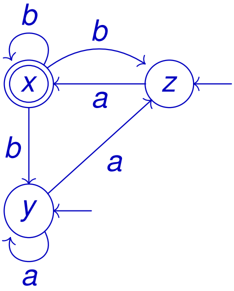
- transitions are reversed
- initial states $\Leftrightarrow$ final states
- $rev(X)$ is non-deterministic

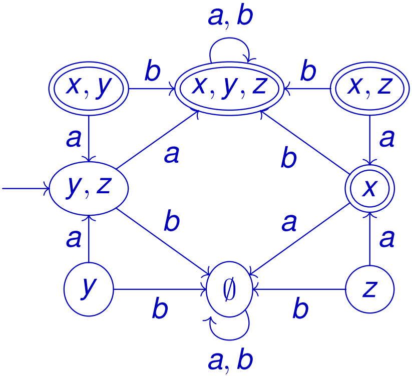
Making it deterministic again: $det(rev(X))$



- new state space: $2^X = \{V \mid V \subseteq \{x, y, z\}\}$
- initial state: $\{y, z\}$ final states: all V with $x \in V$
- $V \xrightarrow{a} W$ $\quad W = \{w \mid v \xrightarrow{a} w, v \in V\}$

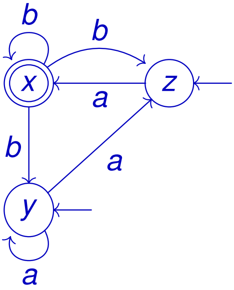
Making it deterministic again: $det(rev(X))$

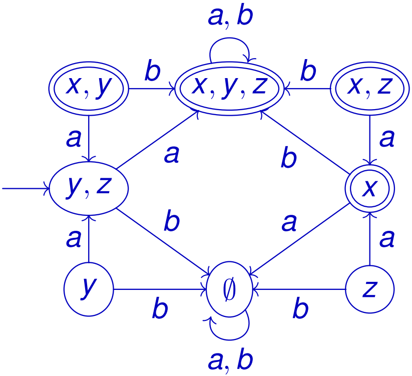




- new state space: $2^X = \{V \mid V \subseteq \{x, y, z\}\}$
- initial state: $\{y, z\}$ final states: all V with $x \in V$
- $V \xrightarrow{a} W$ $W = \{w \mid v \xrightarrow{a} w, v \in V\}$

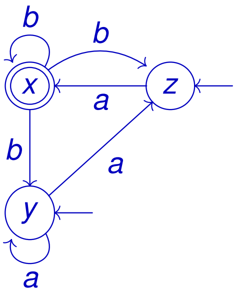
Making it deterministic again: $det(rev(X))$

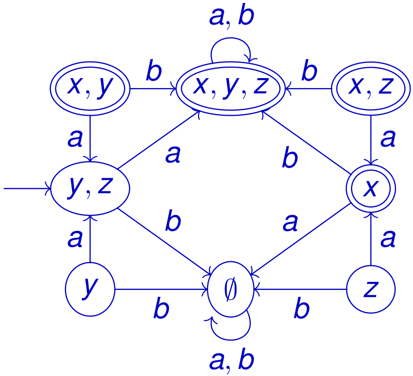




- new state space: $2^X = \{ V \mid V \subseteq \{x, y, z\} \}$
- initial state: $\{y, z\}$ final states: all V with $x \in V$
- $V \xrightarrow{a} W$ $\quad W = \{ w \mid v \xrightarrow{a} w, v \in V \}$

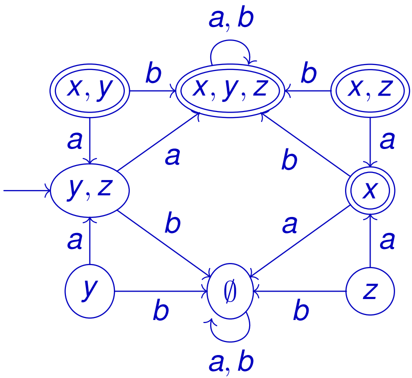
Making it deterministic again: *det*(*rev*(*X*))





- new state space: $2^X = \{ V \mid V \subseteq \{x, y, z\} \}$
- initial state: $\{y, z\}$ final states: all V with $x \in V$
- $V \xrightarrow{a} W \qquad W = \{ w \mid v \xrightarrow{a} w \, , v \in V \}$

The automaton $det(rev(X))$ . . .

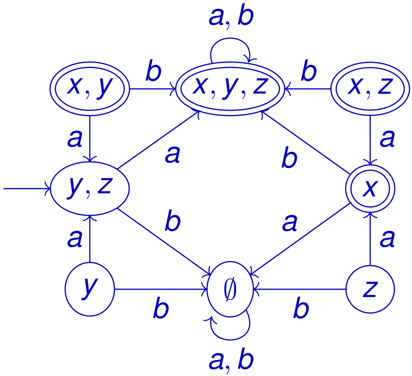


- . . . accepts the reverse of the language accepted by X:

$$L(\,det(rev(X))\,) \;=\; a\{a,b\}^* \;=\; reverse(\,L(X)\,)$$

- . . . and is observable!

The automaton $det(rev(X))$ . . .

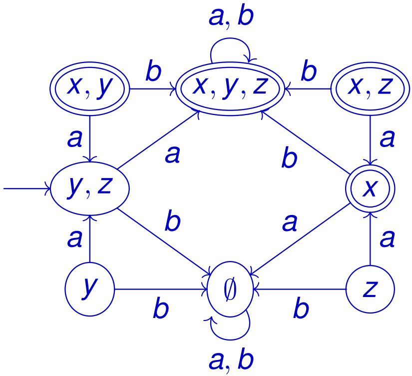


- . . . accepts the reverse of the language accepted by X:

$$L(\, det(rev(X))\,) \;=\; a\{a, b\}^* \;=\; reverse(\, L(X)\,)$$

- . . . and is observable!

The automaton $det(rev(X))$ . . .



- . . . accepts the reverse of the language accepted by X:

$$L(\, det(rev(X))\,) \;=\; a\{a, b\}^* \;=\; reverse(\, L(X)\,)$$

- . . . and is observable!

Brzozowski's Theorem

If: a deterministic automaton X is *reachable* and accepts $L(X)$

then: $det(rev(X))$ is *minimal* and

$$L(\,det(rev(X))\,) = reverse(\,L(X)\,)$$

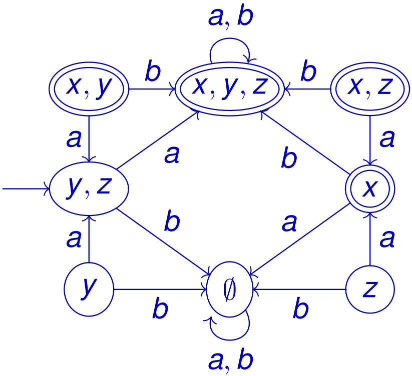
Brzozowski's Theorem

If: a deterministic automaton X is *reachable* and accepts $L(X)$

then: $det(rev(X))$ is *minimal* and

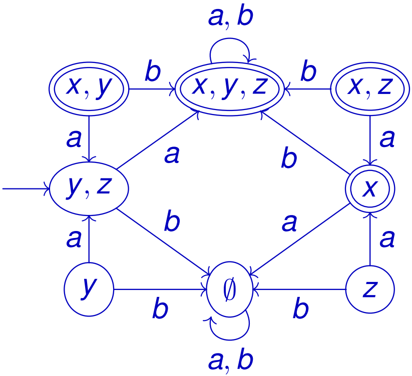
$$L(\,det(rev(X))\,) \;=\; reverse(\,L(X)\,)$$

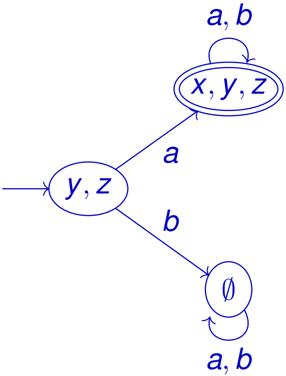
Taking the reachable part of *det*(*rev*(*X*))



- $reach(det(rev(X)))$

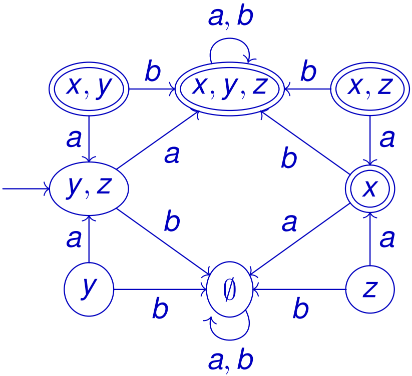
Taking the reachable part of $det(rev(X))$

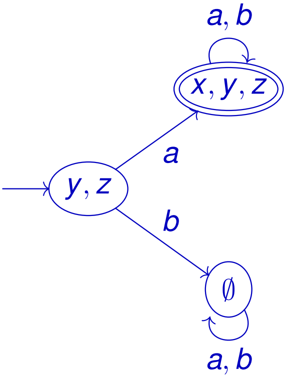




- $reach(det(rev(X)))$

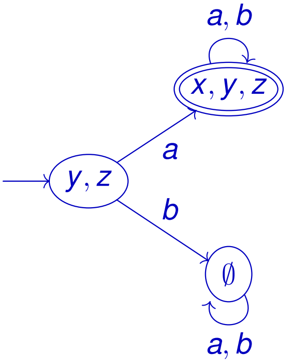
Taking the reachable part of $det(rev(X))$





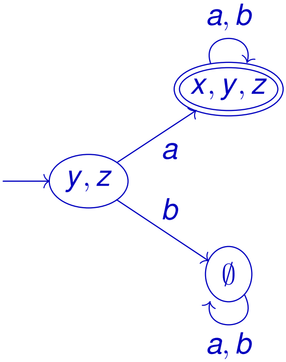
- $reach(det(rev(X)))$ is reachable (by construction)

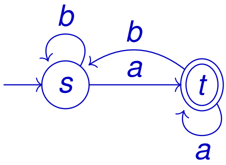
Repeating everything, now for *reach*(*det*(*rev*(*X*)))



- . . . gives us $reach(det(rev(reach(det(rev(X))))))$
- which is (reachable and) minimal and accepts $\{a, b\}^* a$.

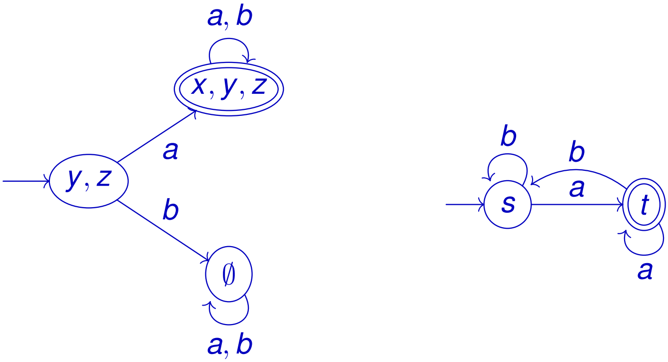
Repeating everything, now for *reach*(*det*(*rev*(*X*)))





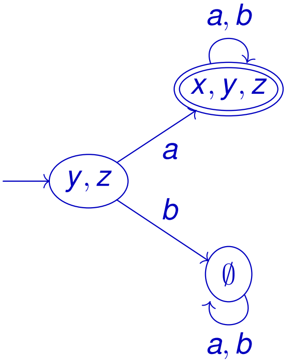
- . . . gives us $reach(det(rev(reach(det(rev(X))))))$
- which is (reachable and) minimal and accepts $\{a, b\}^* a$.

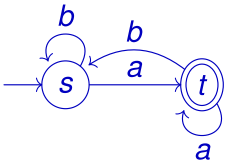
Repeating everything, now for *reach*(*det*(*rev*(*X*)))



- ... gives us *reach*(*det*(*rev*(*reach*(*det*(*rev*(*X*))))))
- which is (reachable and) minimal and accepts $\{a, b\}^* a$.

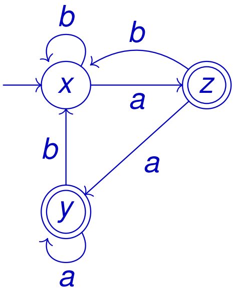
Repeating everything, now for *reach*(*det*(*rev*(*X*)))





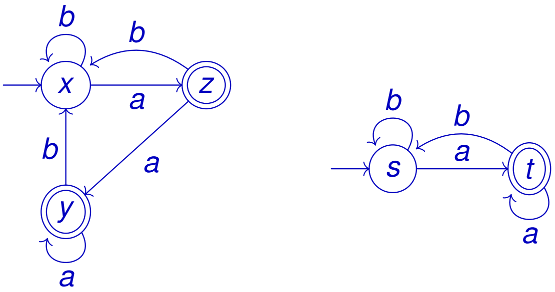
- . . . gives us *reach*(*det*(*rev*(*reach*(*det*(*rev*(*X*))))))
- which is (reachable and) minimal and accepts $\{a, b\}^* a$.

All in all: Brzozowski's algorithm



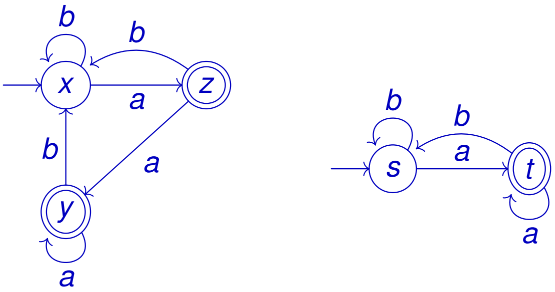
- X is reachable and accepts $\{a, b\}^* a$
- $reach(det(rev(reach(det(rev(X))))))$ also accepts $\{a, b\}^* a$
- ... and is minimal!!

All in all: Brzozowski's algorithm



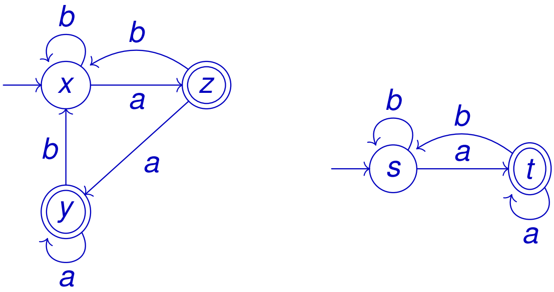
- X is reachable and accepts $\{a, b\}^* a$
- $reach(det(rev(reach(det(rev(X))))))$ also accepts $\{a, b\}^* a$
- . . . and is minimal!!

All in all: Brzozowski's algorithm



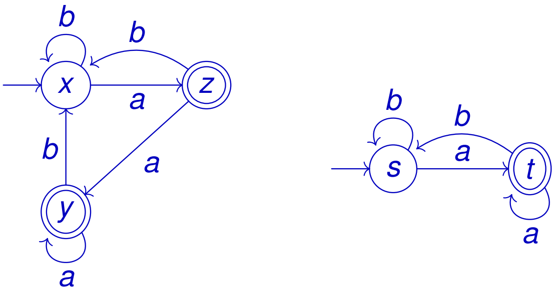
- X is reachable and accepts $\{a, b\}^*\, a$
- $reach(det(rev(reach(det(rev(X))))))$ also accepts $\{a, b\}^*\, a$
- . . . and is minimal!!

All in all: Brzozowski's algorithm



- X is reachable and accepts $\{a, b\}^* a$
- $reach(det(rev(reach(det(rev(X))))))$ also accepts $\{a, b\}^* a$
- ... and is minimal!!

All in all: Brzozowski's algorithm



- X is reachable and accepts $\{a, b\}^* \; a$
- $reach(det(rev(reach(det(rev(X))))))$ also accepts $\{a, b\}^* \; a$
- . . . and is minimal!!

Beyond deterministic automata

```
Brzozowski (X)
(1) reverse and determinize;
(2) take the reachable part;
(3) reverse and determinize;
(4) take the reachable part.
```

Checking language equivalence

Minimize both automata and check for isomorphism.

Crucial observation for generalizations

Reverse and determinize is more general than at first sight!

Beyond deterministic automata

```
Brzozowski (X)
(1) reverse and determinize;
(2) take the reachable part;
(3) reverse and determinize;
(4) take the reachable part.
```

Checking language equivalence

Minimize both automata and check for isomorphism.

Crucial observation for generalizations

Reverse and determinize is more general than at first sight!

Beyond deterministic automata

```
Brzozowski (X)
(1) reverse and determinize;
(2) take the reachable part;
(3) reverse and determinize;
(4) take the reachable part.
```

Checking language equivalence

Minimize both automata and check for isomorphism.

Crucial observation for generalizations

Reverse and determinize is more general than at first sight!

Reverse and determinize

$$2^{(-)} : \quad \begin{array}{c} V \\ g \big\downarrow \\ W \end{array} \quad \mapsto \quad \begin{array}{c} 2^V \\ \big\uparrow 2^g \\ 2^W \end{array}$$

where $2^V = \{S \mid S \subseteq V\}$ and, for all $S \subseteq W$,

$$2^g(S) = g^{-1}(S) \quad (= \{v \in V \mid g(v) \in S\})$$

- This construction is *contravariant* and ......
- Works for general $B^{(-)}$ and ......
- For structured sets (change in category).

Reverse and determinize

$$2^{(-)} \; : \quad \begin{array}{c} V \\ g \downarrow \\ W \end{array} \quad \mapsto \quad \begin{array}{c} 2^V \\ \uparrow 2^g \\ 2^W \end{array}$$

where $2^V = \{S \mid S \subseteq V\}$ and, for all $S \subseteq W$,

$$2^g(S) = g^{-1}(S) \quad (= \{v \in V \mid g(v) \in S\})$$

- This construction is *contravariant* and ......
- Works for general $B^{(-)}$ and ......
- For structured sets (change in category).

Reverse and determinize

$$2^{(-)} : \quad \begin{array}{c} V \\ g \big\downarrow \\ W \end{array} \quad \mapsto \quad \begin{array}{c} 2^V \\ \big\uparrow 2^g \\ 2^W \end{array}$$

where $2^V = \{S \mid S \subseteq V\}$ and, for all $S \subseteq W$,

$$2^g(S) = g^{-1}(S) \quad (= \{v \in V \mid g(v) \in S\})$$

- This construction is *contravariant* and . . . ...
- Works for general $B^{(-)}$ and . . . ...
- For structured sets (change in category).

Brzozowski's algorithm generalized

Deterministic automata	$X \to B \times X^A$	$\mathcal{P}(A^*)$

Brzozowski's algorithm generalized

Deterministic automata	$X \to B \times X^A$	$\mathcal{P}(A^*)$
Moore automata	$X \to B \times X^A$	B^{A^*}
Linear weighted automata	$V \to \mathbb{R} \times V^A$	$\mathbb{R}^{A^*}$
Guarded strings automata	$\mathcal{B} \to \mathbb{B} \times \mathcal{B}^{\mathbb{B} \times A}$	$\mathcal{P}((\mathtt{At} \cdot A)^* \cdot \mathtt{At})$
$\vdots$	$\vdots$	

Correctness and generalizations in [BBRS'12, BBHPRS'13].

Brzozowski's algorithm in concurrency

Cleaveland and Hennessy's acceptance graphs for must/may testing = Moore automata.

Several equivalences of the spectrum (failure, ready-trace , . . .) = regular behaviors Moore automata.

See APLAS paper for details.

Brzozowski's algorithm in concurrency

Cleaveland and Hennessy's acceptance graphs for must/may testing = Moore automata.

Several equivalences of the spectrum (failure, ready-trace , . . .) = regular behaviors Moore automata.

See APLAS paper for details.

Intermezzo

- Brzozowski's algorithm can be uniformly generalized based on the type functor.

Intermezzo

- Brzozowski's algorithm can be uniformly generalized based on the type functor.
- Second example: up-to algorithm (HKC).

Up-to techniques

Tools and proof techniques for systems equivalence

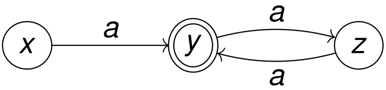
Methodology:

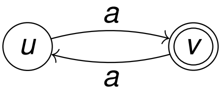
1. characterise coinductively a given notion of equivalence
2. improve the associated proof method

up-to techniques

Deterministic finite automata

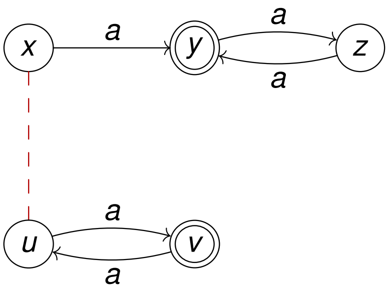
The states x and u are language equivalent





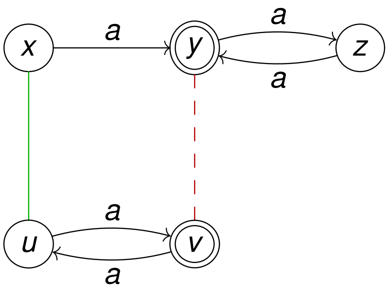
Deterministic finite automata

The states x and u are language equivalent



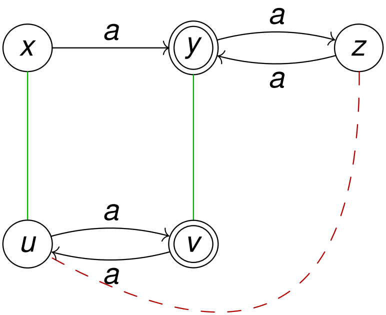
Deterministic finite automata

The states x and u are language equivalent



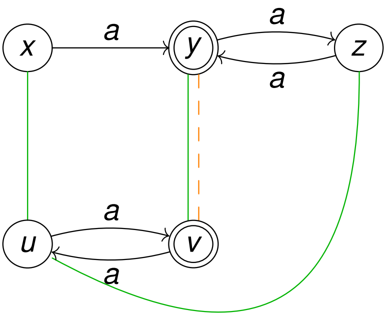
Deterministic finite automata

The states x and u are language equivalent



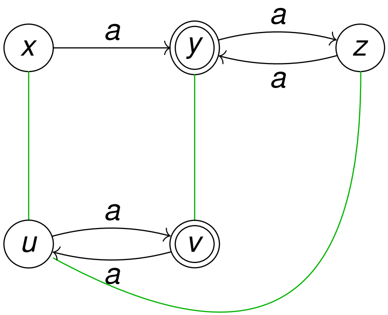
Deterministic finite automata

The states x and u are language equivalent



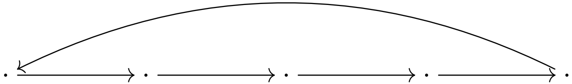
Deterministic finite automata

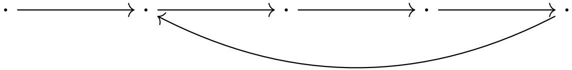
The states x and u are language equivalent



Complexity

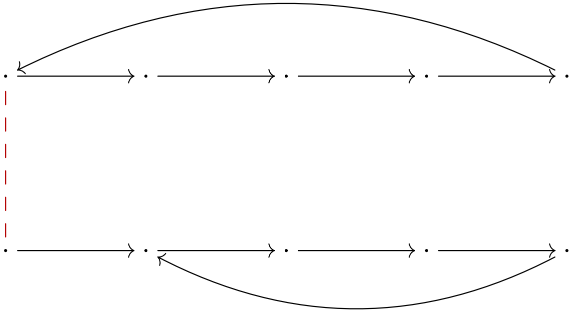
The previous algorithm is *quadratic*





Complexity

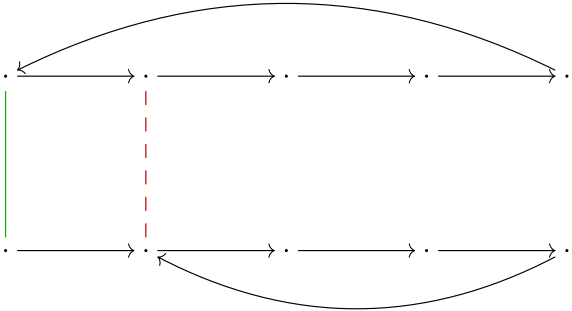
The previous algorithm is *quadratic*



3 pairs

Complexity

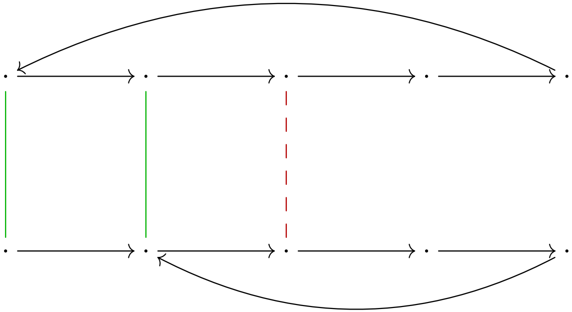
The previous algorithm is *quadratic*



4 pairs

Complexity

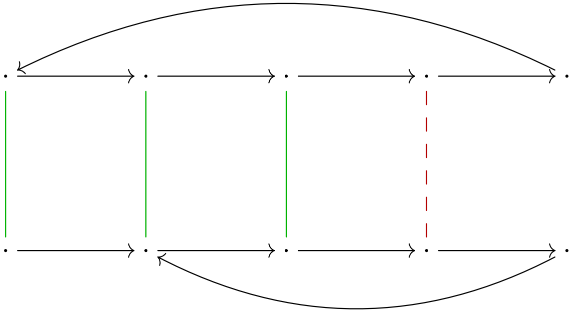
The previous algorithm is *quadratic*



5 pairs

Complexity

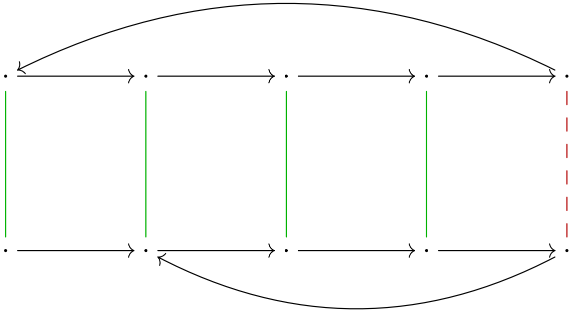
The previous algorithm is *quadratic*



6 pairs

Complexity

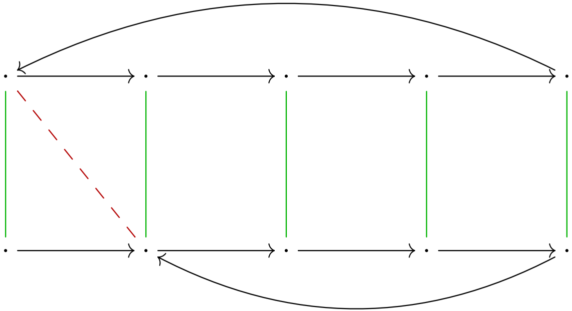
The previous algorithm is *quadratic*



7 pairs

Complexity

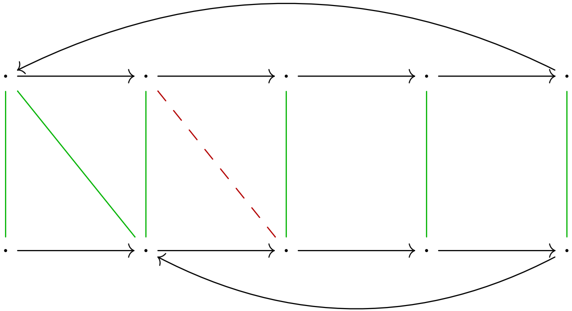
The previous algorithm is *quadratic*



8 pairs

Complexity

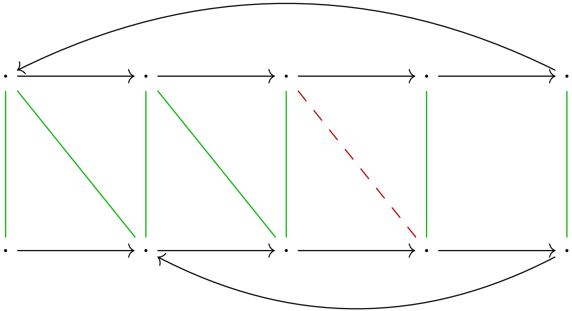
The previous algorithm is *quadratic*



9 pairs

Complexity

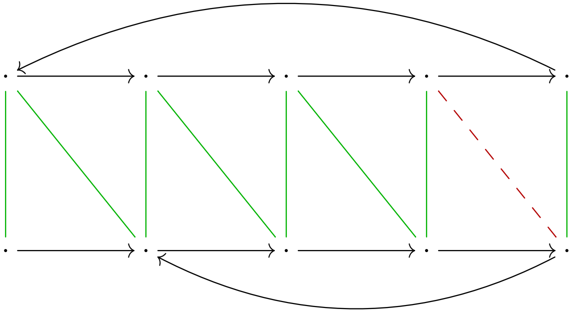
The previous algorithm is *quadratic*



10 pairs

Complexity

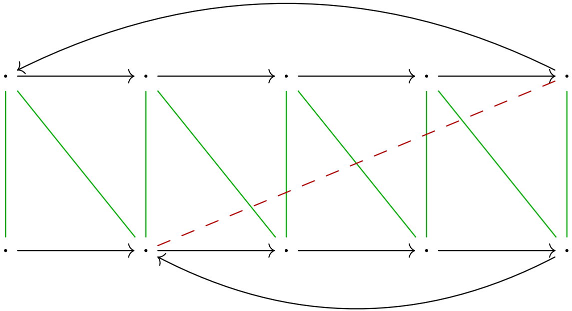
The previous algorithm is *quadratic*



11 pairs

Complexity

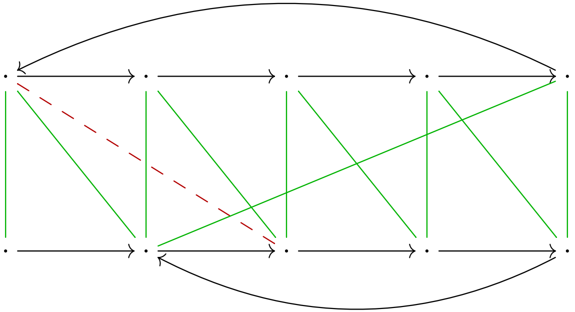
The previous algorithm is *quadratic*



12 pairs

Complexity

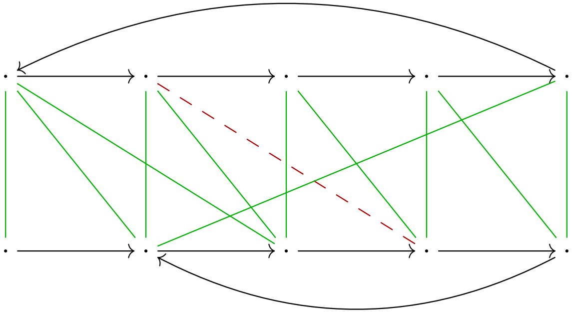
The previous algorithm is *quadratic*



13 pairs

Complexity

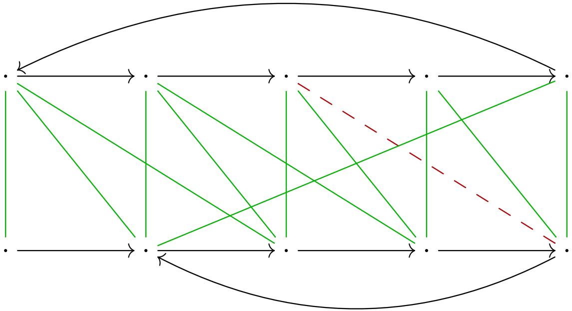
The previous algorithm is *quadratic*



14 pairs

Complexity

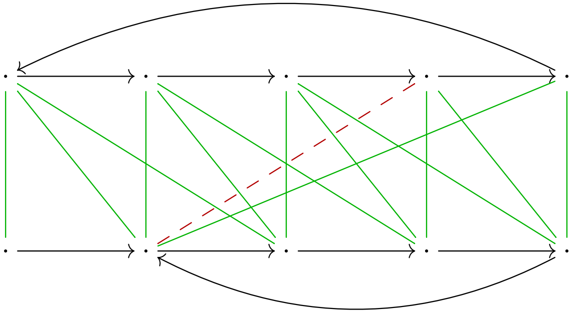
The previous algorithm is *quadratic*



15 pairs

Complexity

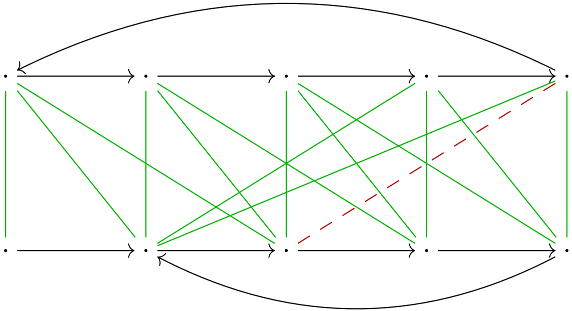
The previous algorithm is *quadratic*



16 pairs

Complexity

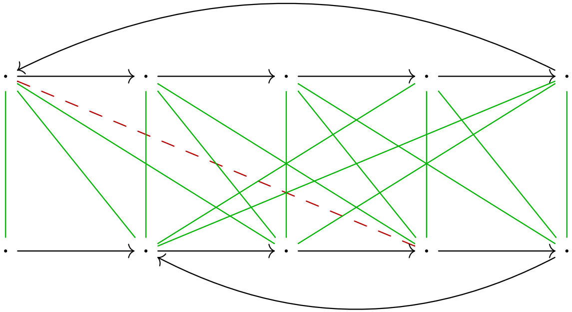
The previous algorithm is *quadratic*



17 pairs

Complexity

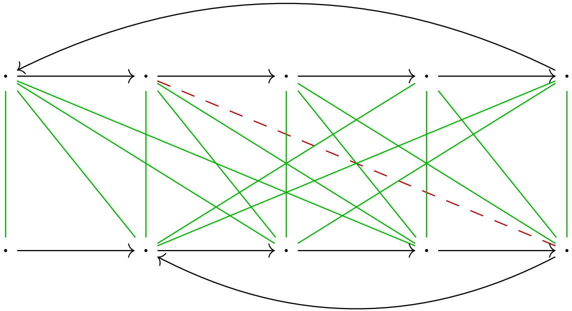
The previous algorithm is *quadratic*



18 pairs

Complexity

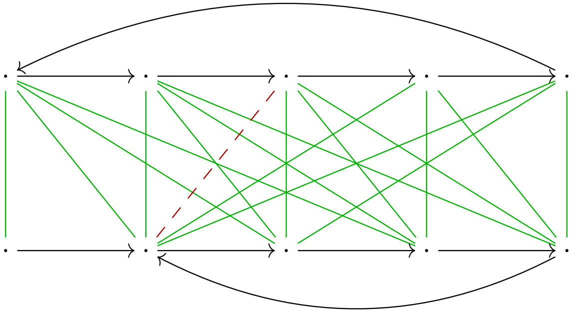
The previous algorithm is *quadratic*



19 pairs

Complexity

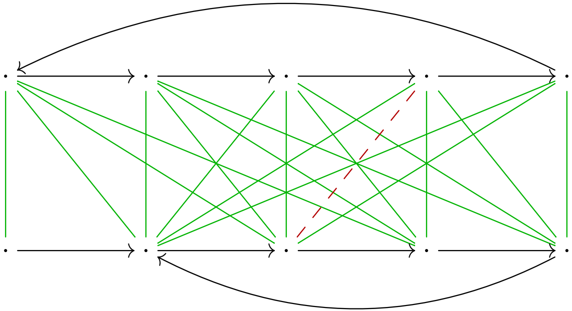
The previous algorithm is *quadratic*



20 pairs

Complexity

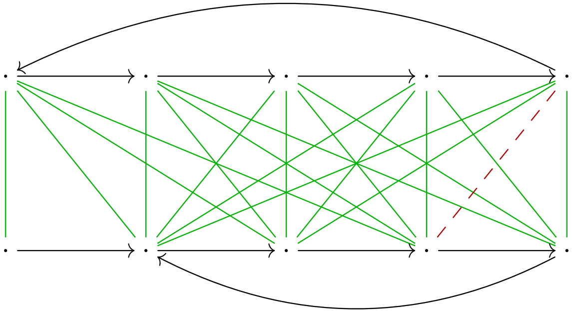
The previous algorithm is *quadratic*



21 pairs

Complexity

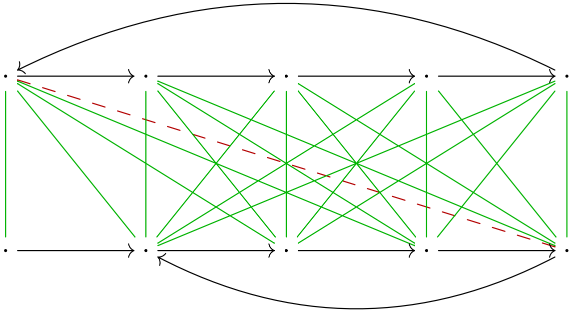
The previous algorithm is *quadratic*



22 pairs

Complexity

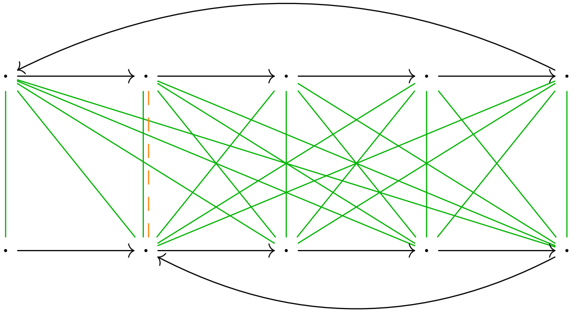
The previous algorithm is *quadratic*



23 pairs

Complexity

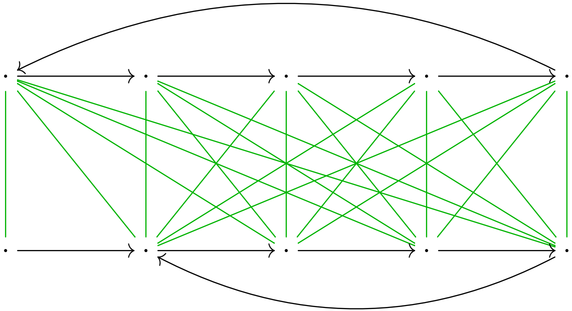
The previous algorithm is *quadratic*



21 pairs

Complexity

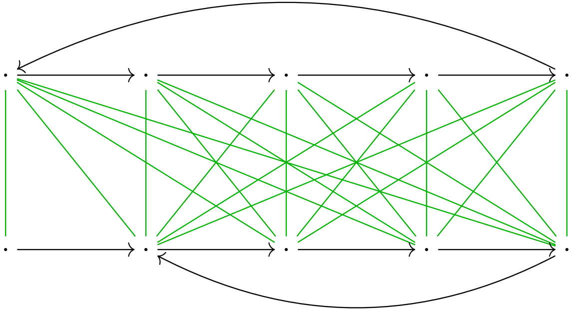
The previous algorithm is *quadratic*



21 pairs

First improvement

One can stop much earlier

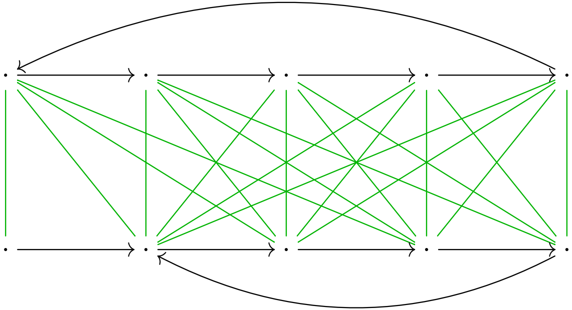


21 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

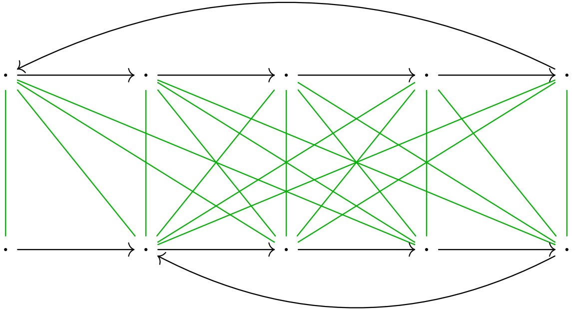


~~21~~ 20 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

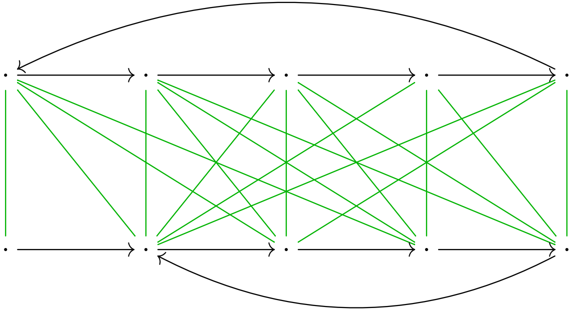


~~21~~ 19 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

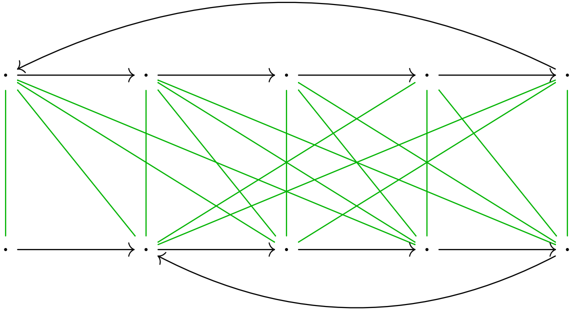


~~21~~ 18 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

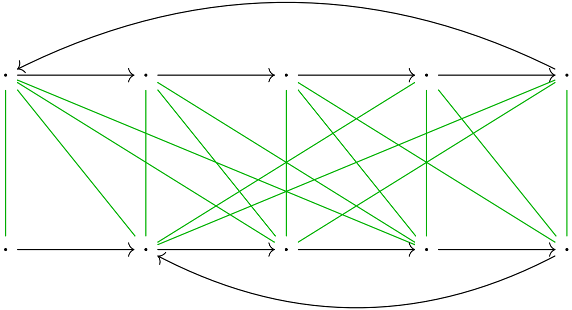


~~21~~ 17 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

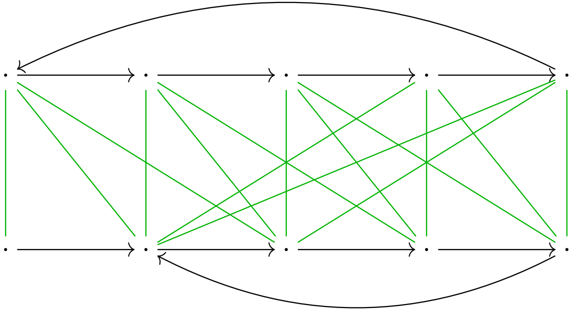


~~21~~ 16 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

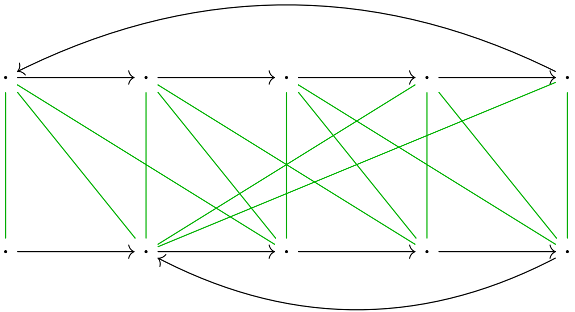


~~21~~ 15 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

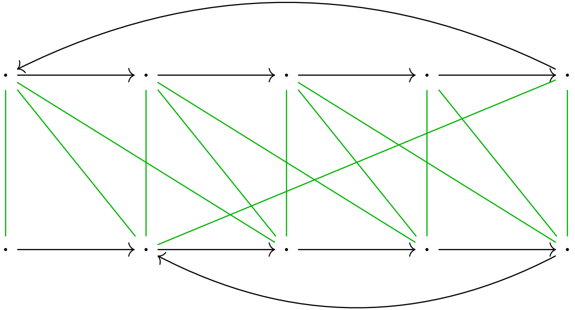


~~21~~ 14 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

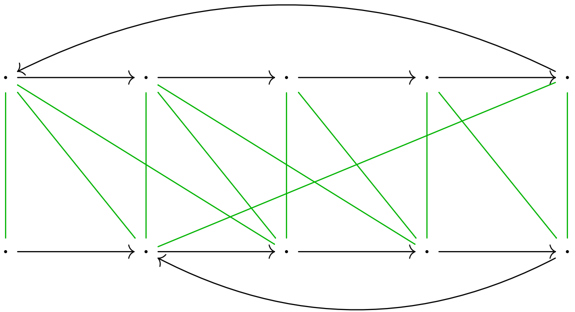


~~21~~ 13 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

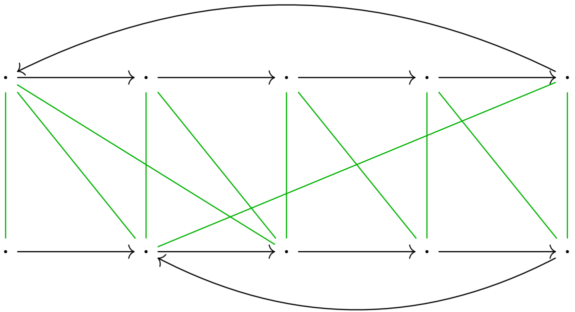


~~21~~ 12 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

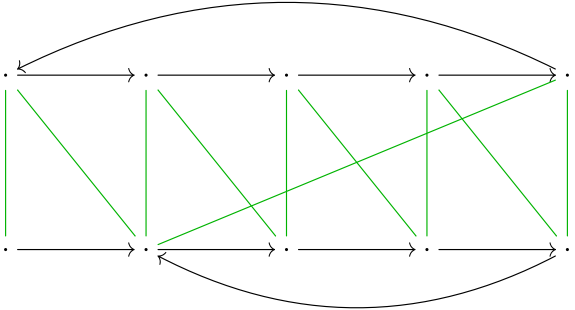


~~21~~ 11 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

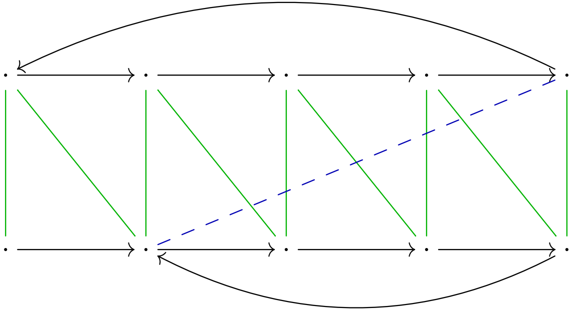


~~21~~ 10 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier

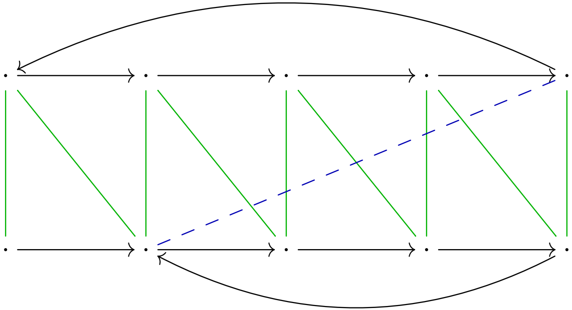


~~21~~ 9 pairs

Complexity: almost linear [Tarjan '75]

First improvement

One can stop much earlier



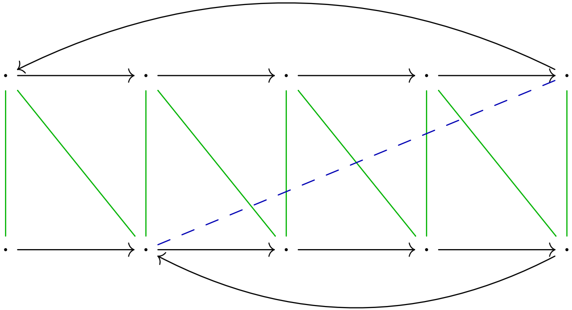
[Hopcroft and Karp '71]

Complexity: almost linear

[Tarjan '75]

First improvement

One can stop much earlier

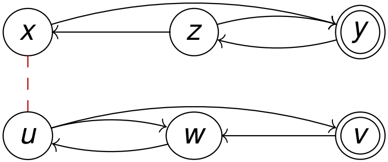


[Hopcroft and Karp '71]
[Tarjan '75]

Complexity: almost linear

Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

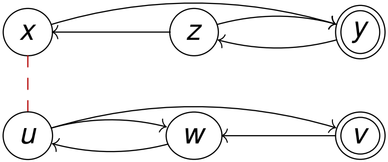
$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$

x y z $x+y$ $y+z$ $x+y+z$

u $v+w$ $u+w$ $u+v+w$

Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

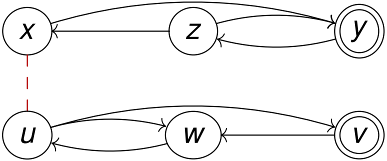
$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$

x y z $x+y$ $y+z$ $x+y+z$

u $v+w$ $u+w$ $u+v+w$

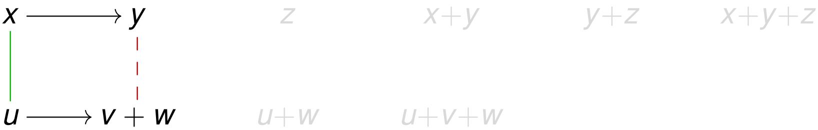
Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



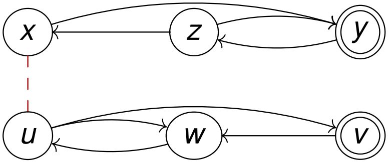
$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$



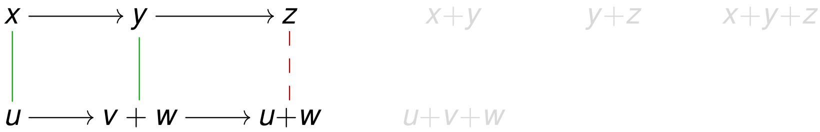
Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



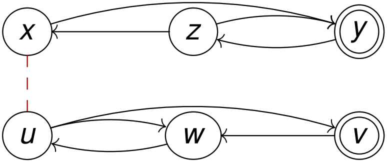
$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$



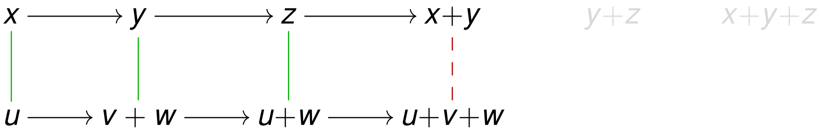
Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



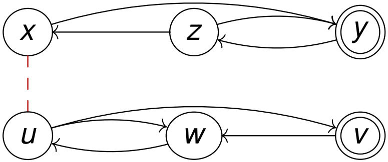
$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$



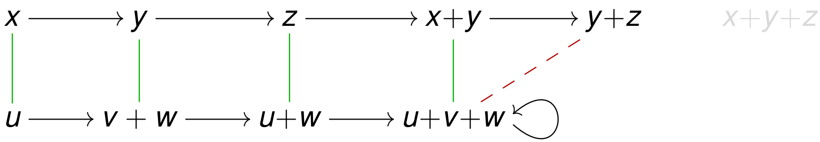
Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



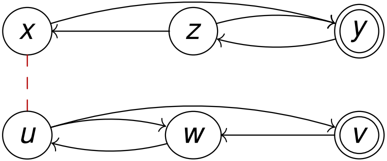
$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$



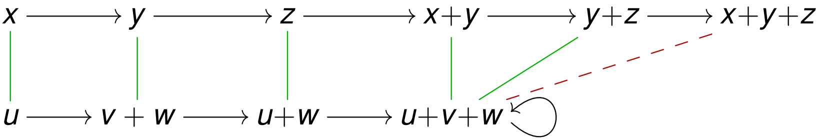
Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



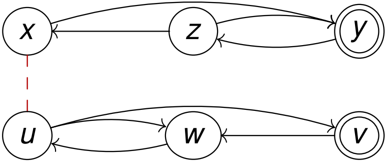
$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$



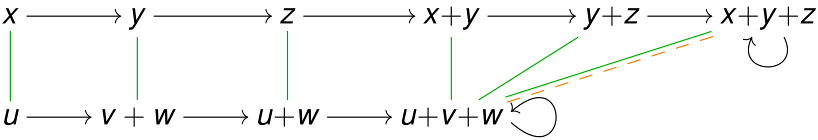
Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



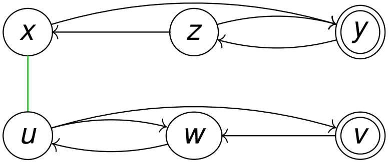
$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$



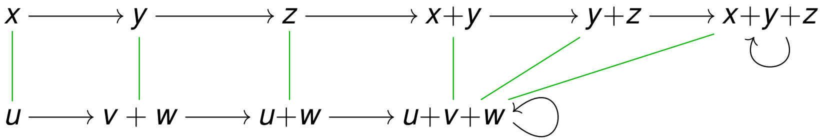
Non-Deterministic Automata

Hopcroft and Karp *on the fly*, with powerset construction:



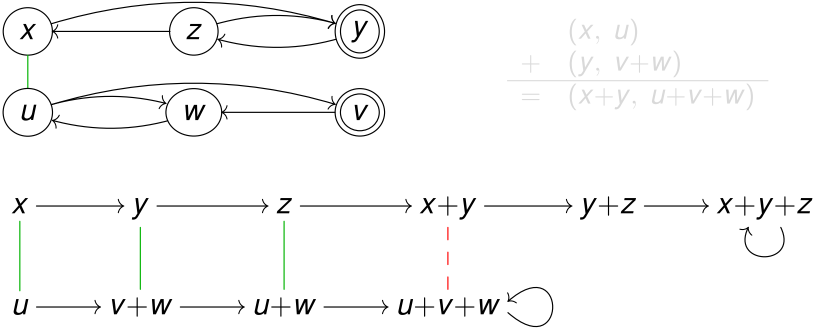
$$o^\sharp(S) = \bigvee_{s \in S} o(s)$$

$$t^\sharp(S)(a) = \bigcup_{s \in S} t(s)(a)$$



Non-Deterministic Automata

One can do better:

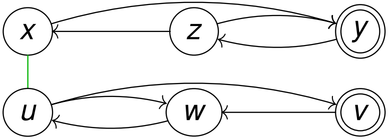


$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$

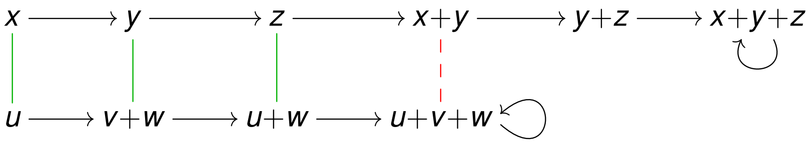
using bisimulations *up to union*

Non-Deterministic Automata

One can do better:



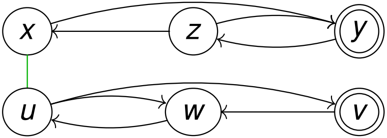
$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$



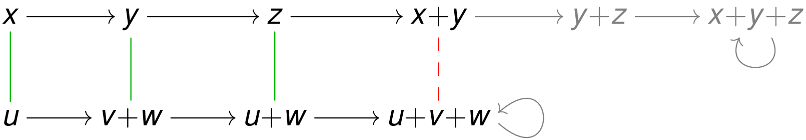
using bisimulations *up to union*

Non-Deterministic Automata

One can do better:



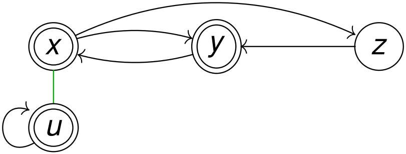
$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$



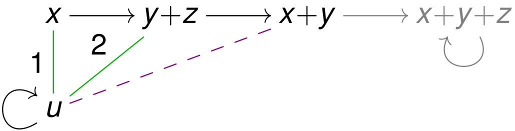
using bisimulations *up to union*

Non-Deterministic Automata

One can do even better:



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

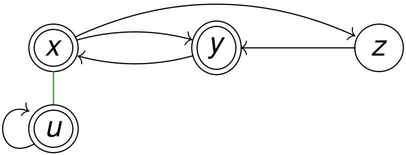


using bisimulations *up to congruence*

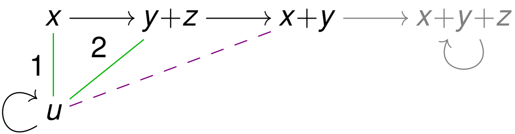
this lead to the HKC algorithm [Bonchi, Pous, POPL'13]

Non-Deterministic Automata

One can do even better:



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

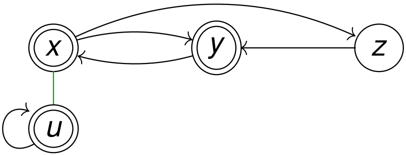


using bisimulations *up to congruence*

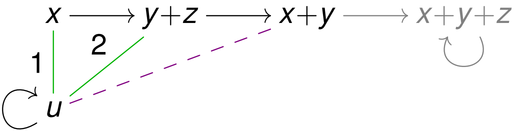
this lead to the HKC algorithm [Bonchi, Pous, POPL'13]

Non-Deterministic Automata

One can do even better:



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

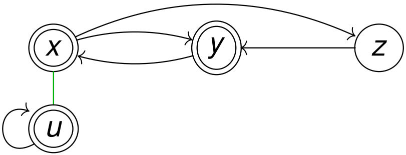


using bisimulations *up to congruence*

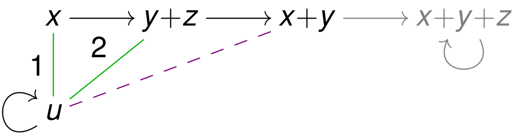
this lead to the HKC algorithm [Bonchi, Pous, POPL'13]

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*

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

HKC is also parametric

```
HKC(X,Y):
(1) R is empty; todo is {(X',Y')};
(2) while todo is not empty, do
 (2.1) extract (X',Y') from todo;
 (2.2) if (X',Y') ∈ c(R ∪ todo) then continue;
 (2.3) if o♯(X') ≠ o♯(Y') then return false;
 (2.4) for all a ∈ A,
           insert (t♯(X')(a), t♯(Y')(a)) in todo;
 (2.5) insert (X',Y') in R;
(3) return true;
```

Powerset construction $o^\sharp$, $t^\sharp$

Generalized to other algebraic structures / functors (weighted, Moore, probabilistic automata, . . .)

Applicable for must/may testing, failure, . . .

HKC is also parametric

```
HKC(X,Y):
(1) R is empty; todo is {(X',Y')};
(2) while todo is not empty, do
 (2.1) extract (X',Y') from todo;
 (2.2) if (X',Y') ∈ c(R ∪ todo) then continue;
 (2.3) if o♯(X') ≠ o♯(Y') then return false;
 (2.4) for all a ∈ A,
          insert (t♯(X')(a), t♯(Y')(a)) in todo;
 (2.5) insert (X',Y') in R;
(3) return true;
```

Powerset construction $o^\sharp$, $t^\sharp$

Generalized to other algebraic structures / functors (weighted, Moore, probabilistic automata, . . .)

Applicable for must/may testing, failure, . . .

Trends / opportunities

Trend I: New language constructs

Trend II: NetKat – applications in networks

Trend III: Automata learning

Trend I : New language constructs

- Extensions of programming languages with coinductive constructs (Agda, CoCaml, ...).
- Algorithms like general HKC enable efficient representation and equivalence check.

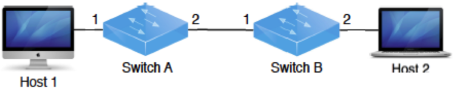
Opportunity for concurrency

- New methods to check equivalence of behaviors.
- Automatic derivation of programming constructs for new models.

Trend II: NetKAT – semantic foundations for networks

Anderson, Foster, Guha, Jeannin, Kozen, Schlesinger, Walker, POPL'14

- Specifying and reasoning about networks.
- Based on Kleene algebra with tests (KAT).



Recent work (submitted)

- Coinductive model of KAT extended to NetKAT.
- Brzozowski and HKC for NetKAT.

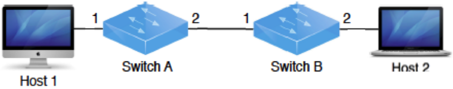
Opportunity for concurrency

- Foundations of networks: transference of results, new challenges.

Trend II: NetKAT – semantic foundations for networks

Anderson, Foster, Guha, Jeannin, Kozen, Schlesinger, Walker, POPL'14

- Specifying and reasoning about networks.
- Based on Kleene algebra with tests (KAT).



Recent work (submitted)

- Coinductive model of KAT extended to NetKAT.
- Brzozowski and HKC for NetKAT.

Opportunity for concurrency

- Foundations of networks: transference of results, new challenges.

Trend III : automata learning

- Angluin's algorithm: inference of regular languages.
- Coalgebra enables generalizations to e.g. weighted automata.

Opportunity for concurrency

- Inference of behaviors in distributed systems.
- Applications in security.

Conclusions

- Coalgebra has applications in automata and concurrency.
- Bridge to transfer results and tools.
- (Co)algebra is not only semantics but also algorithms!

Thanks! Questions?

Conclusions

- Coalgebra has applications in automata and concurrency.
- Bridge to transfer results and tools.
- (Co)algebra is not only semantics but also algorithms!

Thanks! Questions?

Conclusions

- Coalgebra has applications in automata and concurrency.
- Bridge to transfer results and tools.
- (Co)algebra is not only semantics but also algorithms!

Thanks! Questions?