Coalgebraic Up-to Techniques

Alexandra Silva

(joint with Bonsangue, Bonchi, Pous, Rot & Rutten)

Radboud University Nijmegen & CWI Amsterdam

Shonan Meeting 026
07.10.2013

(slide credits: Damien Pous)

Context

Tools and proof techniques for systems equivalence

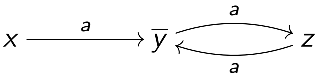
Methodology:

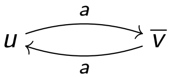
1. characterise coinductively a given notion of equivalence
2. improve the associated proof method

up-to techniques

Deterministic finite automata

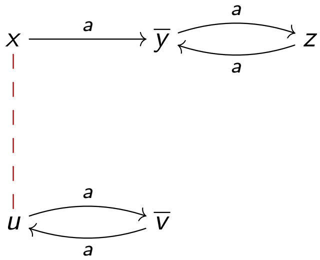
The states x and u are language equivalent





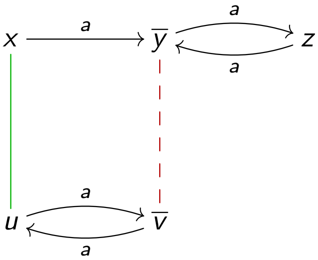
Deterministic finite automata

The states x and u are language equivalent



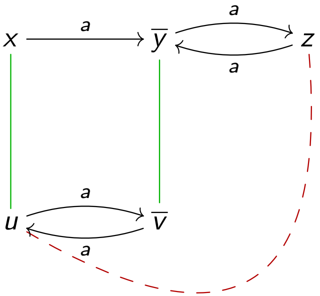
Deterministic finite automata

The states x and u are language equivalent



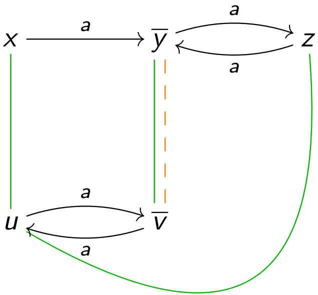
Deterministic finite automata

The states x and u are language equivalent



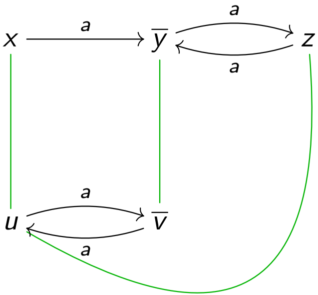
Deterministic finite automata

The states x and u are language equivalent



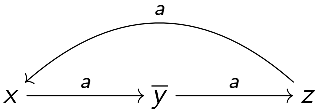
Deterministic finite automata

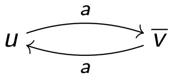
The states x and u are language equivalent



Deterministic finite automata

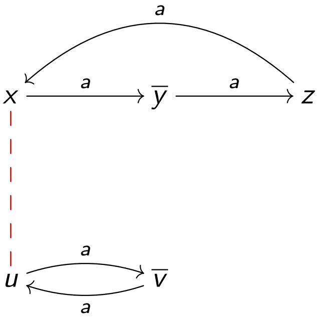
x and u are **not** equivalent





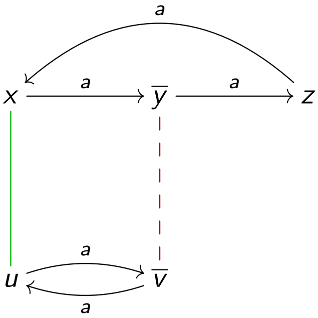
Deterministic finite automata

x and u are **not** equivalent



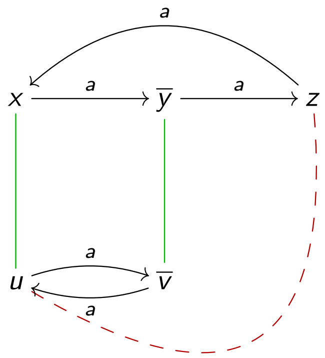
Deterministic finite automata

x and u are **not** equivalent



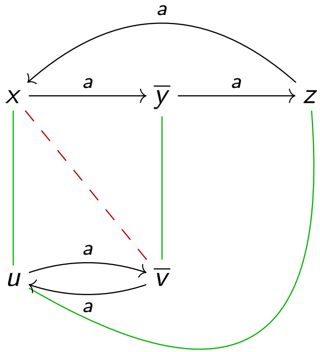
Deterministic finite automata

x and u are **not** equivalent



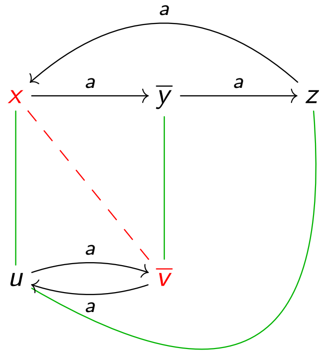
Deterministic finite automata

x and u are **not** equivalent



Deterministic finite automata

x and u are **not** equivalent



Correctness

- A relation R is a 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 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 $x \; R \; y$

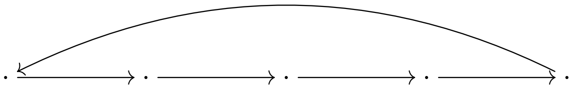
Correctness

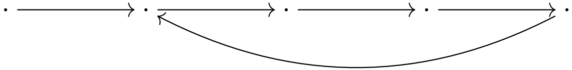
- A relation R is a bisimulation if $x \mathrel{R} y$ entails
  - $o(x) = o(y)$;
  - for all a, $t_a(x) \mathrel{R} t_a(y)$.
- *Theorem:* $L(x) = L(y)$ iff
  there exists a bisimulation R with $x \mathrel{R} y$

The previous algorithm attempts to construct a bisimulation

Complexity

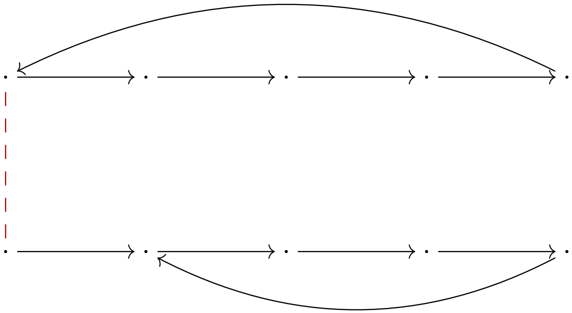
The previous algorithm is quadratic





Complexity

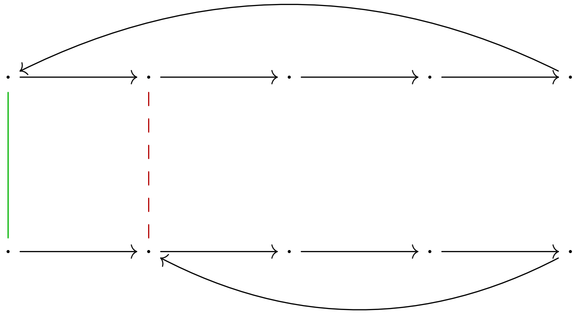
The previous algorithm is quadratic



0 pairs

Complexity

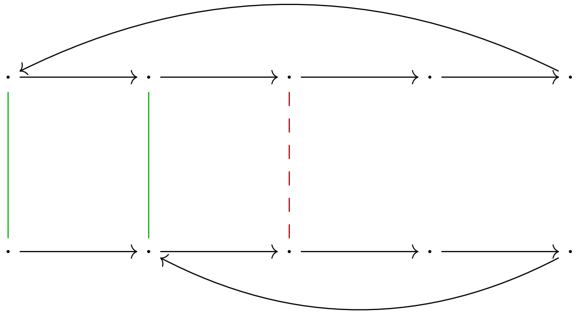
The previous algorithm is quadratic



1 pairs

Complexity

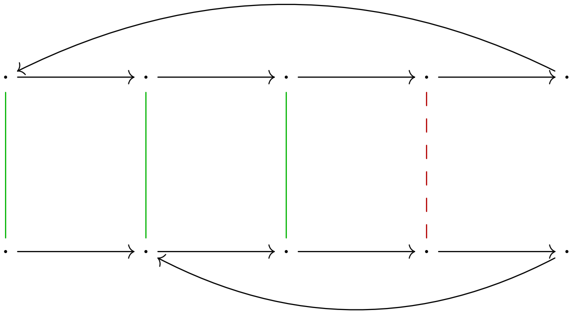
The previous algorithm is quadratic



2 pairs

Complexity

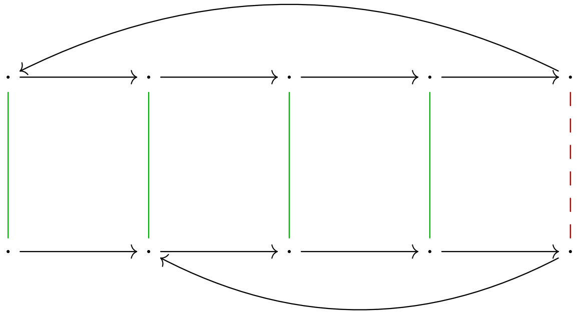
The previous algorithm is quadratic



3 pairs

Complexity

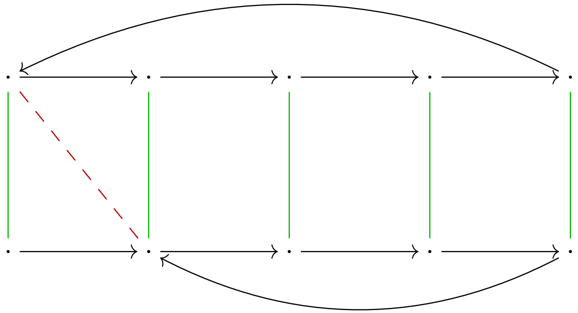
The previous algorithm is quadratic



4 pairs

Complexity

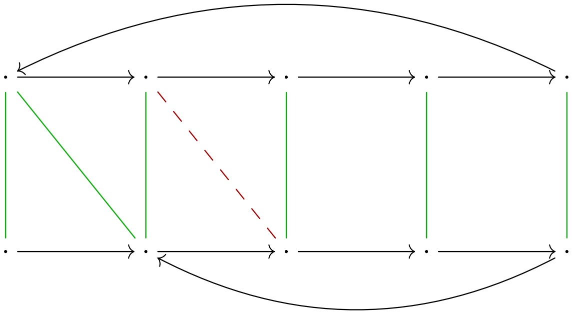
The previous algorithm is quadratic



5 pairs

Complexity

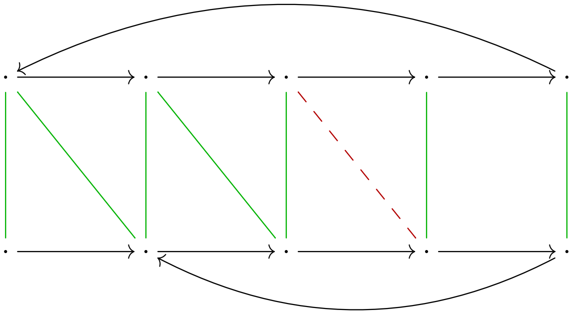
The previous algorithm is quadratic



6 pairs

Complexity

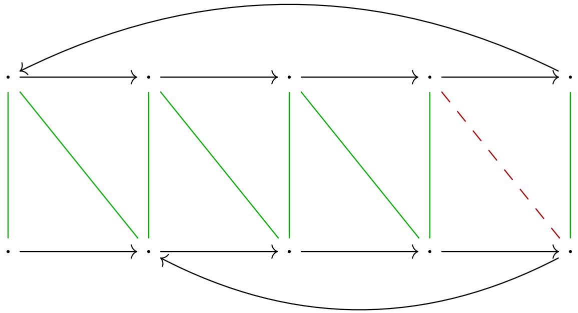
The previous algorithm is quadratic



7 pairs

Complexity

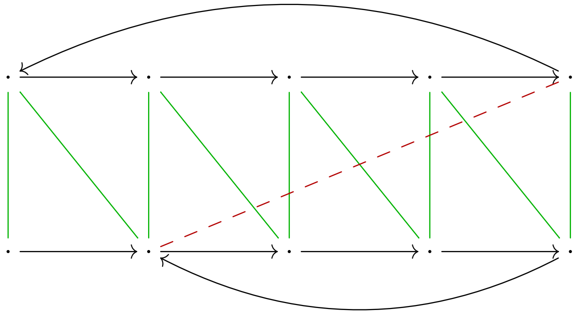
The previous algorithm is quadratic



8 pairs

Complexity

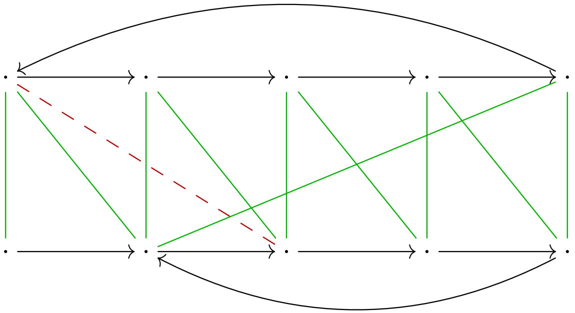
The previous algorithm is quadratic



9 pairs

Complexity

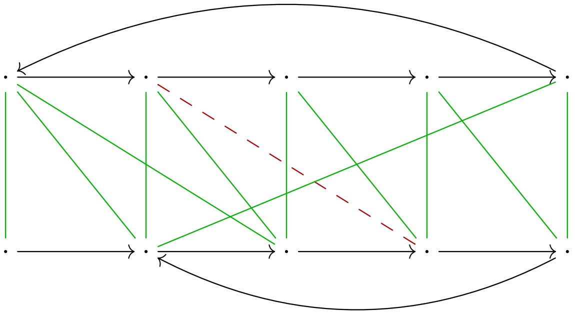
The previous algorithm is quadratic



10 pairs

Complexity

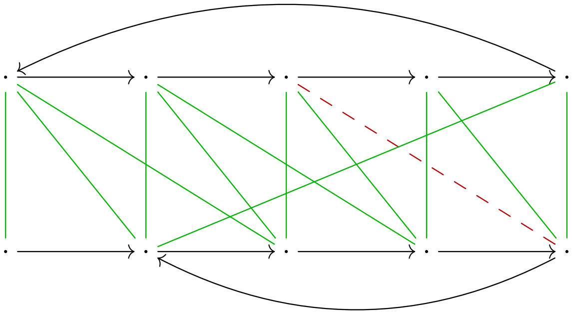
The previous algorithm is quadratic



11 pairs

Complexity

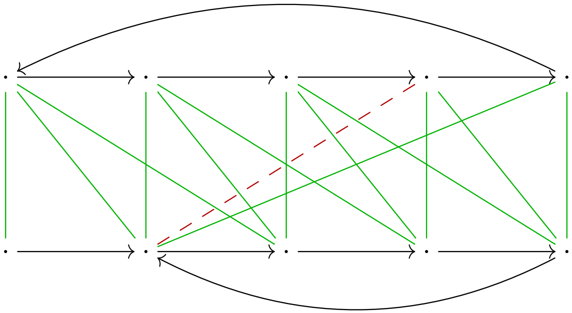
The previous algorithm is quadratic



12 pairs

Complexity

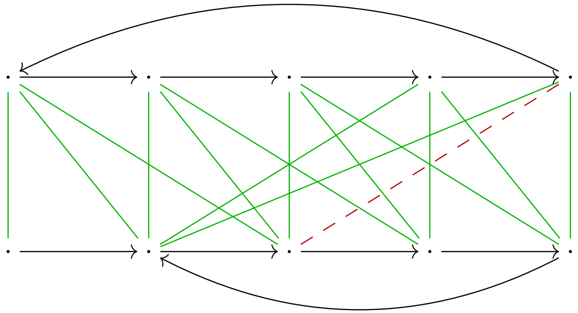
The previous algorithm is quadratic



13 pairs

Complexity

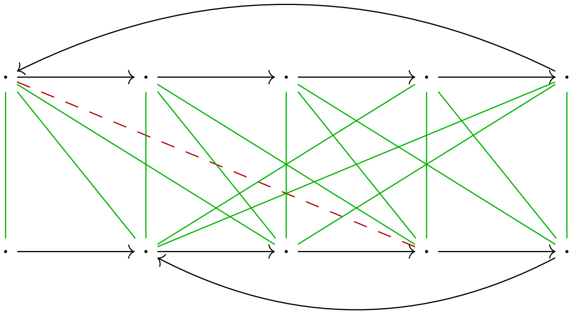
The previous algorithm is quadratic



14 pairs

Complexity

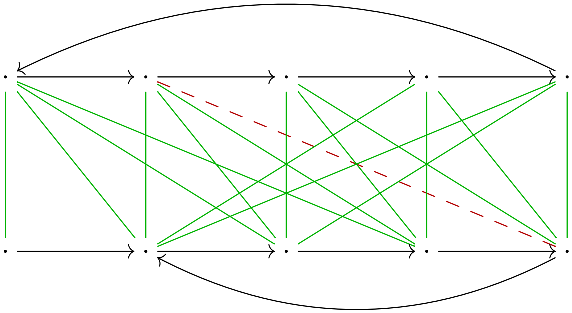
The previous algorithm is quadratic



15 pairs

Complexity

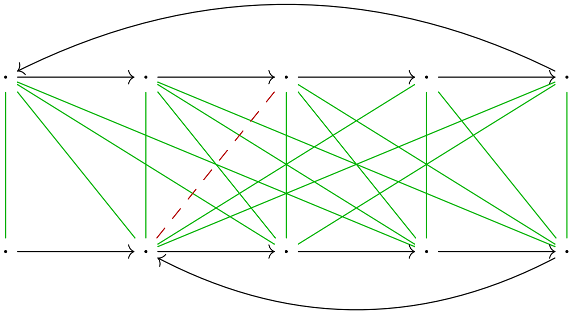
The previous algorithm is quadratic



16 pairs

Complexity

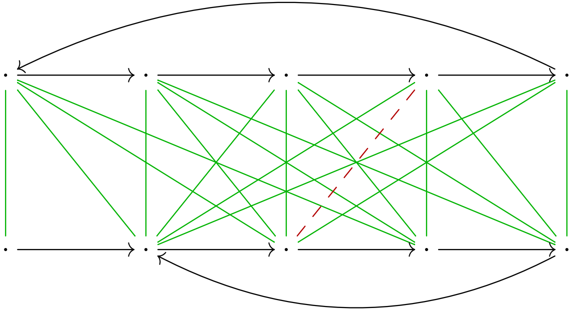
The previous algorithm is quadratic



17 pairs

Complexity

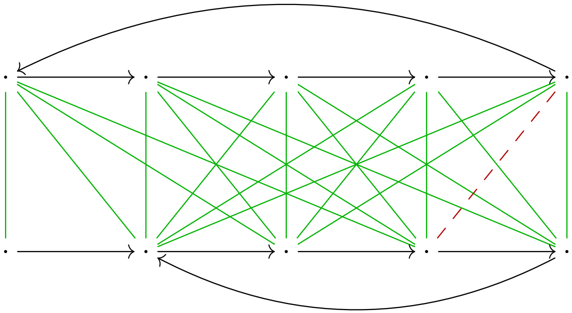
The previous algorithm is quadratic



18 pairs

Complexity

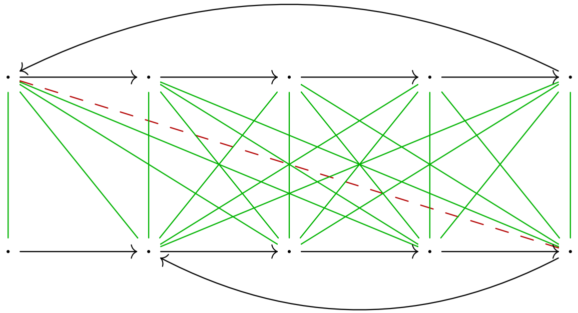
The previous algorithm is quadratic



19 pairs

Complexity

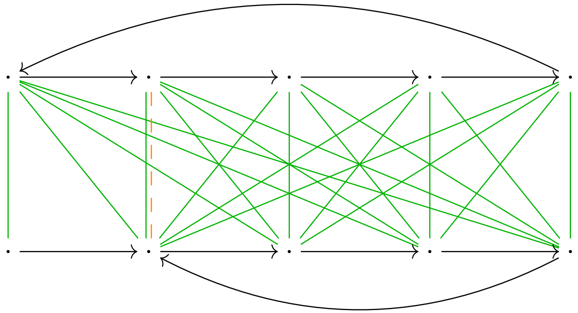
The previous algorithm is quadratic



20 pairs

Complexity

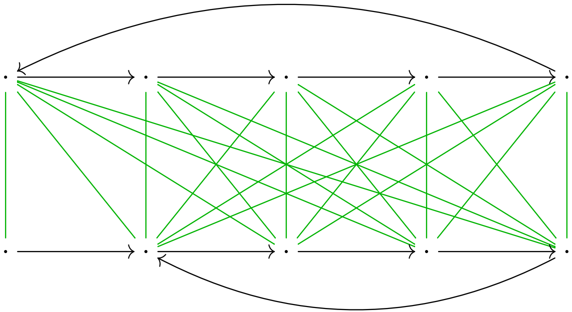
The previous algorithm is quadratic



21 pairs

Complexity

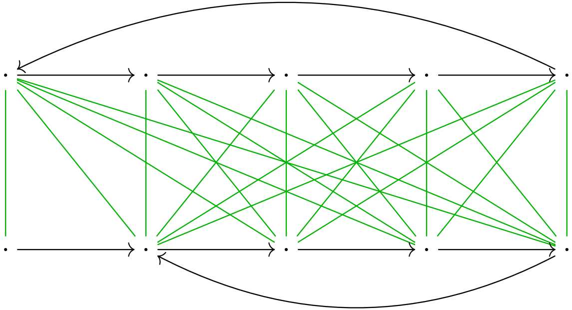
The previous algorithm is quadratic



21 pairs

First improvement

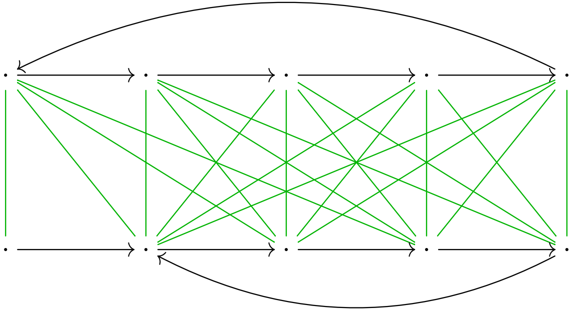
One can stop much earlier



21 pairs

First improvement

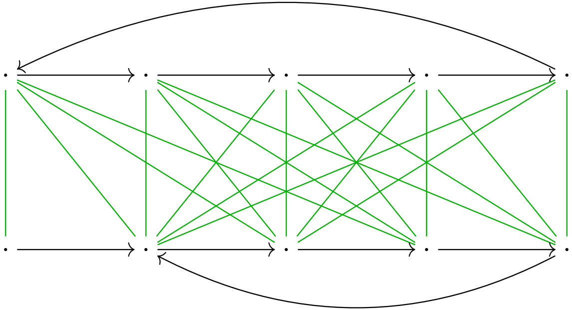
One can stop much earlier



~~21~~ 20 pairs

First improvement

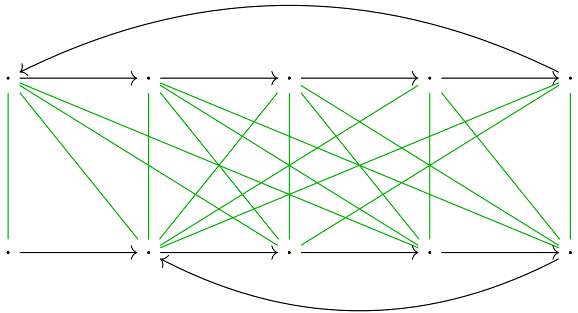
One can stop much earlier



~~21~~ 19 pairs

First improvement

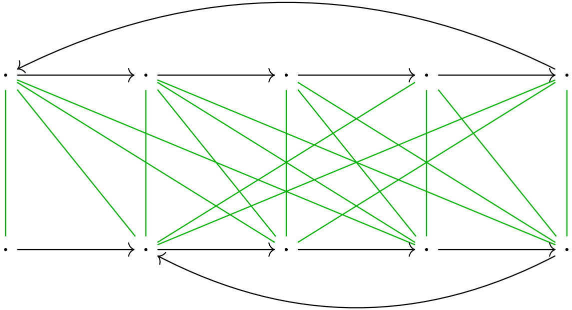
One can stop much earlier



~~21~~ 18 pairs

First improvement

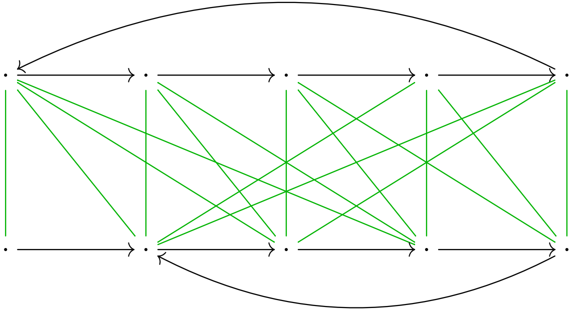
One can stop much earlier



~~21~~ 17 pairs

First improvement

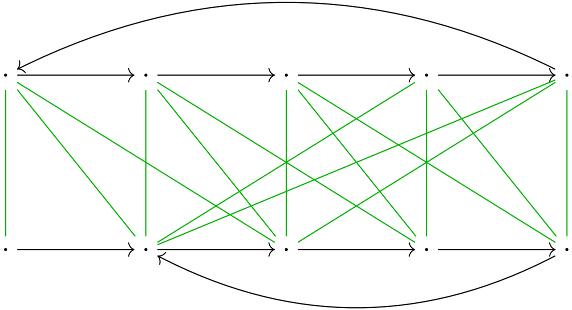
One can stop much earlier



~~21~~ 16 pairs

First improvement

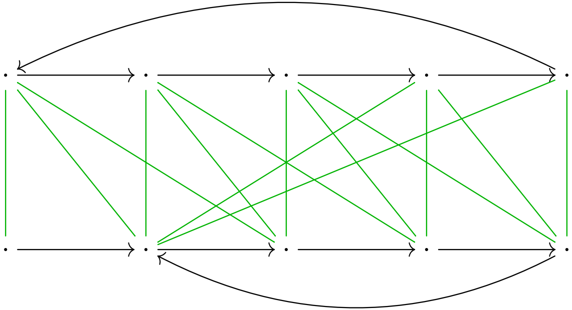
One can stop much earlier



~~21~~ 15 pairs

First improvement

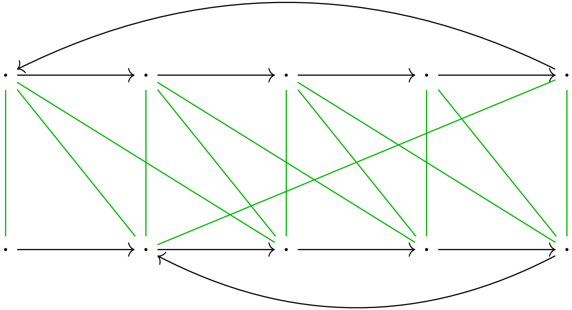
One can stop much earlier



~~21~~ 14 pairs

First improvement

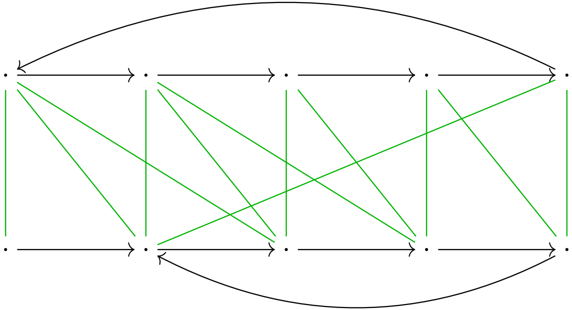
One can stop much earlier



~~21~~ 13 pairs

First improvement

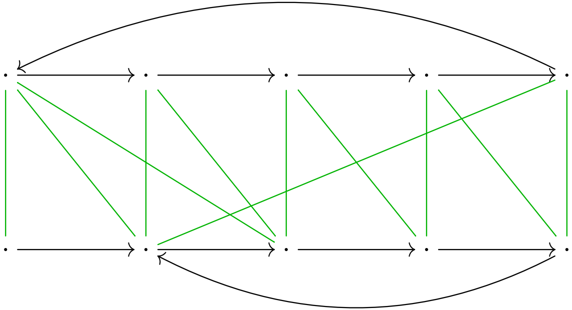
One can stop much earlier



~~21~~ 12 pairs

First improvement

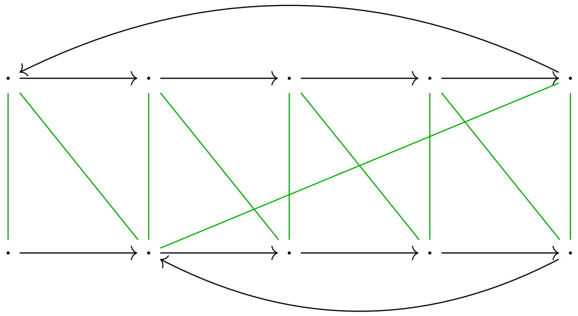
One can stop much earlier



~~21~~ 11 pairs

First improvement

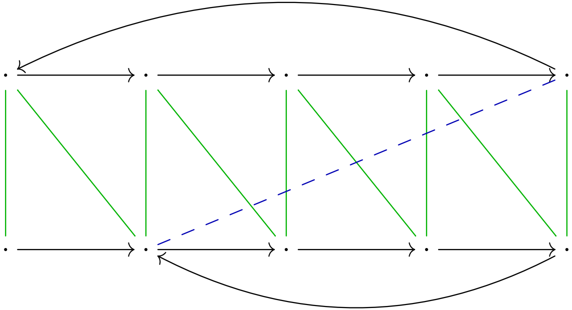
One can stop much earlier



~~21~~ 10 pairs

First improvement

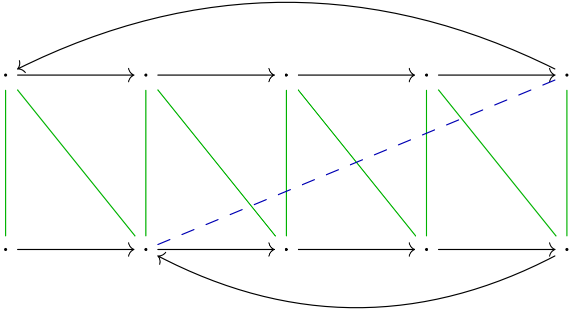
One can stop much earlier



~~21~~ 9 pairs

First improvement

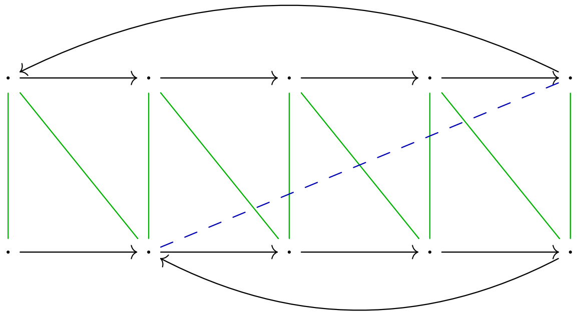
One can stop much earlier



[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:

- Denote by R^e the equivalence closure of R
- R is a bisimulation up to equivalence if $x \; R \; y$ entails
  - $o(x) = o(y)$;
  - for all a, $t_a(x) \; R^e \; t_a(y)$.

Correctness of the improvement

Correctness of HK algorithm, revisited:

- Denote by R^e the equivalence closure of R
- R is a bisimulation up to equivalence if $x \, R \, y$ entails
  - $o(x) = o(y)$;
  - for all a, $t_a(x) \, R^e \, t_a(y)$.
- *Theorem:* $L(x) = L(y)$ iff
  there exists a bisimulation up to equivalence R, with $x \, R \, y$

Correctness of the improvement

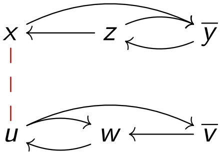
Correctness of HK algorithm, revisited:

- Denote by R^e the equivalence closure of R
- R is a bisimulation up to equivalence if $x \, R \, y$ entails
  - $o(x) = o(y)$;
  - for all a, $t_a(x) \, R^e \, t_a(y)$.
- *Theorem:* $L(x) = L(y)$ iff
  there exists a bisimulation up to equivalence R, with $x \, R \, y$

Ten years before Milner and Park!

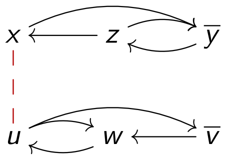
Non-Deterministic Automata

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



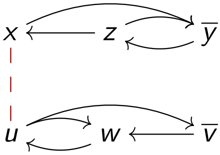
Non-Deterministic Automata

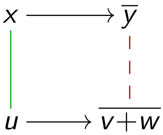
Use Hopcroft and Karp on the fly, through the powerset construction:



Non-Deterministic Automata

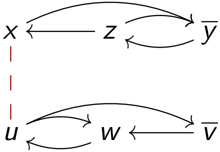
Use Hopcroft and Karp on the fly, through the powerset construction:

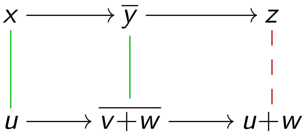




Non-Deterministic Automata

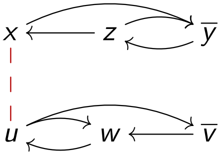
Use Hopcroft and Karp on the fly, through the powerset construction:

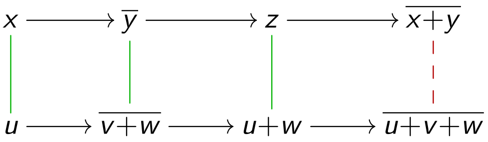




Non-Deterministic Automata

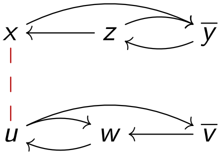
Use Hopcroft and Karp on the fly, through the powerset construction:

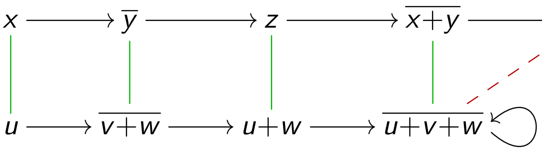




Non-Deterministic Automata

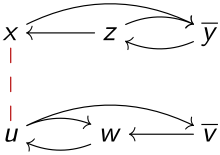
Use Hopcroft and Karp on the fly, through the powerset construction:

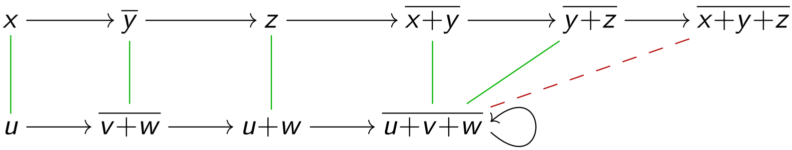




Non-Deterministic Automata

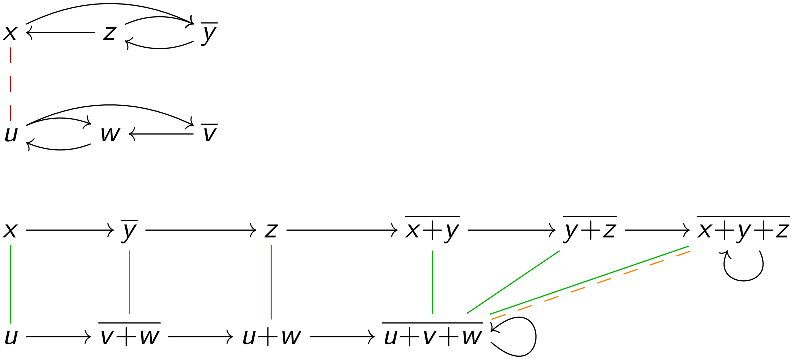
Use Hopcroft and Karp on the fly, through the powerset construction:





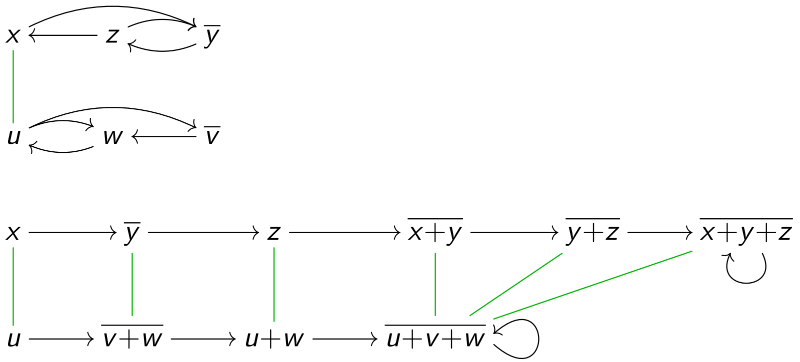
Non-Deterministic Automata

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



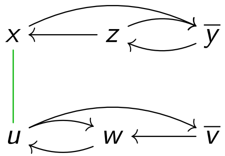
Non-Deterministic Automata

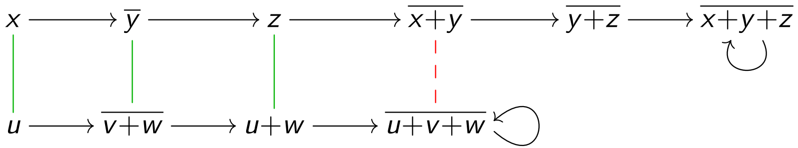
Use Hopcroft and Karp on the fly, through the powerset construction:



Non-Deterministic Automata

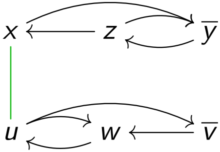
One can do better:



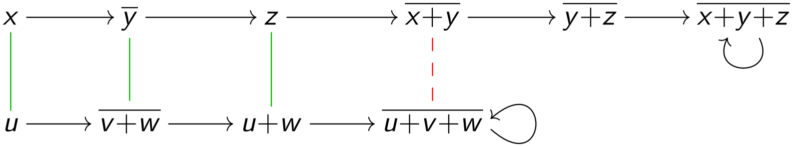


Non-Deterministic Automata

One can do better:

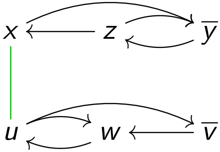


$$
\begin{array}{rl}
 & (x,\ u) \\
+ & (y,\ v{+}w) \\
\hline
= & (x{+}y,\ u{+}v{+}w)
\end{array}
$$

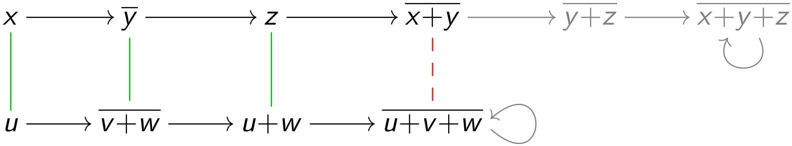


Non-Deterministic Automata

One can do better:



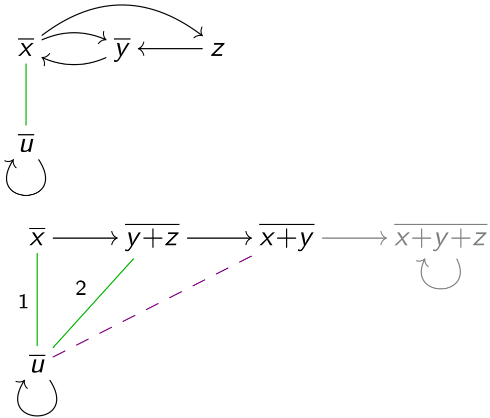
$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$



using bisimulations up to union

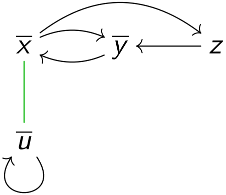
Non-Deterministic Automata

One can do even better:

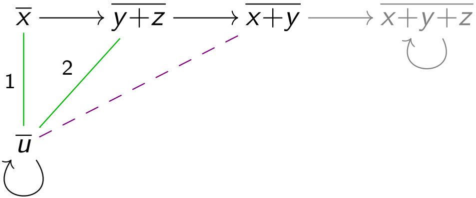


Non-Deterministic Automata

One can do even better:

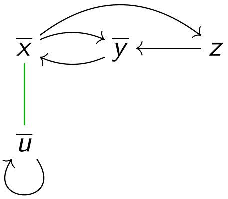


$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z &&\\
&= u && (2)
\end{aligned}
$$

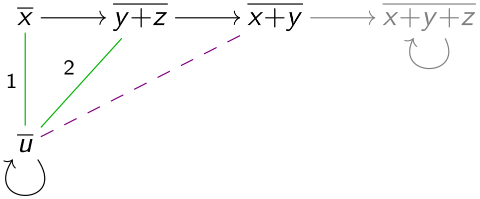


Non-Deterministic Automata

One can do even better:

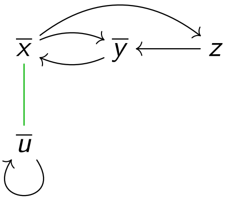


$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z\\
&= u && (2)
\end{aligned}
$$

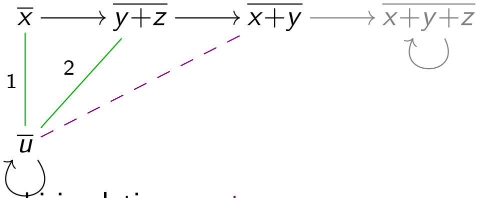


Non-Deterministic Automata

One can do even better:



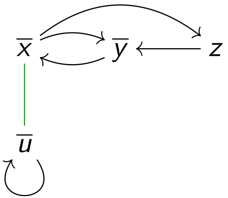
$$\begin{aligned} x+y &= u+y && (1) \\ &= y+z+y && (2) \\ &= y+z && \\ &= u && (2) \end{aligned}$$



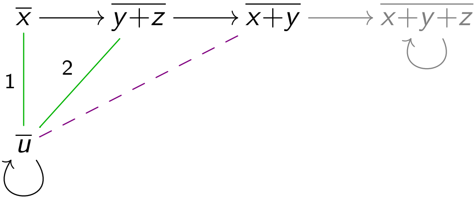
using bisimulations up to congruence

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]

Outline

Up-to techniques at work

Deterministic finite automata

Non-Deterministic Automata

Stream calculus

Abstract coinduction in complete lattices

Coinductive stream calculus [Rutten'03]

Streams can be defined by behavioural differential equations:

$$
\begin{array}{lll}
(\sigma + \tau)' = \sigma' + \tau' & o(\sigma + \tau) = o(\sigma) + o(\tau) & \text{(sum)} \\
(\sigma \otimes \tau)' = \sigma' \otimes \tau + \sigma \otimes \tau' & o(\sigma \otimes \tau) = o(\sigma) \times o(\tau) & \text{(shuffle)} \\
(\sigma^{-1})' = -\sigma' \otimes (\sigma^{-1} \otimes \sigma^{-1}) & o(\sigma^{-1}) = o(\sigma)^{-1} & \text{(inverse)} \\
(i)' = 0 & o(i) = i & \text{(numbers)}
\end{array}
$$

A bisimulation is a relation R such that $\sigma \; R \; \tau$ entails $o(\sigma) = o(\tau)$ and $\sigma' \; R \; \tau'$

- Let us show that $\sigma + 0 \sim \sigma$

Coinductive stream calculus [Rutten'03]

Streams can be defined by behavioural differential equations:

$$
\begin{aligned}
(\sigma+\tau)' &= \sigma'+\tau' & o(\sigma+\tau) &= o(\sigma)+o(\tau) & &\text{(sum)}\\
(\sigma\otimes\tau)' &= \sigma'\otimes\tau+\sigma\otimes\tau' & o(\sigma\otimes\tau) &= o(\sigma)\times o(\tau) & &\text{(shuffle)}\\
(\sigma^{-1})' &= -\sigma'\otimes(\sigma^{-1}\otimes\sigma^{-1}) & o(\sigma^{-1}) &= o(\sigma)^{-1} & &\text{(inverse)}\\
(i)' &= 0 & o(i) &= i & &\text{(numbers)}
\end{aligned}
$$

A bisimulation is a relation R such that $\sigma \; R \; \tau$ entails $o(\sigma) = o(\tau)$ and $\sigma' \; R \; \tau'$

- Let us show that $\sigma + 0 \sim \sigma$
- How about $\sigma \otimes 1 \sim \sigma$?

Coinductive stream calculus [Rutten'03]

Streams can be defined by behavioural differential equations:

$(\sigma+\tau)' = \sigma' + \tau'$	$o(\sigma+\tau) = o(\sigma) + o(\tau)$	(sum)
$(\sigma\otimes\tau)' = \sigma'\otimes\tau + \sigma\otimes\tau'$	$o(\sigma\otimes\tau) = o(\sigma)\times o(\tau)$	(shuffle)
$(\sigma^{-1})' = -\sigma'\otimes(\sigma^{-1}\otimes\sigma^{-1})$	$o(\sigma^{-1}) = o(\sigma)^{-1}$	(inverse)
$(i)' = 0$	$o(i) = i$	(numbers)

A bisimulation up to $\sim$ and is a relation R such that $\sigma\ R\ \tau$ entails $o(\sigma) = o(\tau)$ and $\sigma' \sim R \sim \tau'$

- Let us show that $\sigma + 0 \sim \sigma$
- How about $\sigma \otimes 1 \sim \sigma$?
- And $\sigma \otimes \sigma^{-1} \sim 1$?

Coinductive stream calculus [Rutten'03]

Streams can be defined by behavioural differential equations:

$(\sigma + \tau)' = \sigma' + \tau'$	$o(\sigma + \tau) = o(\sigma) + o(\tau)$	(sum)
$(\sigma \otimes \tau)' = \sigma' \otimes \tau + \sigma \otimes \tau'$	$o(\sigma \otimes \tau) = o(\sigma) \times o(\tau)$	(shuffle)
$(\sigma^{-1})' = -\sigma' \otimes (\sigma^{-1} \otimes \sigma^{-1})$	$o(\sigma^{-1}) = o(\sigma)^{-1}$	(inverse)
$(i)' = 0$	$o(i) = i$	(numbers)

A bisimulation up to $\sim$ and up to context is a relation R such that $\sigma \; R \; \tau$ entails $o(\sigma) = o(\tau)$ and $\sigma' \sim c(R) \sim \tau'$

- Let us show that $\sigma + 0 \sim \sigma$
- How about $\sigma \otimes 1 \sim \sigma$?
- And $\sigma \otimes \sigma^{-1} \sim 1$?

Lessons learned from the examples

- A wide range of up-to techniques
  - up to equivalence
  - up to bisimilarity
  - up to union
  - up to context
- For different kind of systems
  - {deterministic,non-deterministic,(weighted)} automata,
  - streams
  - process algebra [Milner'89, Sangiorgi'98]
- Sometimes they need to be combined together
  - union and equivalence $\rightsquigarrow$ congruence (NFA)
  - c and $R \mapsto \sim R \sim$ $\rightsquigarrow$ $R \mapsto \sim c(R) \sim$ (streams)

Lessons learned from the examples

- A wide range of up-to techniques
  - up to equivalence
  - up to bisimilarity
  - up to union
  - up to context
- For different kind of systems
  - {deterministic,non-deterministic,(weighted)} automata,
  - streams
  - process algebra [Milner'89, Sangiorgi'98]
- Sometimes they need to be combined together
  - union and equivalence $\rightsquigarrow$ congruence (NFA)
  - c and $R \mapsto \sim R \sim$ $\rightsquigarrow$ $R \mapsto \sim c(R) \sim$ (streams)
- ... but is this composition always sound?

Two questions

- Can we study all these proof principles in one framework?
- Can we derive conditions for soundness?

Compatiblity

Use Pous's algebra of enhancements: abstract framework in terms of lattices and monotone functions.

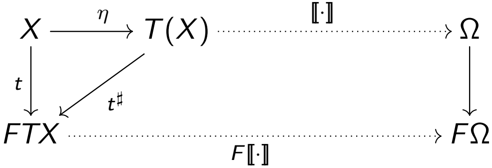
- b-simulation: $R \subseteq b(R)$;
- b-simulation up to f: $R \subseteq b(f(R))$
- Definition: f is b-compatible if $f \circ b \subseteq b \circ f$
- b-compatible functions: f sound and closed under composition

FT-Coalgebra

Coalgebras make it possible to encompass the previous examples in a uniform setting:

systems	functor (F)	monad (T)
deterministic automata	$2 \times -^A$	$(-)$
non-deterministic automata	$2 \times (-)^A$	$\mathcal{P}_{\mathsf{f}}(-)$
weigthed automata	$\mathbb{R} \times (-)^A$	$\mathbb{R}^{(-)}$
streams	$\mathbb{R} \times -$	$(-)$

First generalized powerset construction and then finality:



Behavioural equivalence becomes $x \sim_\alpha y \triangleq [\![\eta(x)]\!] = [\![\eta(y)]\!]$

Coalgebraic bisimulation

Given an F-coalgebra (X, α), define the following function on binary relations:

$$b_\alpha(R) = \{(x, y) \mid \exists z \in FR,\ F(\pi_1^R) = \alpha(x), F(\pi_2^R) = \alpha(y)\}$$

Coalgebraic bisimulation

Given an F-coalgebra (X, α), define the following function on binary relations:

$$b_\alpha(R) = \{(x, y) \mid \exists z \in FR,\ F(\pi_1^R) = \alpha(x), F(\pi_2^R) = \alpha(y)\}$$

Proposition [Rot, Bonchi, Bonsangue, Pous, Rutten, Silva'13]:
b_α satisfies (†) iff F preserves weak pullbacks

(†) $\forall R\, S, b(R) \cdot b(S) \subseteq b(R \cdot S)$

- up to equivalence (almost) always comes for free

Contexts: bialgebras

What about the up to union/context techniques?

- They are all instances of the same framework
  we just exploit some algebraic structure of the state-space:
    - a semilattice for non-deterministic automata
    - a vector space for weighted automata
    - a syntax for streams
- Can be captured using λ-bialgebras:

$$\lambda : TF \Rightarrow FT$$

$$TX \xrightarrow{\beta} X \xrightarrow{\alpha} FX$$

$$(\alpha \circ \beta = F\beta \circ \lambda_X \circ T\alpha)$$

[Turi&Plotkin'97, Bartels'04, Klin'11]

Summary

Coalgebras make it possible

- to exploit the abstract theory of up-to techniques for a wide range of systems
- to design algorithms in a uniform way

  (e.g., HKC for must-testing [Bonchi, Caltais, Pous, Silva'13])