Lecture four:

Coalgebraic up-to techniques

Jan Rutten

CWI Amsterdam & Radboud University Nijmegen

IPM, Tehran - 13 January 2016

Context

Combining algebra and coalgebra together yields ...

... a set of very efficient tools and proof techniques for proving the equivalence of various types of systems (such as automata, streams, etc.).

Cf. Hacking nondeterminism with induction and coinduction.
Filippo Bonchi and Damien Pous.
Communications of the ACM 58(2), 2015.
(Also in: Proceedings of POPL 2013.)

Acknowledgement: most of these slides are taken from the presentation of Damien Pous at CALCO 2013.

Context

Combining algebra and coalgebra together yields ...

... a set of very efficient tools and proof techniques for proving the equivalence of various types of systems (such as automata, streams, etc.).

Cf. Hacking nondeterminism with induction and coinduction.
Filippo Bonchi and Damien Pous.
Communications of the ACM 58(2), 2015.
(Also in: Proceedings of POPL 2013.)

Acknowledgement: most of these slides are taken from the presentation of Damien Pous at CALCO 2013.

Context

Combining algebra and coalgebra together yields ...

... a set of very efficient tools and proof techniques for proving the equivalence of various types of systems (such as automata, streams, etc.).

Cf. Hacking nondeterminism with induction and coinduction.
Filippo Bonchi and Damien Pous.
Communications of the ACM 58(2), 2015.
(Also in: Proceedings of POPL 2013.)

Acknowledgement: most of these slides are taken from the presentation of Damien Pous at CALCO 2013.

Context

Combining algebra and coalgebra together yields . . .

. . . a set of very efficient tools and proof techniques for proving the equivalence of various types of systems (such as automata, streams, etc.).

Cf. Hacking nondeterminism with induction and coinduction.
Filippo Bonchi and Damien Pous.
Communications of the ACM 58(2), 2015.
(Also in: Proceedings of POPL 2013.)

Acknowledgement: most of these slides are taken from the presentation of Damien Pous at CALCO 2013.

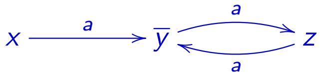
Table of contents

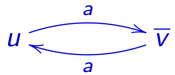
1. Bisimulation up-to

- Deterministic automata
- Nondeterministic automata
- Weighted automata
- Streams

Deterministic finite automata

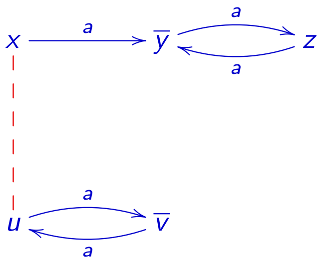
The following automata are equivalent:





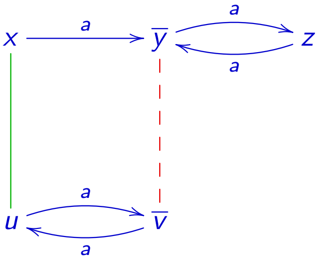
Deterministic finite automata

The following automata are equivalent:



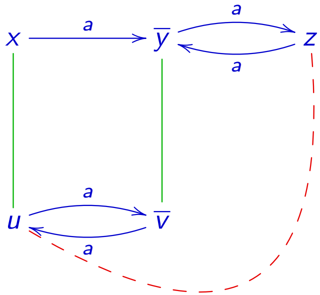
Deterministic finite automata

The following automata are equivalent:



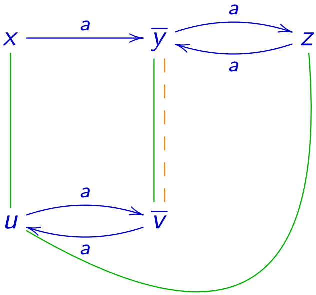
Deterministic finite automata

The following automata are equivalent:



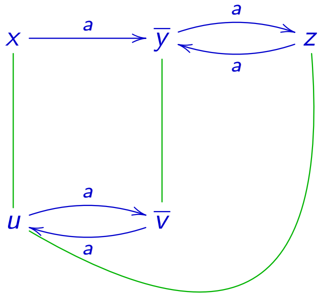
Deterministic finite automata

The following automata are equivalent:



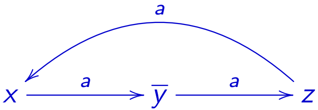
Deterministic finite automata

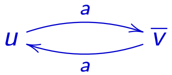
The following automata are equivalent:



Deterministic finite automata

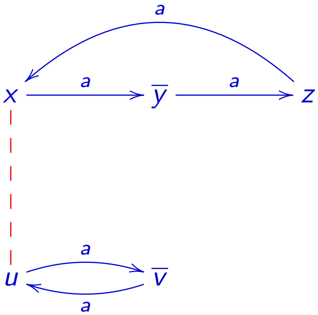
These two automata are **not** equivalent:





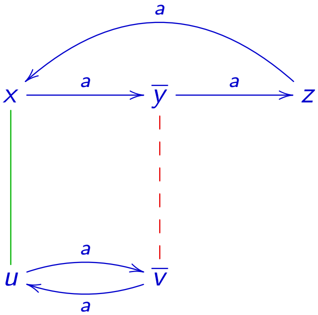
Deterministic finite automata

These two automata are **not** equivalent:



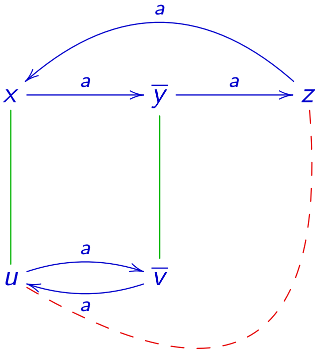
Deterministic finite automata

These two automata are **not** equivalent:



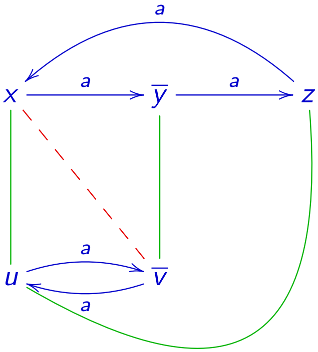
Deterministic finite automata

These two automata are **not** equivalent:



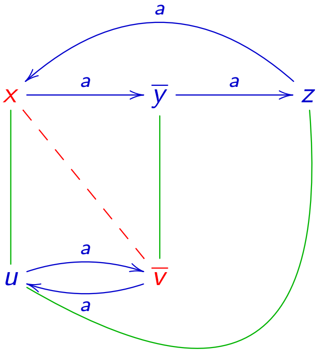
Deterministic finite automata

These two automata are **not** equivalent:



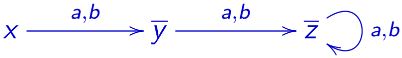
Deterministic finite automata

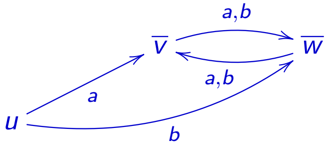
These two automata are **not** equivalent:



Deterministic finite automata

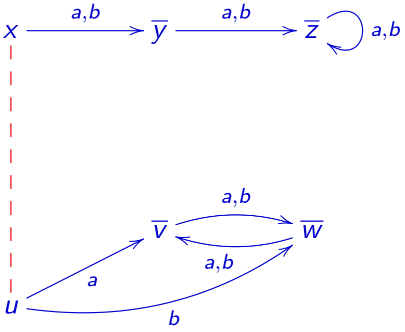
Another example, with two letters:





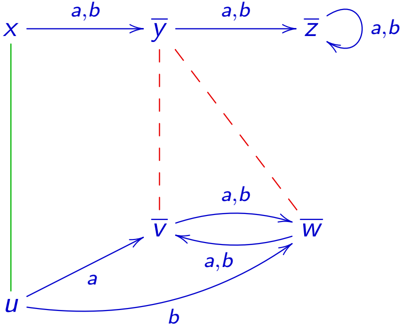
Deterministic finite automata

Another example, with two letters:



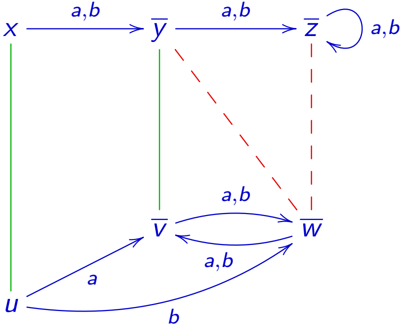
Deterministic finite automata

Another example, with two letters:



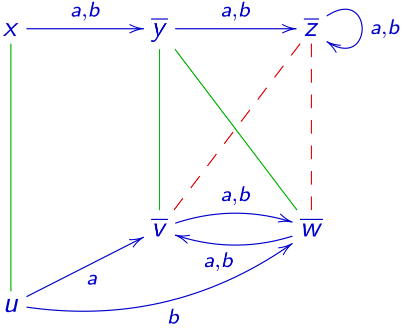
Deterministic finite automata

Another example, with two letters:



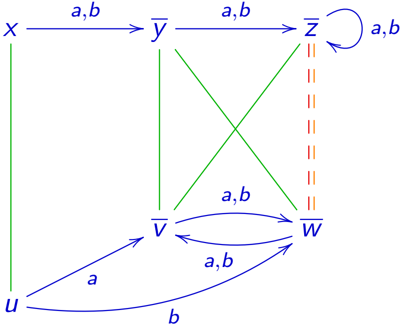
Deterministic finite automata

Another example, with two letters:



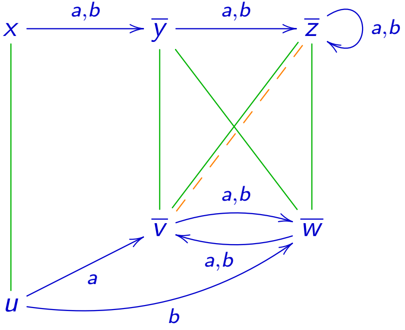
Deterministic finite automata

Another example, with two letters:



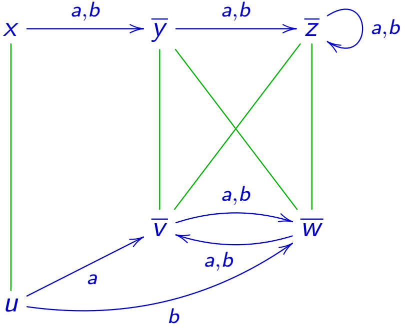
Deterministic finite automata

Another example, with two letters:



Deterministic finite automata

Another example, with two letters:



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

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 $x \; R \; y$

The previous algorithm attempts to construct a bisimulation

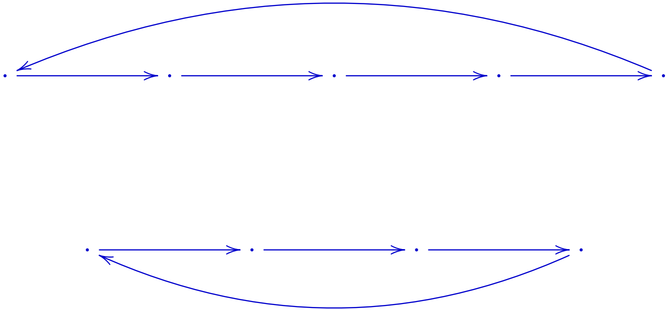
Correctness

- 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 $x \, R \, y$

The previous algorithm attempts to construct a bisimulation

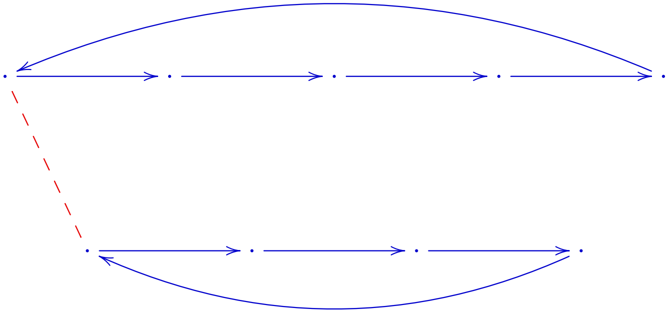
Checking language equivalence

Deterministic case, naive algorithm: quadratic complexity



Checking language equivalence

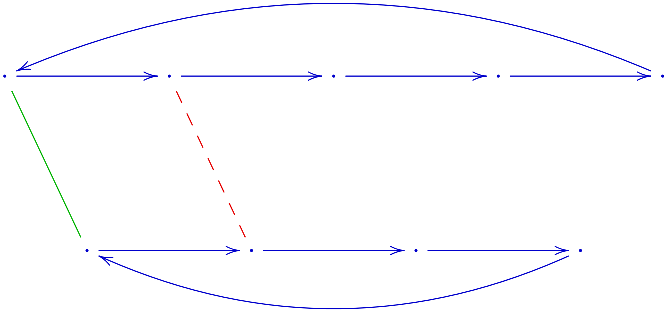
Deterministic case, naive algorithm: quadratic complexity



1 pairs

Checking language equivalence

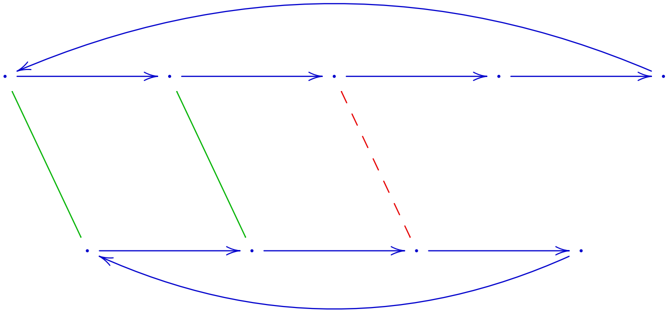
Deterministic case, naive algorithm: quadratic complexity



2 pairs

Checking language equivalence

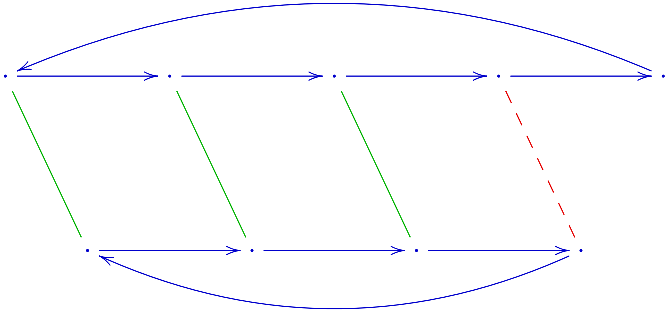
Deterministic case, naive algorithm: quadratic complexity



3 pairs

Checking language equivalence

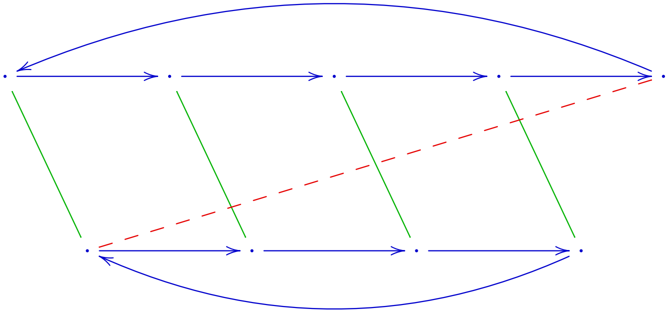
Deterministic case, naive algorithm: quadratic complexity



4 pairs

Checking language equivalence

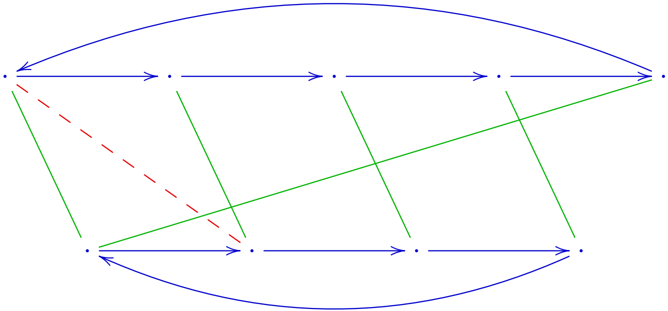
Deterministic case, naive algorithm: quadratic complexity



5 pairs

Checking language equivalence

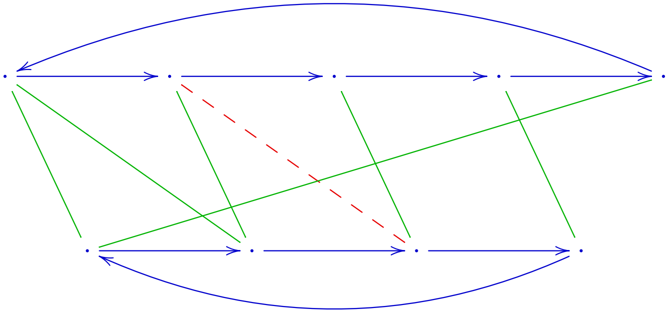
Deterministic case, naive algorithm: quadratic complexity



6 pairs

Checking language equivalence

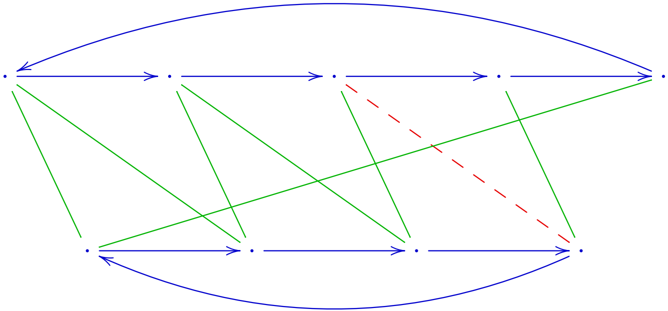
Deterministic case, naive algorithm: quadratic complexity



7 pairs

Checking language equivalence

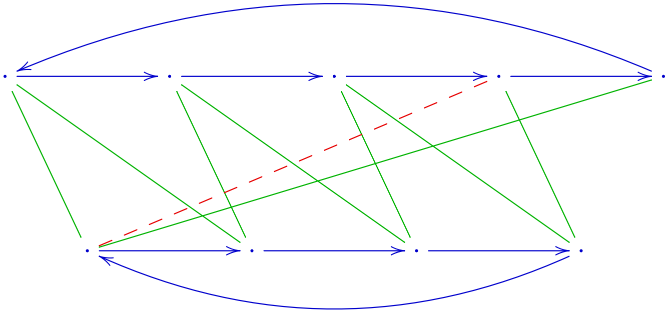
Deterministic case, naive algorithm: quadratic complexity



8 pairs

Checking language equivalence

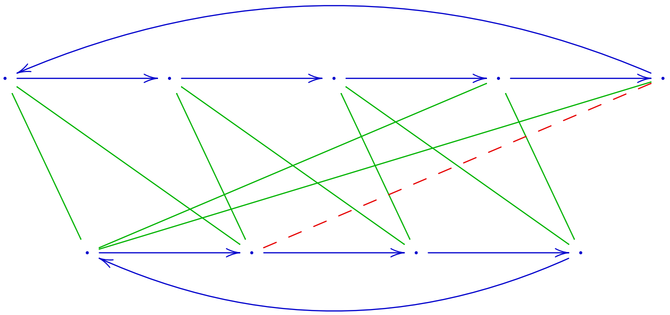
Deterministic case, naive algorithm: quadratic complexity



9 pairs

Checking language equivalence

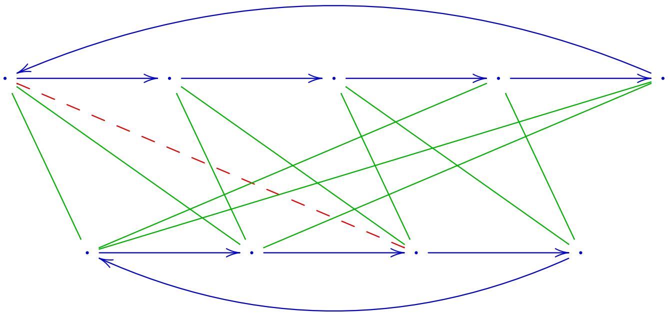
Deterministic case, naive algorithm: quadratic complexity



10 pairs

Checking language equivalence

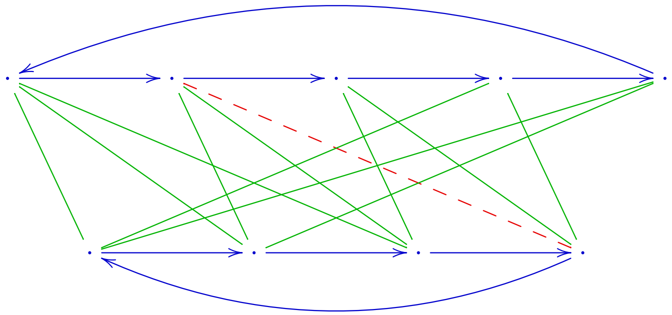
Deterministic case, naive algorithm: quadratic complexity



11 pairs

Checking language equivalence

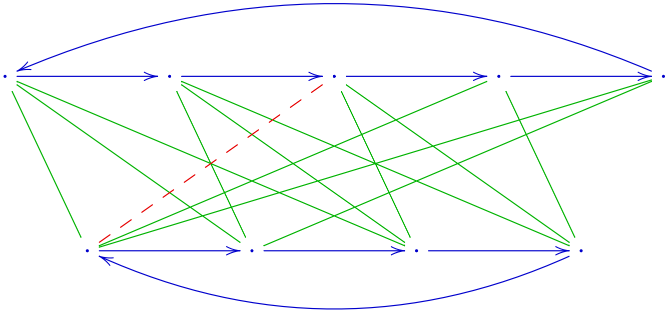
Deterministic case, naive algorithm: quadratic complexity



12 pairs

Checking language equivalence

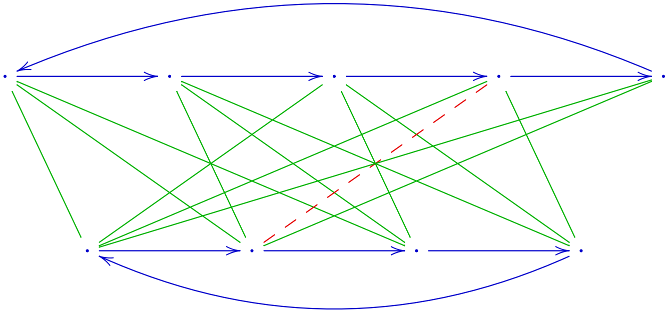
Deterministic case, naive algorithm: quadratic complexity



13 pairs

Checking language equivalence

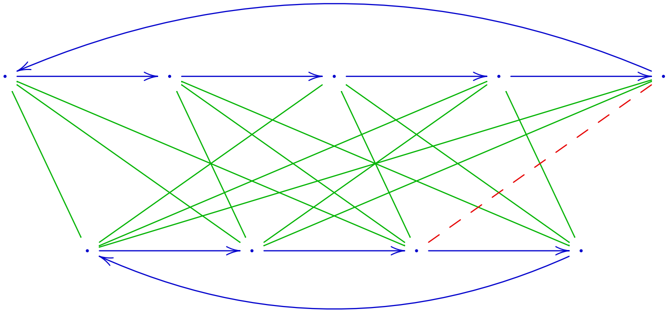
Deterministic case, naive algorithm: quadratic complexity



14 pairs

Checking language equivalence

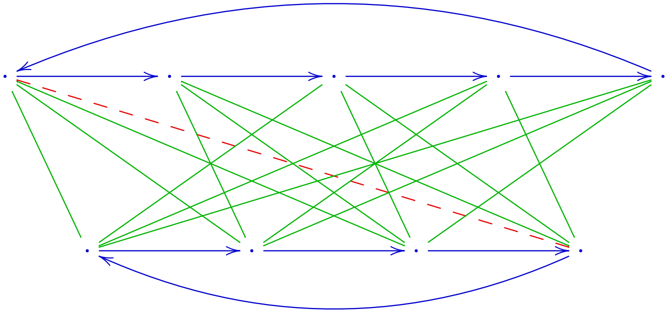
Deterministic case, naive algorithm: quadratic complexity



15 pairs

Checking language equivalence

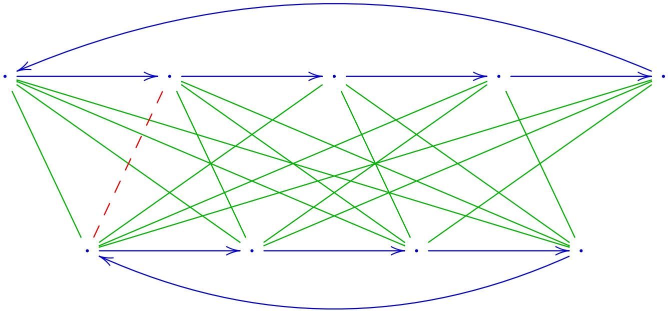
Deterministic case, naive algorithm: quadratic complexity



16 pairs

Checking language equivalence

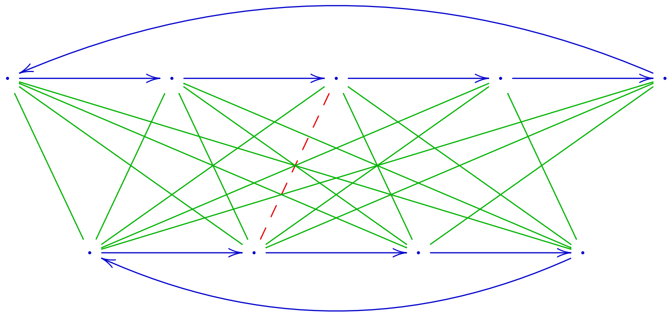
Deterministic case, naive algorithm: quadratic complexity



17 pairs

Checking language equivalence

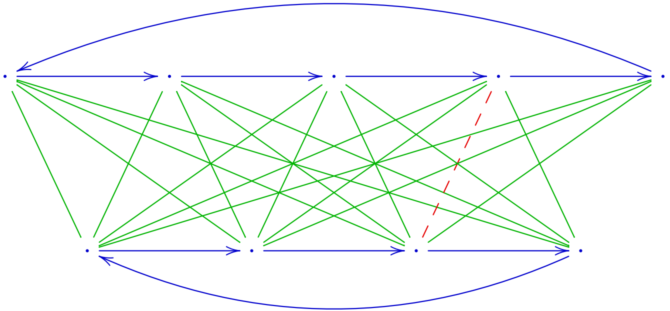
Deterministic case, naive algorithm: quadratic complexity



18 pairs

Checking language equivalence

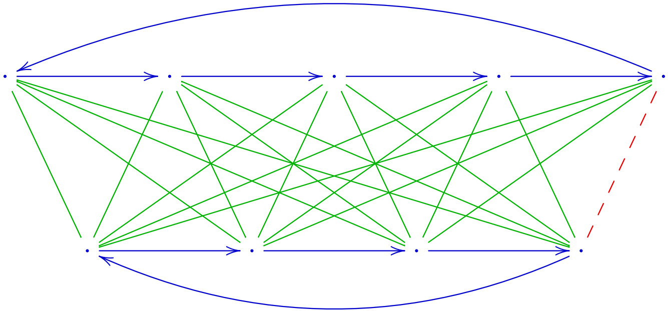
Deterministic case, naive algorithm: quadratic complexity



19 pairs

Checking language equivalence

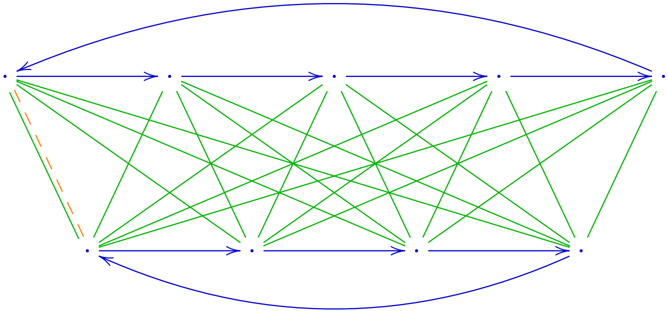
Deterministic case, naive algorithm: quadratic complexity



20 pairs

Checking language equivalence

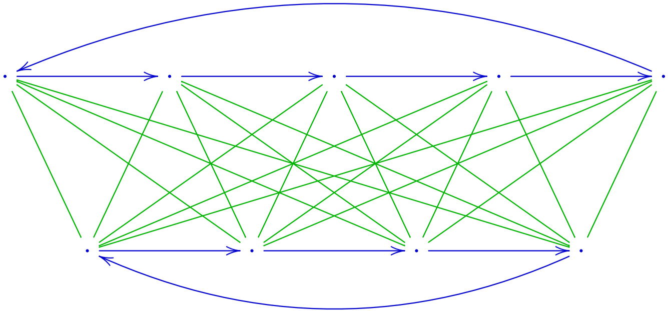
Deterministic case, naive algorithm: quadratic complexity



20 pairs

Checking language equivalence

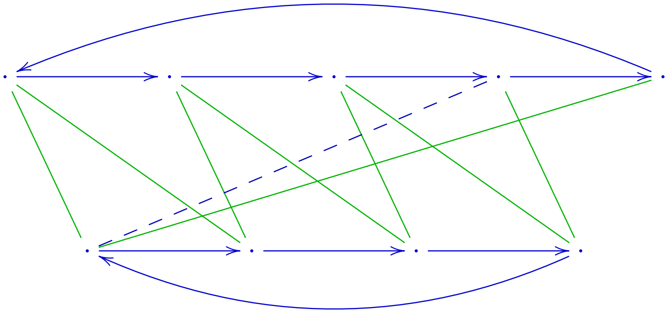
Deterministic case, naive algorithm: quadratic complexity



20 pairs

Checking language equivalence

One can stop much earlier

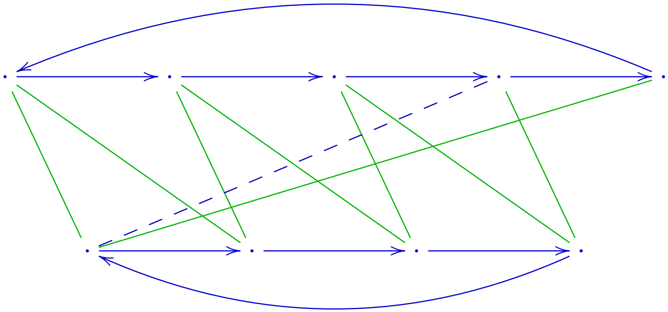


~~20~~ 8 pairs

Complexity: almost linear [Tarjan '75]

Checking language equivalence

One can stop much earlier



[Hopcroft and Karp '71]

[Tarjan '75]

Complexity: almost linear

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!

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!

Correctness of the improvement

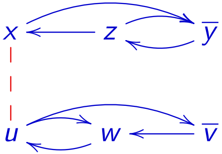
Correctness of HK algorithm, revisited:

- 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:

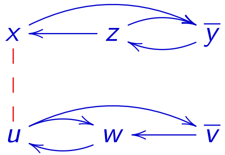


x $\overline{y}$ z $\overline{x+y}$ $\overline{y+z}$ $\overline{x+y+z}$

u $\overline{v+w}$ $u+w$ $\overline{u+v+w}$

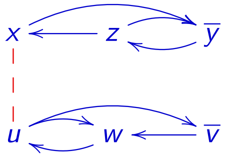
Non-Deterministic Automata

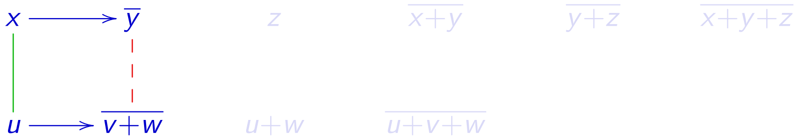
Use Hopcroft and Karp on the fly, through the powerset construction:



Non-Deterministic Automata

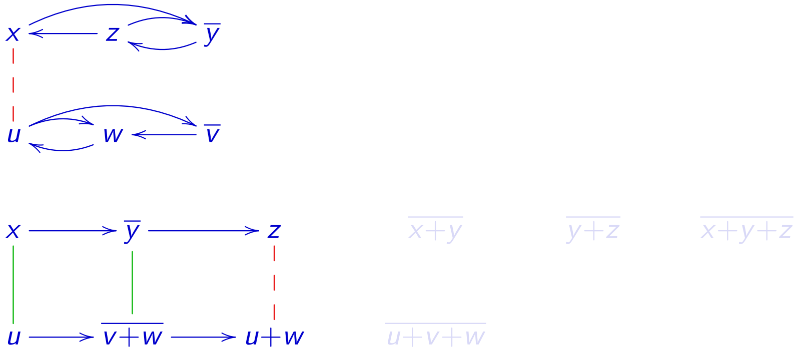
Use Hopcroft and Karp on the fly, through the powerset construction:





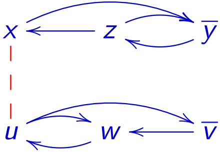
Non-Deterministic Automata

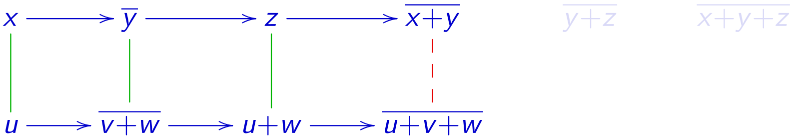
Use Hopcroft and Karp on the fly, through the powerset construction:



Non-Deterministic Automata

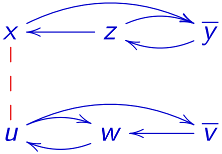
Use Hopcroft and Karp on the fly, through the powerset construction:

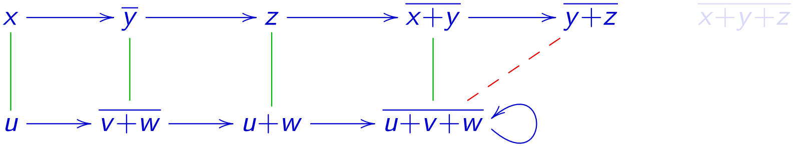




Non-Deterministic Automata

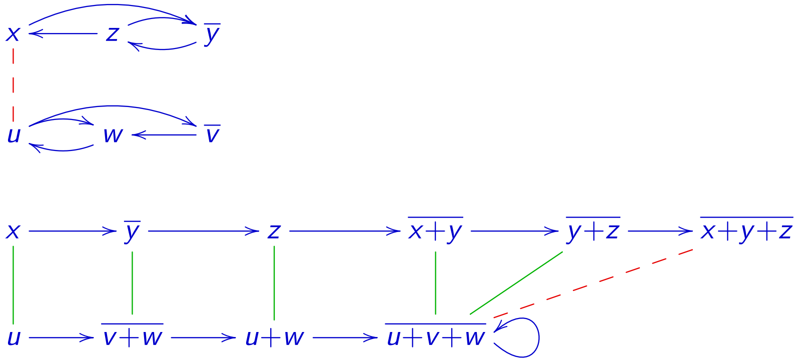
Use Hopcroft and Karp on the fly, through the powerset construction:





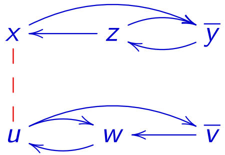
Non-Deterministic Automata

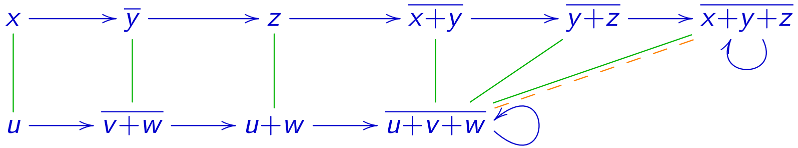
Use Hopcroft and Karp on the fly, through the powerset construction:



Non-Deterministic Automata

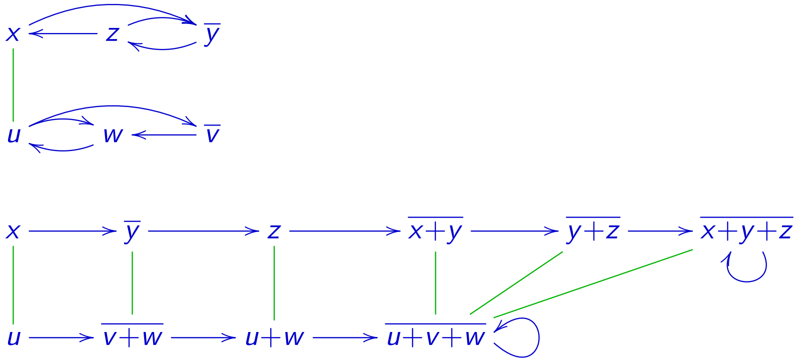
Use Hopcroft and Karp on the fly, through the powerset construction:





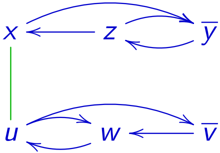
Non-Deterministic Automata

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

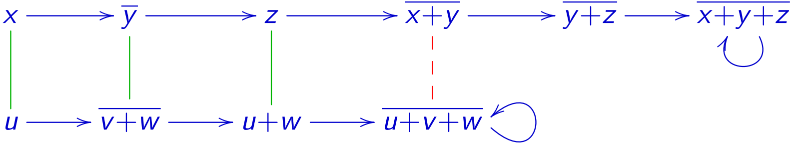


Non-Deterministic Automata

One can do better:



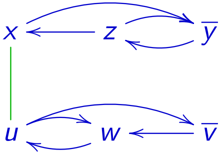
$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$



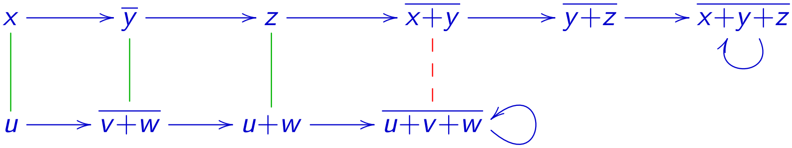
using bisimulations up to union

Non-Deterministic Automata

One can do better:



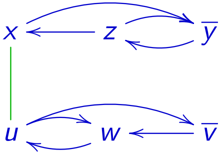
$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$



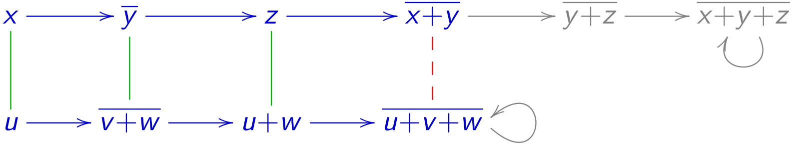
using bisimulations up to union

Non-Deterministic Automata

One can do better:



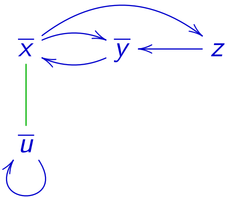
$$
\begin{array}{rl}
 & (x,\ u) \\
+ & (y,\ v+w) \\
\hline
= & (x+y,\ u+v+w)
\end{array}
$$



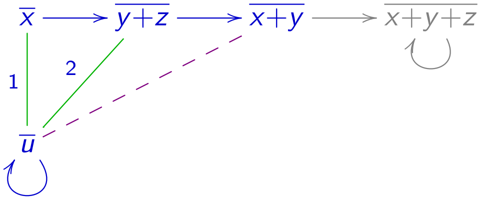
using bisimulations up to union

Non-Deterministic Automata

One can do even better:



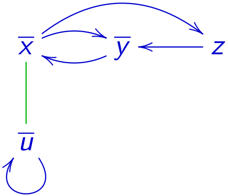
$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z &&\\
&= u && (2)
\end{aligned}
$$



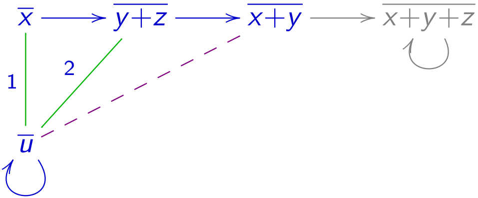
using bisimulations up to congruence.

Non-Deterministic Automata

One can do even better:



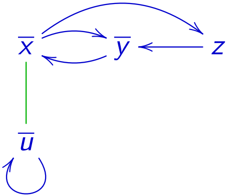
$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z &&\\
&= u && (2)
\end{aligned}
$$



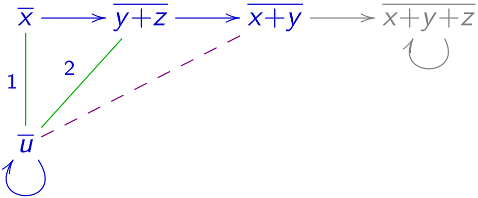
using bisimulations up to congruence.

Non-Deterministic Automata

One can do even better:



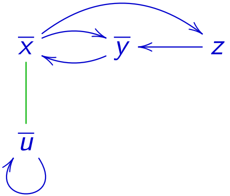
$$
\begin{aligned}
x+y &= u+y && (1)\\
&= y+z+y && (2)\\
&= y+z &&\\
&= u && (2)
\end{aligned}
$$



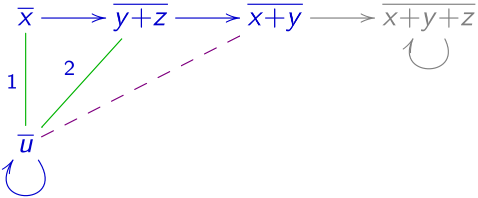
using bisimulations up to congruence.

Non-Deterministic Automata

One can do even better:



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



using bisimulations up to congruence.

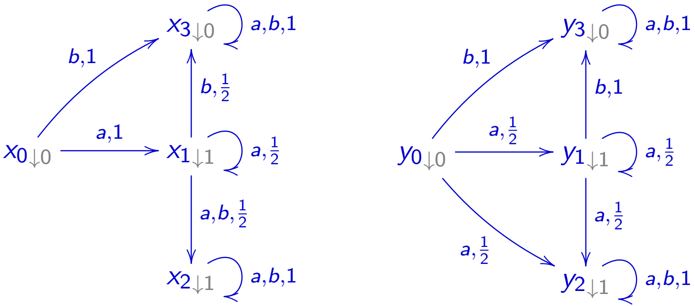
Bonchi and Pous: HKC algorithm

Cf. Hacking nondeterminism with induction and coinduction.
Filippo Bonchi and Damien Pous.
Communications of the ACM 58(2), 2015.
(Also in: Proceedings of POPL 2013.)

A combination of Hopcroft and Karp's algorithm (which is already up-to-equivalence) and the use of bisimulations up to context, yielding:

HKC algorithm: Hopcroft and Karp up to Congruence

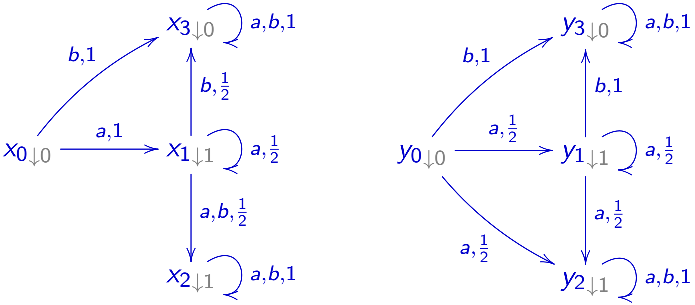
Other classes of examples: weighted automata



- Any bisimulation relating x_0 and y_0 is infinite:
- They are related by a finite bisimulation up to linear combinations:

$$\{(x_0, y_0),\ (x_1, \tfrac{1}{2}y_1 + \tfrac{1}{2}y_2),\ (x_2, y_2),\ (x_3, y_3)\}$$

Other classes of examples: weighted automata



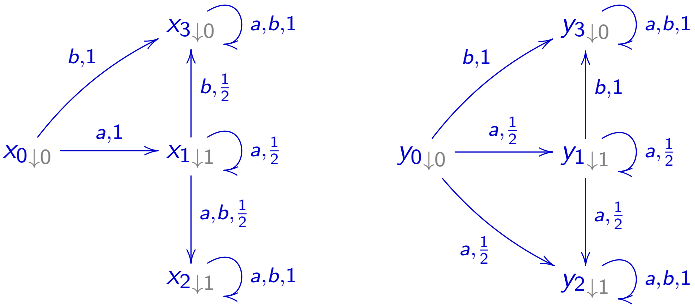
- Any bisimulation relating x_0 and y_0 is infinite:

$$x_0 \xrightarrow{a} x_1 \xrightarrow{a} \frac{1}{2}x_1 + \frac{1}{2}x_2 \xrightarrow{a} \frac{1}{4}x_1 + \frac{3}{4}x_2 \xrightarrow{a} \dots$$

$$y_0 \xrightarrow{a} \frac{1}{2}y_1 + \frac{1}{2}y_2 \xrightarrow{a} \frac{1}{4}y_1 + \frac{3}{4}y_2 \xrightarrow{a} \frac{1}{8}y_1 + \frac{7}{8}y_2 \xrightarrow{a} \dots$$

- They are related by a finite

Other classes of examples: weighted automata



- Any bisimulation relating x_0 and y_0 is infinite:
- They are related by a finite bisimulation up to linear combinations:

$$\{(x_0, y_0),\ (x_1, \tfrac{1}{2}y_1 + \tfrac{1}{2}y_2),\ (x_2, y_2),\ (x_3, y_3)\}$$

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 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 $\sigma \otimes 1 \sim \sigma$?
- 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 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 $\sigma \otimes 1 \sim \sigma$?
- And $\sigma \otimes \sigma^{-1} \sim 1$?

Coinductive stream calculus [Rutten'03]

Streams can be defined by behavioural differential equations:

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

A bisimulation 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:

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

A bisimulation 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 linear combinations
  - 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)

2. General theory: using lattices and fixed points

Abstract coinduction

Let b be a monotone function on a complete lattice

- a b-simulation is an element x such that $x \subseteq b(x)$
- b-similarity is the greatest b-simulation:
  $\mathrm{gfp}(b) \triangleq \bigcup\{x \mid x \subseteq b(x)\}$

(For deterministic automata, one choses

$$b(R) = \{(x, y) \mid o(x) = o(y) \wedge \forall a, t_a(x) \; R \; t_a(y)\}$$

so that b-simulations are precisely the bisimulations, and one proves that $\mathrm{gfp}(b)$ is just language equivalence)

Abstract up-to techniques

Let f be another monotone function

- a b-simulation up to f is an element x such that $x \subseteq b(f(x))$
- f is b-sound if all b-simulations up to f are contained in b-similarity

(Candidates for f: $R \mapsto \sim R \sim$, equivalence closure, context closure, congruence closure . . .)

Unfortunately, b-sound functions cannot be freely composed!

Compatible functions [P.'07, P.&Sangiorgi'12]

Definition: f is b-compatible if $f \circ b \subseteq b \circ f$

Theorem: b-compatible functions are b-sound

Proposition: b-compatible functions can be freely composed

Lemma: in the lattice of relations, $R \mapsto \sim R \sim$ and equivalence closure are b-compatible, provided that

$$\forall R\, S,\; b(R) \cdot b(S) \subseteq b(R \cdot S) \qquad (\dagger)$$

3. General theory: combining algebra and coalgebra

Coalgebra

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

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

Semantics is defined through the final coalgebra:

$$\begin{array}{ccc} X & \xrightarrow{[\![\cdot]\!]} & \Omega \\ \downarrow & & \downarrow \\ FX & \xrightarrow{F[\![\cdot]\!]} & F\Omega \end{array}$$

So is behavioural equivalence: $x \sim_\alpha y \triangleq [\![x]\!] = [\![y]\!]$

Coalgebraic bisimulation

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

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

Theorem [Rutten'98, Hermina&Jacobs'98]: $\sim_\alpha = \mathrm{gfp}(b_\alpha)$

- one can use abstract coinduction directly

Proposition [Rot, Bonchi, Bonsangue, P., 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

Coalgebraic bisimulation

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

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

Theorem [Rutten'98, Hermina&Jacobs'98]: $\sim_\alpha = \text{gfp}(b_\alpha)$

- one can use abstract coinduction directly

Proposition [Rot, Bonchi, Bonsangue, P., Rutten, Silva'13]:
b_α satisfies $(\dagger)$ iff F preserves weak pullbacks

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

- up to equivalence (almost) always comes for free

Contexts: bialgebras

What about the
up to union/linear combinations/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]

Up to context in bialgebras

In the T-algebra (X, β), the context closure of a relation can be defined as:

$$c_\beta(R) = \langle \beta \circ T\pi_1^R, \beta \circ T\pi_2^R \rangle$$

Proposition [Rot, Bonchi, Bonsangue, P., Rutten, Silva'13]:
c_β is b_α-compatible whenever (X, α, β) is a λ-bialgebra.

Corollary [Turi&Plotkin'97, Bartels'04]: In all λ-bialgebras, behavioural equivalence is a congruence.
Corollary: Up to congruence is sound in all λ-bialgebras if F preserves weak pullbacks.

4. In conclusion

Summary

Combining algebra and coalgebra makes 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, P., Silva'13])

Open question

How to handle (up-to techniques for) weak bisimilarity coalgebraically?