Coalgebra: applications in automata theory and programming language design

Alexandra Silva

Radboud Universiteit Nijmegen and CWI

Chocola meeting, 16 May 2013

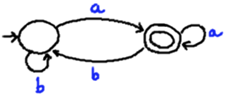
Coalgebra

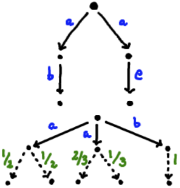
Specify and reason about systems.

Coalgebra

Specify and reason about systems.

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





Coalgebra

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

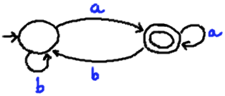
Syntax
RE, CCS, . . .

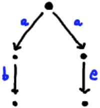
$b^*a(b^*a)^*$

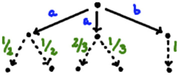
$a.b.0 + a.c.0$

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

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

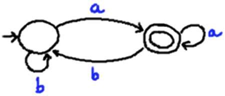
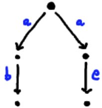
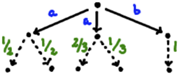






Coalgebra

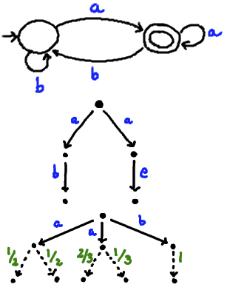
Specify and **reason** about **systems**.

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

Coalgebra

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

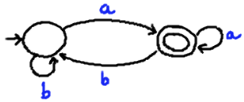
Specify	reason	systems
Syntax RE, CCS, ...	Axiomatization KA,...	state-machines e.g. DFA, LTS, PA
$b^*a(b^*a)^*$	$1 + a a^* = a^*$ $\vdots$	
$a.b.0 + a.c.0$	$P + 0 = P$ $\vdots$	
$a.(\frac{1}{2}\cdot 0 \oplus \frac{1}{2}\cdot 0) + \cdots$	$p.P \oplus p'.P = (p+p').P$ $\vdots$	

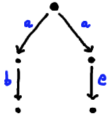


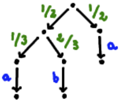
Can we do all of this uniformly in a single framework?

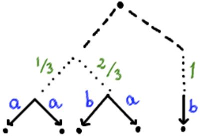
What do this things have in common?

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

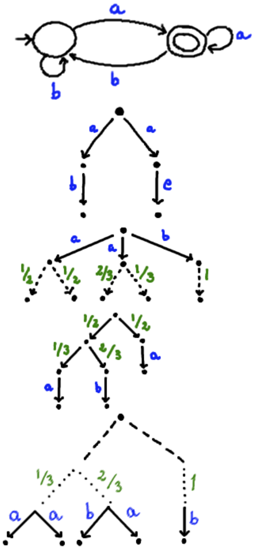








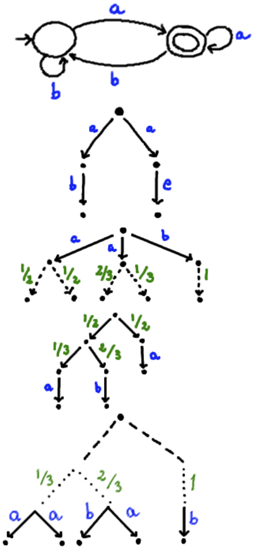
What do this things have in common?



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

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

What do this things have in common?

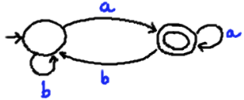


$$(S, t : S \to 2 \times S^A)$$

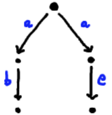
$$(S, t : S \to \mathcal{P}S^A)$$

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

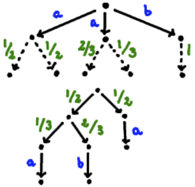
What do this things have in common?



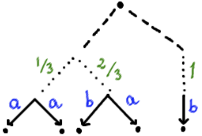
$$(S, t : S \to 2 \times S^A)$$



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

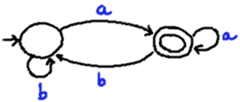


$$(S, t : S \to \mathcal{P}\mathcal{D}_\omega(S)^A)$$

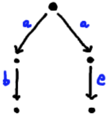


$$(S, t : S \to \mathcal{D}_\omega(S) + (A \times S) + 1)$$

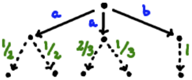
What do this things have in common?



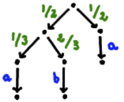
$$(S, t : S \to 2 \times S^A)$$



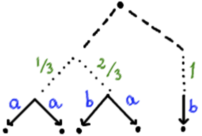
$$(S, t : S \to \mathcal{P}S^A)$$



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

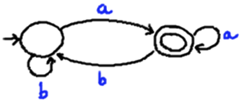


$$(S, t : S \to \mathcal{D}_\omega(S) + (A \times S) + 1)$$

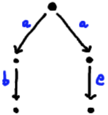


$$(S, t : S \to \mathcal{P}(\mathcal{D}_\omega(\mathcal{P}S)^A))$$

What do this things have in common?

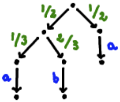


$(S, t : S \to 2 \times S^A)$

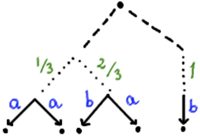


$(S, t : S \to \mathcal{P}S^A)$

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



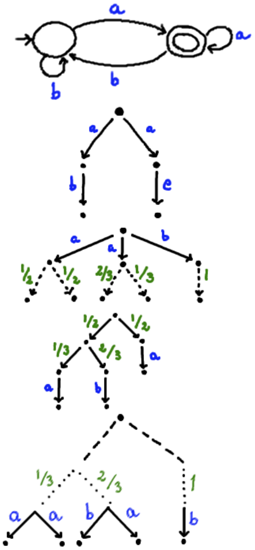
$(S, t : S \to \mathcal{D}_\omega(S) + (A \times S) + 1)$



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

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

What do this things have in common?



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

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

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

$(S, t : S \to \mathcal{D}_\omega(S) + (A \times S) + 1)$

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

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

The power of T

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

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

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^*}

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.

How about algorithms?

- Coalgebra has found its place in the semantic side of the world: operational/denotational semantics, logics, . . .
- Are there also opportunities for contributions in algorithms?

How about algorithms?

- Coalgebra has found its place in the semantic side of the world: operational/denotational semantics, logics, ...
- Are there also opportunities for contributions in algorithms?

Brzozowski's algorithm (co)algebraically

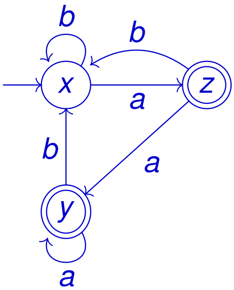
Motivation

- duality between reachability and observability (Arbib and Manes 1975): beautiful, not very well-known.
- combined use of algebra and coalgebra.
- our understanding of automata is still very limited; cf. recent research: universal automata, àtomata, weighted automata (Sakarovitch, Brzozowski, . . .)

Motivation

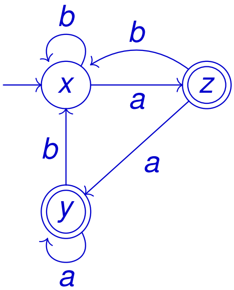
- duality between reachability and observability (Arbib and Manes 1975): beautiful, not very well-known.
- combined use of algebra and coalgebra.
- our understanding of automata is still very limited;
  cf. recent research: universal automata, àtomata, weighted automata (Sakarovitch, Brzozowski, . . .)
- joint work with Bonchi, Bonsangue, Rutten (Dexter's festschrift 2012)

Brzozowski algorithm (by example)



- initial state: x
- final states: y and z
- $L(x) \;=\; \{a, b\}^* \, a$

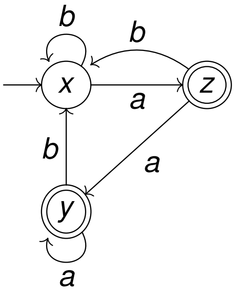
Brzozowski algorithm (by example)



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

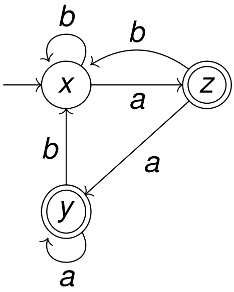
Reversing the automaton: $rev(X)$

$X =$

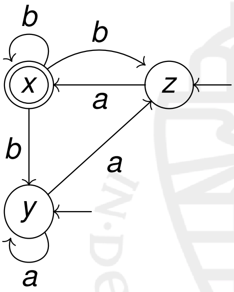


Reversing the automaton: *rev*(X)

$X =$

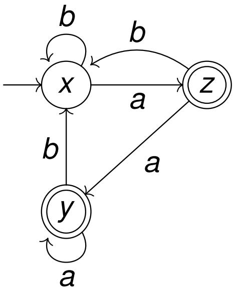


$rev(X) =$

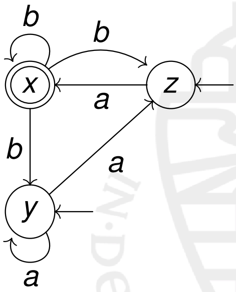


Reversing the automaton: *rev*(X)

$X =$



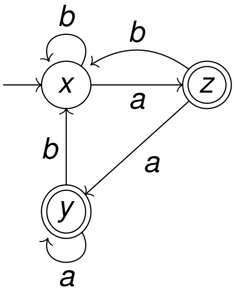
rev$(X) =$



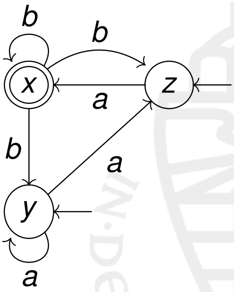
- transitions are reversed
- initial states $\Leftrightarrow$ final states

Reversing the automaton: *rev*(X)

$X =$

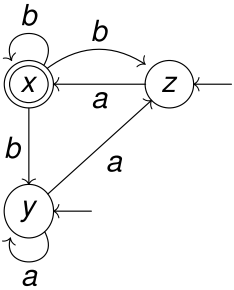


$rev(X) =$

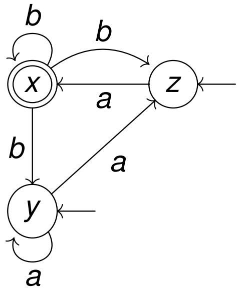


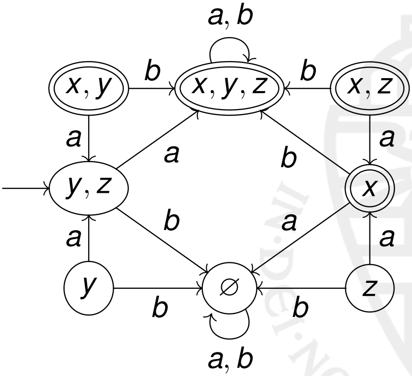
- transitions are reversed
- initial states $\Leftrightarrow$ final states
- $rev(X)$ is non-deterministic

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

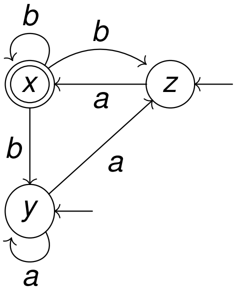


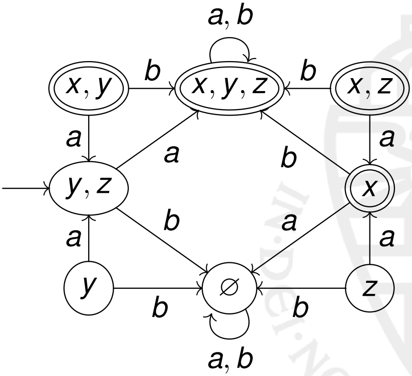
Making it deterministic again: $det(rev(X))$





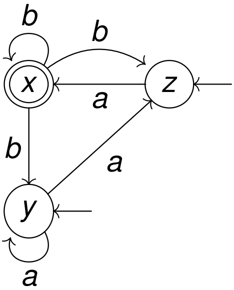
Making it deterministic again: $det(rev(X))$

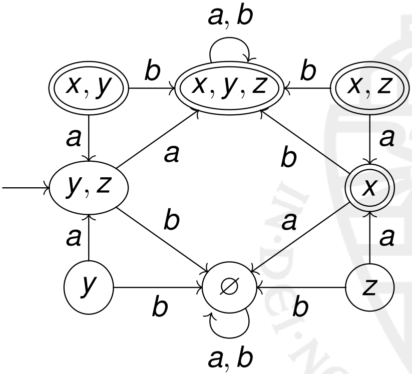




- new state space: $2^X = \{ V \mid V \subseteq \{x, y, z\} \}$

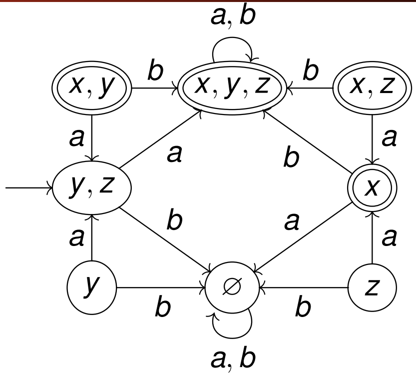
Making it deterministic again: *det*(*rev*(*X*))



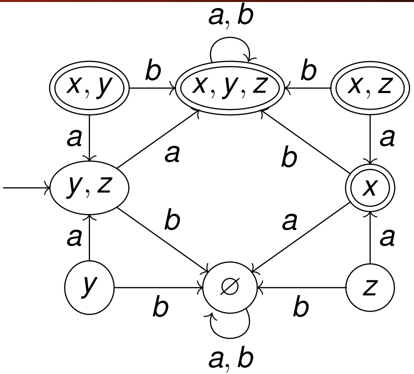


- new state space: $2^X = \{ V \mid V \subseteq \{x, y, z\} \}$
- $V \xrightarrow{a} W \qquad W = \{ w \mid \ v \xrightarrow{a} w \ , v \in V \}$

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



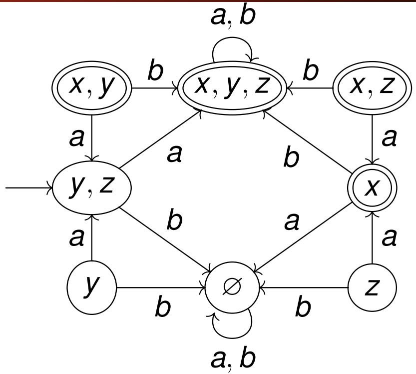
The automaton *det*(*rev*(*X*)) . . .



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

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

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!

Today's Theorem

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

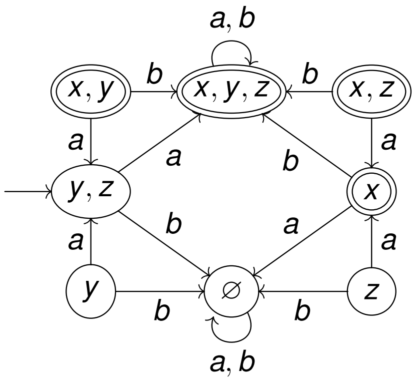
Today's Theorem

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

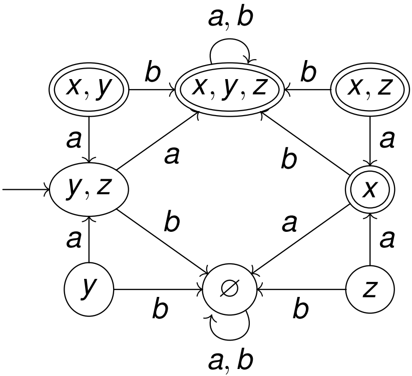
then: $det(rev(X))$ is minimal and

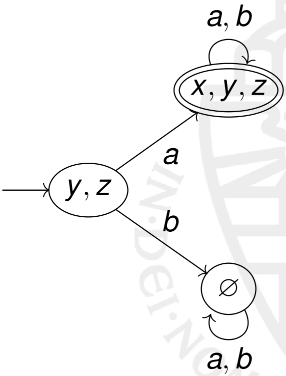
$$L(\, det(rev(X))\,) \;=\; reverse(\, L(X)\,)$$

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



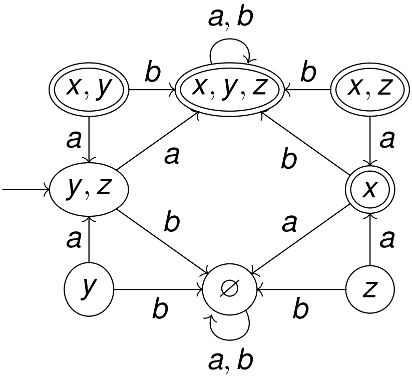
Taking the reachable part of $det(rev(X))$

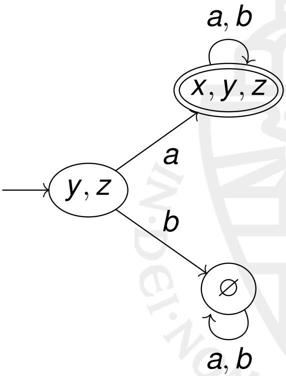




- $reach(det(rev(X)))$

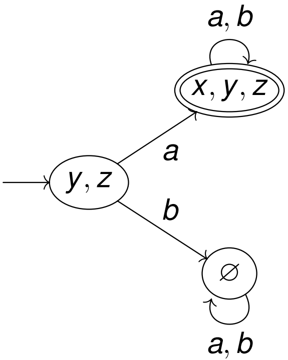
Taking the reachable part of *det*(*rev*(*X*))



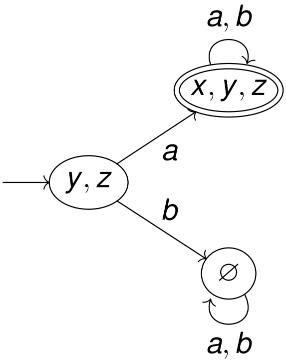


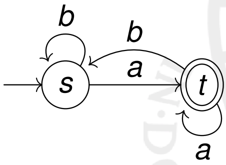
- $reach(det(rev(X)))$ is reachable (by construction)

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

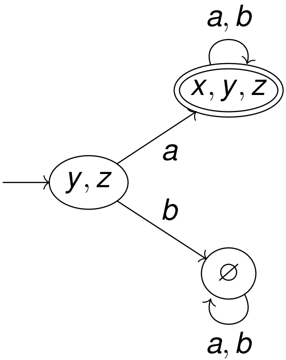


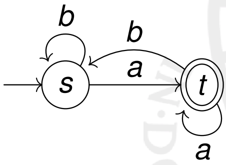
Repeating everything, now for *reach*(*det*(*rev*(*X*)))





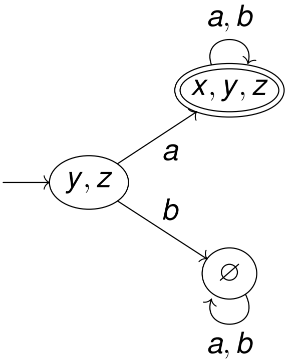
Repeating everything, now for *reach*(*det*(*rev*(*X*)))

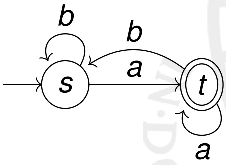




- . . . gives us $reach(det(rev(reach(det(rev(X))))))$

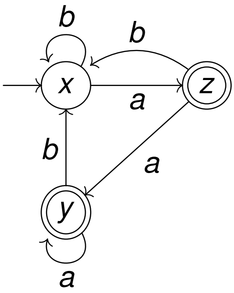
Repeating everything, now for *reach*(*det*(*rev*(*X*)))



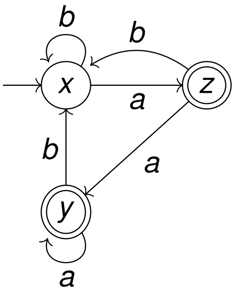


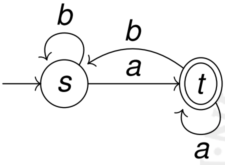
- . . . gives us *reach*(*det*(*rev*(*reach*(*det*(*rev*(*X*))))))
- which is (reachable and) minimal and accepts $\{a, b\}^*\, a$.

All in all: Brzozowski's algorithm

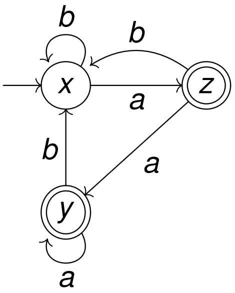


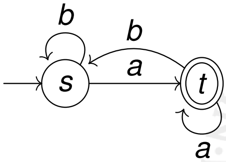
All in all: Brzozowski's algorithm





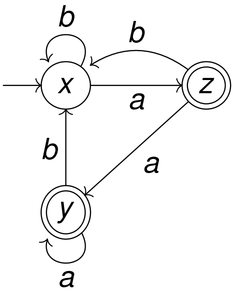
All in all: Brzozowski's algorithm

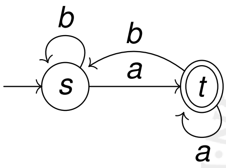




- X is reachable and accepts $\{a, b\}^* \, a$

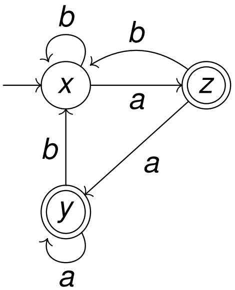
All in all: Brzozowski's algorithm

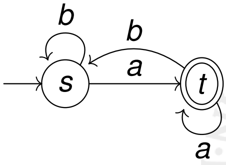




- X is reachable and accepts $\{a, b\}^* \, a$
- $reach(det(rev(reach(det(rev(X))))))$ also accepts $\{a, b\}^* \, a$

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!!

Goal of the day

- Correctness of Brzozowski's algorithm (co)algebraically
- Generalizations to other types of automata

(Co)algebra

algebras:
$$\begin{tikzcd} F(X) \arrow[d, "f"'] \\ X \end{tikzcd}$$

coalgebras:
$$\begin{tikzcd} X \arrow[d, "f"'] \\ F(X) \end{tikzcd}$$

Examples of algebras

$$\begin{tikzcd} \mathbb{N} \times \mathbb{N} \arrow[d, "+"'] \\ \mathbb{N} \end{tikzcd}$$

Examples of algebras

$$
\begin{tikzcd}
\mathbb{N} \times \mathbb{N} \arrow[d, "+"'] \\
\mathbb{N}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
1 + \mathbb{N} \arrow[d, "{[0, S]}"'] \\
\mathbb{N}
\end{tikzcd}
\equiv
\begin{tikzcd}
1 \arrow[dr, "0"'] & & \mathbb{N} \arrow[dl, "S"] \\
& \mathbb{N} &
\end{tikzcd}
\equiv
\begin{tikzcd}
1 \arrow[dr, "0"'] & \\
& \mathbb{N} \arrow[d, "S"] \\
& \mathbb{N}
\end{tikzcd}
$$

Examples of coalgebras

$$\begin{tikzcd} X \arrow[d, "t"'] \\ \mathcal{P}(A \times X) \end{tikzcd} \qquad x \xrightarrow{a} y \quad \leftrightarrow \quad \langle a, y \rangle \in t(x)$$

Examples of coalgebras

$$\begin{tikzcd} X \arrow[d, "t"'] \\ \mathcal{P}(A \times X) \end{tikzcd} \qquad x \xrightarrow{a} y \quad \leftrightarrow \quad \langle a, y \rangle \in t(x)$$

$$\begin{tikzcd} X \arrow[d, "\langle Left, label, Right \rangle"'] \\ X \times A \times X \end{tikzcd}$$

Examples of coalgebras

$$
\begin{tikzcd}
2^\omega \arrow[d, "\langle \text{head}, \text{tail} \rangle"'] \\
2 \times 2^\omega
\end{tikzcd}
\equiv
\begin{tikzcd}
& 2^\omega \arrow[dl, "\text{head}"'] \arrow[dr, "\text{tail}"] & \\
2 & & 2^\omega
\end{tikzcd}
\equiv
\begin{tikzcd}
& 2 \\
2^\omega \arrow[ur, "\text{head}"'] \arrow[d, "\text{tail}"] & \\
2^\omega &
\end{tikzcd}
$$

$$\text{head}((b_0, b_1, b_2, \ldots)) = b_0$$

$$\text{tail}((b_0, b_1, b_2, \ldots)) = (b_1, b_2, b_3 \ldots)$$

Homomorphisms

$$\begin{tikzcd} F(X) \arrow[r, "F(h)"] \arrow[d, "f"'] & F(Y) \arrow[d, "g"] \\ X \arrow[r, "h"'] & Y \end{tikzcd}$$

Homomorphisms

$$\begin{tikzcd} F(X) \arrow[r, "F(h)"] & F(Y) \\ X \arrow[u, leftarrow, "f"'] \arrow[r, "h"'] & Y \arrow[u, leftarrow, "g"'] \end{tikzcd}$$

$$\begin{tikzcd} X \arrow[r, "h"] \arrow[d, "f"'] & Y \arrow[d, "g"] \\ F(X) \arrow[r, "F(h)"'] & F(Y) \end{tikzcd}$$

Initiality, finality

$$
\begin{tikzcd}
F(A) \arrow[r, dashed, "F(h)"] \arrow[d, "\alpha"'] & F(X) \arrow[d, "f"] \\
A \arrow[r, dashed, "\exists !\, h"'] & X
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, dashed, "\exists !\, h"] \arrow[d, "f"'] & Z \arrow[d, "\beta"] \\
F(X) \arrow[r, dashed, "F(h)"'] & F(Z)
\end{tikzcd}
$$

Initiality, finality

$$
\begin{tikzcd}
F(A) \arrow[r, dashed, "F(h)"] \arrow[d, "\alpha"'] & F(X) \arrow[d, "f"] \\
A \arrow[r, dashed, "\exists!\, h"'] & X
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
X \arrow[r, dashed, "\exists!\, h"] \arrow[d, "f"'] & Z \arrow[d, "\beta"] \\
F(X) \arrow[r, dashed, "F(h)"'] & F(Z)
\end{tikzcd}
$$

- initial algebras $\leftrightarrow$ induction

Initiality, finality

$$\begin{tikzcd} F(A) \arrow[r, dashed, "F(h)"] \arrow[d, "\alpha"'] & F(X) \arrow[d, "f"] \\ A \arrow[r, dashed, "\exists ! h"'] & X \end{tikzcd} \qquad \begin{tikzcd} X \arrow[r, dashed, "\exists ! h"] \arrow[d, "f"'] & Z \arrow[d, "\beta"] \\ F(X) \arrow[r, dashed, "F(h)"'] & F(Z) \end{tikzcd}$$

- initial algebras $\leftrightarrow$ induction
- final coalgebras $\leftrightarrow$ coinduction

Automata, (co)algebraically

- Automata are complicated structures:

  part of them is algebra - part of them is coalgebra

Automata, (co)algebraically

- Automata are complicated structures:

  part of them is algebra - part of them is coalgebra
- (. . . in two different ways . . .)

A deterministic automaton

$$
\begin{tikzcd}
1 \arrow[dr, "i"] & & 2 \\
& X \arrow[ur, "f"] \arrow[d, "t"] & \\
& X^A &
\end{tikzcd}
$$

A deterministic automaton

$$
\begin{tikzcd}
1 \arrow[dr, "i"] & & 2 \\
& X \arrow[ur, "f"] \arrow[d, "t"] & \\
& X^A &
\end{tikzcd}
$$

where

$$1 = \{0\} \qquad 2 = \{0,1\} \qquad X^A = \{\, g \mid g : A \to X \,\}$$

$$x \xrightarrow{a} y \quad \leftrightarrow \quad t(x)(a) = y$$

$i(0) \in X$ is the initial state

x is final (or accepting) $\leftrightarrow$ $f(x) = 1$

Automata: algebra or coalgebra?

- initial state: algebraic – final states: coalgebraic

$$
\begin{tikzcd}
1 \arrow[dr, "i"] & & 2 \\
& X \arrow[ur, "f"] &
\end{tikzcd}
$$

Automata: algebra or coalgebra?

- initial state: algebraic – final states: coalgebraic

$$\begin{tikzcd} 1 \arrow[dr, "i"] & & 2 \\ & X \arrow[ur, "f"'] & \end{tikzcd}$$

- transition function: both algebraic and coalgebraic

$$\frac{X \xrightarrow{t} X^A}{\dfrac{X \longrightarrow (A \longrightarrow X)}{X \times A \xrightarrow{t} X}}$$

Automata: algebra and coalgebra!

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[ur, "f"] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

Automata: algebra and coalgebra!

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[d, "t"] \arrow[ur, "f"] & 2^{A^*} \arrow[u, "\epsilon ?"'] \arrow[d, "\beta"'] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

To take home: this picture!! . . .

Automata: algebra and coalgebra!

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[ur, "f"] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

To take home: this picture!! . . . which we'll explain next . . .

The "automaton" of languages

$$
\begin{tikzcd}
2 \\
2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(2^{A^*})^A
\end{tikzcd}
$$

$$\epsilon?(L) = 1 \leftrightarrow \epsilon \in L$$

$$2^{A^*} = \{g \mid g : A^* \to 2\} \cong \{L \mid L \subseteq A^*\}$$

$$\beta(L)(a) = L_a = \{w \in A^* \mid a \cdot w \in L\}$$

The "automaton" of languages

$$\begin{tikzcd} 2 \\ 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"'] \\ (2^{A^*})^A \end{tikzcd}$$

$$\epsilon?(L) = 1 \leftrightarrow \epsilon \in L$$

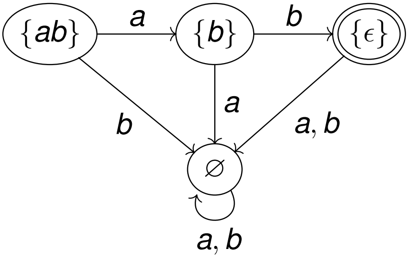
$$2^{A^*} = \{g \mid g : A^* \to 2\} \cong \{L \mid L \subseteq A^*\}$$

$$\beta(L)(a) = L_a = \{w \in A^* \mid a \cdot w \in L\}$$

- We say "automaton": it does not have an initial state.

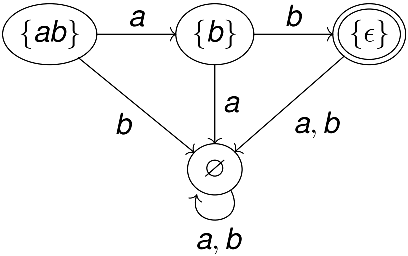
The automaton of languages

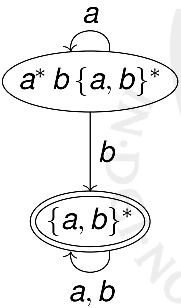
- transitions: $L \xrightarrow{a} L_a$ where $L_a = \{w \in A^* \mid a \cdot w \in L\}$
- for instance:



The automaton of languages

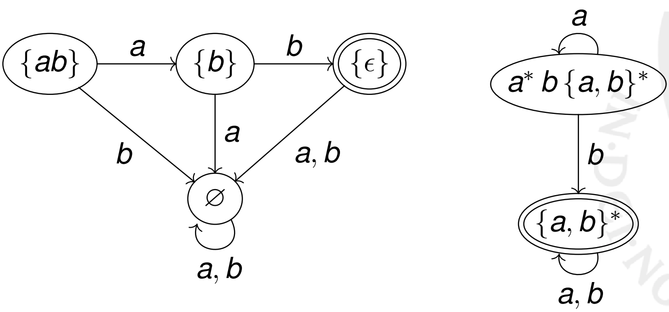
- transitions: $L \xrightarrow{a} L_a$ where $L_a = \{w \in A^* \mid a \cdot w \in L\}$
- for instance:





The automaton of languages

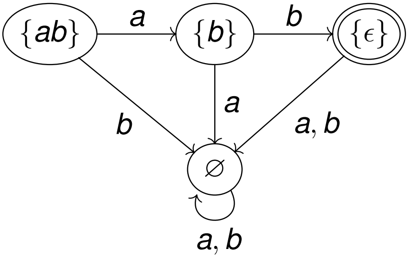
- transitions: $L \xrightarrow{a} L_a$ where $L_a = \{w \in A^* \mid a \cdot w \in L\}$
- for instance:

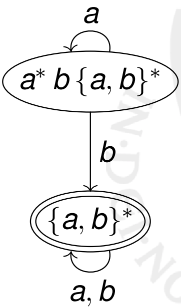


- note: every state L accepts . . .

The automaton of languages

- transitions: $L \xrightarrow{a} L_a$ where $L_a = \{w \in A^* \mid a \cdot w \in L\}$
- for instance:





- note: every state L accepts . . . . . . the language L !!

The automaton of languages is . . . final

$$\begin{tikzcd}
& 2 \\
X \arrow[ur, "f"] \arrow[r, dashed, "\exists!\, o"'] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

$$\begin{aligned} o(x) &= \{w \in A^* \mid f(x_w) = 1\} \\ &= \text{the language accepted by } x \end{aligned}$$

The automaton of languages is . . . final

$$\begin{tikzcd} & 2 \\ X \arrow[ur, "f"] \arrow[r, dashed, "\exists!\, o"'] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\ X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A \end{tikzcd}$$

$$o(x) = \{w \in A^* \mid f(x_w) = 1\}$$

$$= \text{the language accepted by } x$$

where: x_w is the state reached after inputting the word w,

and: $o^A(g) = o \circ g$, all $g \in X^A$.

Back to today's picture

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[ur, "f"] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

Back to today's picture

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[d, "t"] \arrow[ur, "f"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

On the right: final coalgebra

Back to today's picture

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[d, "t"] \arrow[ur, "f"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

On the right: final coalgebra

On the left: initial algebra . . .

The "automaton" of words

$$\begin{array}{c} 1 \\ \downarrow{\scriptstyle \epsilon} \\ A^* \\ \downarrow{\scriptstyle \alpha} \\ (A^*)^A \end{array}$$

ϵ is initial state

$$\alpha(w)(a) = w \cdot a$$

that is, transitions: $w \xrightarrow{a} w \cdot a$

The automaton of words is . . . initial

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, dashed, "\exists!\, r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A
\end{tikzcd}$$

$$\begin{aligned}
i \in X &= \text{initial state} \\
&\ \text{(to be precise: } i(0)\text{)} \\
r(w) &= i_w \\
&= \text{the state reached from } i \\
&\ \text{after inputting } w
\end{aligned}$$

- Proof: easy exercise.
- Proof: formally, because A^* is an initial $1 + A \times (-)$-algebra!

Duality

- Reachability and observability are dual:

  Arbib and Manes, 1975.

- (here observable = minimal)

Reachability and observability

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[ur, "f"] \arrow[r, dashed, "o"'] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

Reachability and observability

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[d, "t"] \arrow[ur, "f"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

$$\begin{aligned}
r(w) &= \text{state reached on input } w \\
o(x) &= \text{language accepted by } x
\end{aligned}$$

Reachability and observability

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[ur, "f"] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
$$

$$
\begin{aligned}
r(w) &= \text{state reached on input } w \\
o(x) &= \text{language accepted by } x
\end{aligned}
$$

- We call X reachable if r is surjective.

Reachability and observability

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[ur, "f"] \arrow[r, dashed, "o"'] \arrow[d, "t"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}$$

$$r(w) = \text{state reached on input } w$$

$$o(x) = \text{language accepted by } x$$

- We call X reachable if r is surjective.
- We call X observable (= minimal) if o is injective.

Reversing the automaton

- Reachability $\leftrightarrow$ observability
- Being precise about homomorphisms is crucial.
- Forms the basis for proof Brzozowski's algorithm.

Powerset construction

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

Powerset construction

$$2^{(-)} : \quad \begin{tikzcd} V \arrow[d, "g"'] \\ W \end{tikzcd} \quad \mapsto \quad \begin{tikzcd} 2^V \\ 2^W \arrow[u, "2^g"'] \end{tikzcd}$$

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

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

Powerset construction

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

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 !!

Powerset construction

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

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 !!
- Note: if g is surjective, then 2^g is injective.

Reversing transitions

$$
\begin{tikzcd}
X \arrow[d, "t"'] \\
X^A
\end{tikzcd}
$$

Reversing transitions

$$
\begin{array}{c||c}
\begin{tikzcd} X \arrow[d, "t"'] \\ X^A \end{tikzcd} & \begin{tikzcd} X \times A \arrow[d] \\ X \end{tikzcd}
\end{array}
$$

Reversing transitions

$$
\begin{array}{c|c}
\begin{tikzcd} X \arrow[d, "t"'] \\ X^A \end{tikzcd} & \begin{tikzcd} X \times A \arrow[d] \\ X \end{tikzcd}
\end{array}
\quad \xmapsto{\;2^{(-)}\;} \quad
\begin{tikzcd} 2^{X \times A} \\ 2^X \arrow[u] \end{tikzcd}
$$

Reversing transitions

$$
\begin{array}{c|c}
X & X \times A \\
t \downarrow & \downarrow \\
X^A & X
\end{array}
\quad \overset{2^{(-)}}{\longmapsto} \quad
\begin{array}{c|c}
2^{X \times A} & (2^X)^A \\
\uparrow & \uparrow \\
2^X & 2^X
\end{array}
$$

Reversing transitions

$$
\begin{array}{c|c}
X & X \times A \\
\Big\downarrow t & \Big\downarrow \\
X^A & X
\end{array}
\quad \overset{2^{(-)}}{\longmapsto} \quad
\begin{array}{c|c|c}
2^{X \times A} & (2^X)^A & 2^X \\
\Big\uparrow & \Big\uparrow & \Big\downarrow 2^t \\
2^X & 2^X & (2^X)^A
\end{array}
$$

Initial $\leftrightarrow$ final

$$\begin{tikzcd} 1 \arrow[dr, "i"] & \\ & X \end{tikzcd}$$

Initial $\leftrightarrow$ final

$$
\begin{tikzcd}
1 \arrow[dr, "i"] & & & & & 2 \\
& X & \arrow[r, mapsto, "2^{(-)}"] & {} & 2^X \arrow[ur, "2^i"] &
\end{tikzcd}
$$

Initial $\leftrightarrow$ final

$$1 \xrightarrow{i} X \qquad \xmapsto{2^{(-)}} \qquad 2^X \xrightarrow{2^i} 2$$

$$X \xrightarrow{f} 2$$

Initial $\leftrightarrow$ final

$$\left(1 \xrightarrow{i} X \right) \quad \xmapsto{2^{(-)}} \quad \left(2^X \xrightarrow{2^i} 2 \right)$$

$$\left(X \xrightarrow{f} 2 \right) \quad \longleftrightarrow \quad \left(1 \xrightarrow{f} 2^X \right)$$

Reversing the entire automaton

$$\begin{tikzcd} 1 \arrow[dr, "i"] & & 2 \\ & X \arrow[ur, "f"] \arrow[d, "t"] & \\ & X^A & \end{tikzcd}$$

Reversing the entire automaton

$$\begin{tikzcd}
1 \arrow[dr, "i"] & & 2 \\
& X \arrow[ur, "f"] \arrow[d, "t"] & \\
& X^A &
\end{tikzcd}
\qquad \stackrel{2^{(-)}}{\longmapsto} \qquad
\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"] \arrow[d, "2^t"] & \\
& (2^X)^A &
\end{tikzcd}$$

Reversing the entire automaton

$$
\begin{tikzcd}
1 \arrow[dr, "i"] & & 2 \\
& X \arrow[ur, "f"] \arrow[d, "t"] & \\
& X^A &
\end{tikzcd}
\qquad \xmapsto{2^{(-)}} \qquad
\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"] \arrow[d, "2^t"] & \\
& (2^X)^A &
\end{tikzcd}
$$

- Initial and final are exchanged . . .

Reversing the entire automaton

$$
\begin{tikzcd}
1 \arrow[dr, "i"] & & 2 \\
& X \arrow[ur, "f"] \arrow[d, "t"] & \\
& X^A &
\end{tikzcd}
\quad \xmapsto{2^{(-)}} \quad
\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"'] \arrow[d, "2^t"] & \\
& (2^X)^A &
\end{tikzcd}
$$

- Initial and final are exchanged . . .
- transitions are reversed . . .

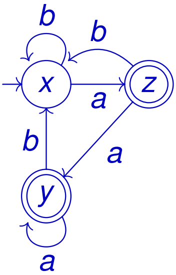
Reversing the entire automaton

$$
\begin{tikzcd}
1 \arrow[dr, "i"] & & 2 \\
& X \arrow[ur, "f"] \arrow[d, "t"] & \\
& X^A &
\end{tikzcd}
\qquad \xmapsto{\;2^{(-)}\;} \qquad
\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"] \arrow[d, "2^t"] & \\
& (2^X)^A &
\end{tikzcd}
$$

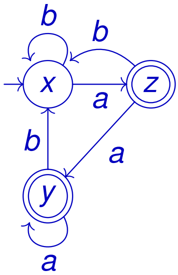
- Initial and final are exchanged . . .
- transitions are reversed . . .
- and the result is again deterministic!

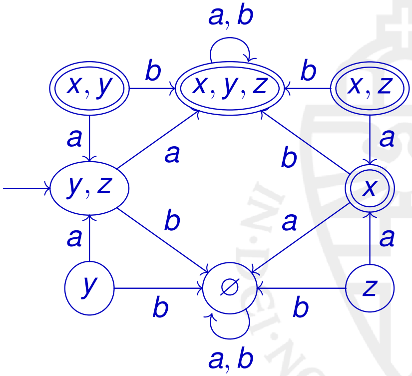
Our previous example

$X =$



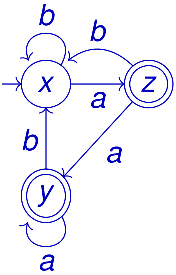
Our previous example

$X =$ 

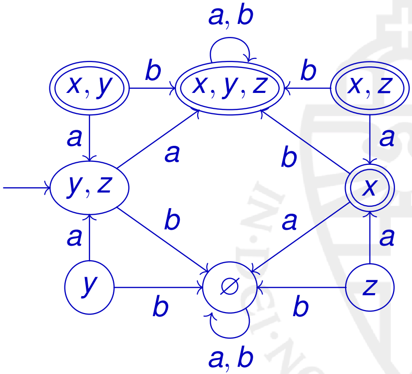
$2^X =$ 

Our previous example

$X =$



$2^X =$



- Note that X has been reversed and determinized:

$$2^X = det(rev(X))$$

Proving today's Theorem

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

Proving today's Theorem

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

then: 2^X ($=$ $det(rev(X))$) is minimal/observable and

$$L(2^X) = reverse(L(X))$$

Proof: by reversing $A^* \xrightarrow{r} X$

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}
$$

Proof: by reversing $A^* \xrightarrow{r} X$

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & & & 2 \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] & \arrow[r, mapsto, "2^{(-)}"] & 2^X \arrow[ur, "2^i"] \arrow[r, "2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "2^\epsilon"'] \arrow[d, "2^\alpha"] \\
(A^*)^A \arrow[r] & X^A & & (2^X)^A \arrow[r] & (2^{A^*})^A
\end{tikzcd}
$$

Proof: by reversing $A^* \xrightarrow{r} X$

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}
\quad \overset{2^{(-)}}{\longmapsto} \quad
\begin{tikzcd}
 & 2 \\
2^X \arrow[ur, "2^i"] \arrow[r, "2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "2^\epsilon"'] \arrow[d, "2^\alpha"] \\
(2^X)^A \arrow[r] & (2^{A^*})^A
\end{tikzcd}
$$

- X becomes 2^X

Proof: by reversing $A^* \xrightarrow{r} X$

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}
\quad \xmapsto{2^{(-)}} \quad
\begin{tikzcd}
 & 2 \\
2^X \arrow[ur, "2^i"] \arrow[r, "2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "2^\epsilon"'] \arrow[d, "2^\alpha"] \\
(2^X)^A \arrow[r] & (2^{A^*})^A
\end{tikzcd}
$$

- X becomes 2^X
- initial automaton A^* becomes (almost) final automaton 2^{A^*}

Proof: by reversing $A^* \xrightarrow{r} X$

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}
\qquad \overset{2^{(-)}}{\longmapsto} \qquad
\begin{tikzcd}
 & 2 \\
2^X \arrow[ur, "2^i"] \arrow[r, "2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "2^\epsilon"'] \arrow[d, "2^\alpha"] \\
(2^X)^A \arrow[r] & (2^{A^*})^A
\end{tikzcd}
$$

- X becomes 2^X
- initial automaton A^* becomes (almost) final automaton 2^{A^*}
- r is surjective $\Rightarrow$ 2^r is injective

Reachable becomes observable

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}$$

Reachable becomes observable

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}
\quad \overset{2^{(-)}}{\longmapsto} \quad
\begin{tikzcd}
 & 2 & \\
2^X \arrow[ur, "2^i"] \arrow[r, "2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "2^\epsilon"'] \arrow[r, dashed, "rev"'] \arrow[d, "2^\alpha"] & 2^{A^*} \arrow[ul, "\epsilon?"'] \arrow[d, "\beta"] \\
(2^X)^A \arrow[r] & (2^{A^*})^A \arrow[r, dashed] & (2^{A^*})^A
\end{tikzcd}
$$

Reachable becomes observable

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}
\quad \xmapsto{2^{(-)}} \quad
\begin{tikzcd}
 & 2 & \\
2^X \arrow[ur, "2^i"] \arrow[r, "2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "2^\epsilon"'] \arrow[r, dashed, "rev"'] \arrow[d, "2^\alpha"] & 2^{A^*} \arrow[ul, "\epsilon?"'] \arrow[d, "\beta"] \\
(2^X)^A \arrow[r] & (2^{A^*})^A \arrow[r, dashed] & (2^{A^*})^A
\end{tikzcd}
$$

- If r is surjective then (2^r and hence) $rev \circ 2^r$ is injective.

Reachable becomes observable

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & \\
A^* \arrow[r, "r"'] \arrow[d, "\alpha"'] & X \arrow[d, "t"] \\
(A^*)^A \arrow[r] & X^A
\end{tikzcd}
\quad \xmapsto{2^{(-)}} \quad
\begin{tikzcd}
 & 2 & \\
2^X \arrow[ur, "2^i"] \arrow[r, "2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "2^\epsilon"'] \arrow[r, dashed, "rev"'] \arrow[d, "2^\alpha"] & 2^{A^*} \arrow[ul, "\epsilon?"'] \arrow[d, "\beta"] \\
(2^X)^A \arrow[r] & (2^{A^*})^A \arrow[r, dashed] & (2^{A^*})^A
\end{tikzcd}
$$

- If r is surjective then (2^r and hence) $rev \circ 2^r$ is injective.
- That is, 2^X is observable (= minimal).

Summarizing

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[ur, "f"] \arrow[d, "t"] & \\
(A^*)^A \arrow[r, dashed] & X^A &
\end{tikzcd}$$

Summarizing

$$\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[ur, "f"] \arrow[d, "t"'] & \\
(A^*)^A \arrow[r, dashed] & X^A &
\end{tikzcd}$$

$$\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"] \arrow[r, dashed, "rev \circ 2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
& (2^X)^A \arrow[r, dashed] & (2^{A^*})^A
\end{tikzcd}$$

Summarizing

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[ur, "f"] \arrow[d, "t"'] & \\
(A^*)^A \arrow[r, dashed] & X^A &
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"] \arrow[r, dashed, "rev \circ 2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
& (2^X)^A \arrow[r, dashed] & (2^{A^*})^A
\end{tikzcd}
$$

- If: X is reachable, i.e., r is surjective

Summarizing

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[ur, "f"] \arrow[d, "t"'] & \\
(A^*)^A \arrow[r, dashed] & X^A &
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"] \arrow[r, dashed, "rev \circ 2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"'] \\
& (2^X)^A \arrow[r, dashed] & (2^{A^*})^A
\end{tikzcd}
$$

- If: X is reachable, i.e., r is surjective

  then: $rev \circ 2^r$ is injective, i.e., 2^X is observable = minimal.

Summarizing

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[ur, "f"] \arrow[d, "t"'] & \\
(A^*)^A \arrow[r, dashed] & X^A &
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \arrow[dr, "f"] & & 2 \\
& 2^X \arrow[ur, "2^i"] \arrow[r, dashed, "rev \circ 2^r"'] \arrow[d, "2^t"'] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\beta"] \\
& (2^X)^A \arrow[r, dashed] & (2^{A^*})^A
\end{tikzcd}
$$

- If: X is reachable, i.e., r is surjective

  then: $rev \circ 2^r$ is injective, i.e., 2^X is observable = minimal.
- And: $rev(2^r(f)) = rev(o(i))$, i.e., $L(\,2^X\,) \;=\; reverse(\,L(X)\,)$

Corollary: Brzozowski's algorithm

- X becomes 2^X, accepting $\mathit{reverse}(L(X))$

Corollary: Brzozowski's algorithm

- X becomes 2^X, accepting *reverse*$(L(X))$
- take reachable part: $Y = \mathit{reachable}(2^X)$

Corollary: Brzozowski's algorithm

- X becomes 2^X, accepting *reverse*$(L(X))$
- take reachable part: $Y = \mathit{reachable}(2^X)$
- Y becomes 2^Y, which is minimal and accepts

$$\mathit{reverse}(\mathit{reverse}(L(X))) = L(X)$$

Generalizations

$$
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & 2 \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[d, "t"] \arrow[ur, "f"] & 2^{A^*} \arrow[u, "\epsilon?"'] \arrow[d, "\zeta"] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (2^{A^*})^A
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \arrow[d, "\epsilon"'] \arrow[dr, "i"] & & B \\
A^* \arrow[r, dashed, "r"'] \arrow[d, "\alpha"'] & X \arrow[r, dashed, "o"'] \arrow[d, "t"] \arrow[ur, "f"] & B^{A^*} \arrow[u] \arrow[d] \\
(A^*)^A \arrow[r, dashed, "r^A"'] & X^A \arrow[r, dashed, "o^A"'] & (B^{A^*})^A
\end{tikzcd}
$$

- A Brzozowski minimization algorithm for Moore automata.

$$B^X = \{\varphi \mid \varphi \colon X \to B\} \qquad B^f(\varphi) = \varphi \circ f$$

Further generalizations

- Moore automata generalization: uniform algorithm for decorated traces and must testing (joint work with Bonchi, Caltais and Pous);
- Further generalizations to non-deterministic and weighted automata.

Further generalizations

- Moore automata generalization: uniform algorithm for decorated traces and must testing (joint work with Bonchi, Caltais and Pous);
- Further generalizations to non-deterministic and weighted automata.

Coalgebra is not only semantics but also algorithms!

CoCaml: Programming with Coinductive Types

(joint work with Jean-Baptiste Jeannin and Dexter Kozen)

Computing with Coalgebraic Data

- Inductive datatypes and functions on those are well-understood; coinductive datatypes often considered difficult to handle, not many programming languages offer the constructs for them.
- OCaml offers the possibility of defining coinductive datatypes, but the means to define recursive functions on them are limited.
- Often the obvious definitions do not halt or provide the wrong solution.
- Even so, there are often perfectly good solutions (examples forthcoming!)
- We show how to extend the language to allow it!

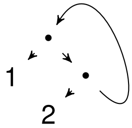
Motivating example

```
type list = N | C of int * list

let rec ones = C(1, ones);; 1,1,1,1,...
let rec alt = C(1, C(2, alt));; 1,2,1,2,...
```

Motivating example

```
type list = N | C of int * list

let rec ones = C(1, ones);; 1,1,1,1,...
let rec alt = C(1, C(2, alt));; 1,2,1,2,...
```

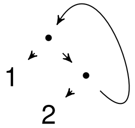
Infinite lists but… regular:



Motivating example

```
type list = N | C of int * list

let rec ones = C(1, ones);; 1,1,1,1,...
let rec alt = C(1, C(2, alt));; 1,2,1,2,...
```

Infinite lists but. . . regular:



A simple function:

```
let set l = match l with
| N -> N
| C(h, t) -> (insert h (set t));;
```

We expect `set ones = {1}` and `set alt = {1,2}`.

What is the problem?

- The function definition above will not halt in OCaml…
- even though it is clear what the answer should be;

What is the problem?

- The function definition above will not halt in OCaml...
- even though it is clear what the answer should be;
- Note that this is not a corecursive definition: we are not asking for a greatest solution or a unique solution in a final coalgebra,
- but rather a least solution in a different ordered domain from the one provided by the standard semantics of recursive functions.
- Standard semantics: least solution in the flat Scott domain with bottom element $\perp$ representing nontermination
- Intended semantics: least solution in a different CPO, namely $(\mathcal{P}(\mathbb{Z}), \subseteq)$ with bottom element $\varnothing$.

Motivating example c'd

We would like to use (almost) the same definition and get the intended solution. . .

```
let set l = match l with
| N -> N
| C(h, t) -> (insert h (set t));;
```

Motivating example c'd

We would like to use (almost) the same definition and get the intended solution...

```
let set l = match l with
| N -> N
| C(h, t) -> (insert h (set t));;
```

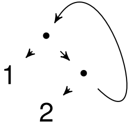
We change it to:

```
let corec[iterator(N)] set l = match l with
| N -> N
| C(h, t) -> insert h (set t);;
```

The construct `corec` with the parameter `iterator(N)` specifies to the compiler how to solve equations.

Motivating example c'd

For instance, for the infinite list `alt`:



the compiler will generate two equations:

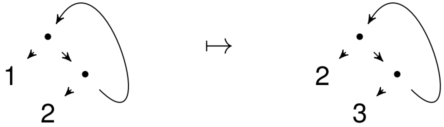
```
set(x) = insert 1 (set(y))
set(y) = insert 2 (set(x))
```

then solve them using `iterator` (least fixed point) which will produce the intended set $\{1, 2\}$.

Motivating example c'd

```
let map f = match arg with
| N -> N
| C(h, t) -> C(f(h), map(f,t));;
```

We would like: `map plusOne alt` to produce the infinite list $2, 3, 2, 3, \ldots$:



This is not a least fixed point computation anymore but rather a solution in the final coalgebra.

Another Example

Free variables of a λ-term

```
type term =
   | Var of string
   | App of term *
term
   | Lam of string *
term
```

x

$(f \ \ e)$

$\lambda x.e$

```
let rec fv = function
   | Var v -> {v}
   | App(t1,t2) -> fv t1 U fv t2
   | Lam(x,t) -> (fv t) - {x}
```

Another Example

But what about infinitary λ-terms (λ-coterms)?

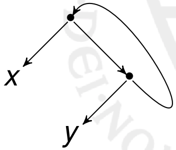
```
type term =
   | Var of string
   | App of term *
term
   | Lam of string *
term
```

x

$(f \;\; e)$

$\lambda x.e$

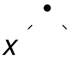
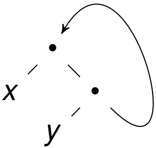
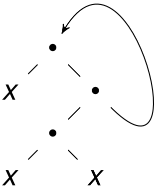
```
let rec fv = function
   | Var v -> {v}
   | App(t1,t2) -> fv t1 U fv t2
   | Lam(x,t) -> (fv t) - {x}
```



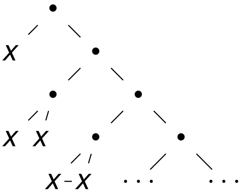

```
let rec t = App(Var "x", App(Var "y", t))
```

We would like: `fv t = {x,y}` (again LFP).

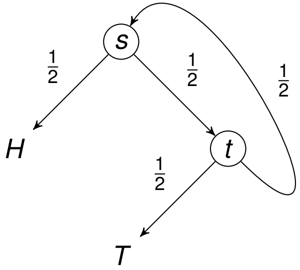
Substitution

Replace *y* by  in  to get  .

The usual semantics would infinitely unfold the term on the left, generating instead:

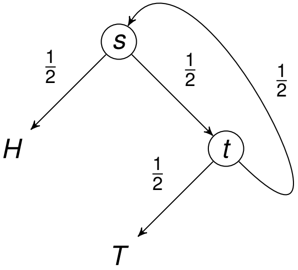


Probabilistic Protocols



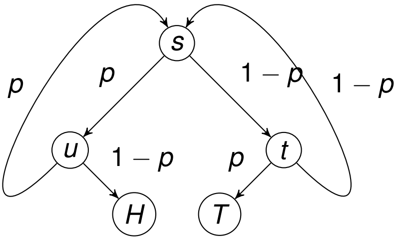
$$\mathrm{Pr}_H(s) = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \frac{1}{128} + \cdots = \frac{2}{3}$$
$$\mathrm{Pr}_H(t) = \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \cdots = \frac{1}{3}$$

Probabilistic Protocols



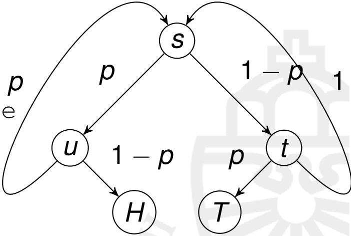
$$\mathrm{Pr}_H(s) = \frac{1}{2} + \frac{1}{2} \cdot \mathrm{Pr}_H(t)$$
$$\mathrm{Pr}_H(t) = \frac{1}{2} \cdot \mathrm{Pr}_H(s)$$

The Von Neumann Trick



$$\begin{aligned}
\Pr_H(s) &= p \cdot \Pr_H(u) + (1-p) \cdot \Pr_H(t) \\
\Pr_H(u) &= (1-p) + p \cdot \Pr_H(s) \\
\Pr_H(t) &= (1-p) \cdot \Pr_H(s)
\end{aligned}$$

The Von Neumann Trick




```
type state =
  | H
  | T
  | Flip of float * state * state

let rec pr_heads s = function
  | H -> 1.
  | T -> 0.
  | Flip(p,u,v) ->
     p *. (pr_heads u) +. (1 -. p) *. (pr_heads v)

let rec s = Flip(.345,u,t)
and u = Flip(.345,H,s)
and t = Flip(.345,T,s)

print p_heads s
```

Theoretical Foundations

- Well-founded coalgebras [Taylor 99]
- Recursive coalgebras [Adámek, Lücke, Milius 07]
- Elgot algebras [Adámek, Milius, Velebil 06]
- Corecursive algebras [Capretta, Uustalu, Vene 09]

Ingredients:

- Functor F (usually polynomial or power set)
- domain: an F-coalgebra (C, γ)
- range: an F-algebra (A, α)

$$\begin{tikzcd} C \arrow[r, "h"] \arrow[d, "\gamma"'] & A \\ FC \arrow[r, "Fh"'] & FA \arrow[u, "\alpha"'] \end{tikzcd}$$

What about Non-Well-Founded Coalgebras?

The foundations existing so far were for unique solutions; we want alternative solutions.

What about Non-Well-Founded Coalgebras?

The foundations existing so far were for unique solutions; we want alternative solutions.

$$\begin{tikzcd} C \arrow[r, "h"] \arrow[d, "\gamma"'] & A \\ FC \arrow[r, "Fh"'] & FA \arrow[u, "\alpha"'] \end{tikzcd}$$

- Even if (C, γ) is not well-founded, the diagram may still have a canonical solution, provided (A, α) comes equipped with a method for solving systems of equations
- The diagram specifies the system to be solved
- The variables are the elements of C and h is their interpretation in A

The general idea

The programmer specifies the equations as usual with an extra parameter, like in:

```
let corec[iterator(N)] set l = match l with
| N -> N
| C(h, t) -> insert h (set t);;
```

The general idea

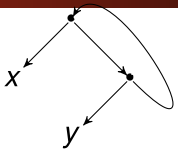
The programmer specifies the equations as usual with an extra parameter, like in:

```
let corec[iterator(N)] set l = match l with
| N -> N
| C(h, t) -> insert h (set t);;
```

The compiler generates equations and solves them using the extra parameter.

Free Variables of a λ-Coterm

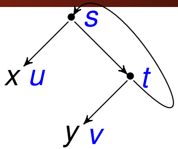
The free variables of



are $\{x, y\}$

Free Variables of a λ-Coterm

The free variables of



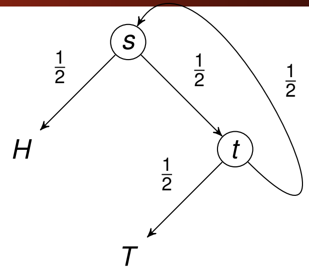
are $\{x, y\}$

$$
\begin{aligned}
fv(s) &= fv(u) \cup fv(t) \\
fv(t) &= fv(v) \cup fv(s) \\
fv(u) &= \{x\} \\
fv(v) &= \{y\}
\end{aligned}
$$

The least solution in $(\mathcal{P}(\mathrm{Var}), \subseteq)$ is $\{x, y\}$

Standard semantics: $A \cup \perp = \perp$, whereas here $A \cup \varnothing = A$

Example: Probabilistic Protocols



$$\mathrm{Pr}_H(s) = \tfrac{1}{2} + \tfrac{1}{2} \cdot \mathrm{Pr}_H(t) \qquad \mathrm{Pr}_H(t) = \tfrac{1}{2} \cdot \mathrm{Pr}_H(s)$$

- Can calculate expected running times, higher moments, outcome functions similarly
- These are all least solutions in an appropriate ordered domain—in the above example, $([0,1], \leq)$

Other Non-Well-Founded Examples

- static analysis, abstract interpretation
- *p*-adic arithmetic
- automata constructions

Implementation

- We implemented `corec` constructor which takes a solver as a parameter
- We implemented several general solvers: least fixed point, unique solution in a final coalgebra, gaussian elimination, …

Implementation

- We implemented `corec` constructor which takes a solver as a parameter
- We implemented several general solvers: least fixed point, unique solution in a final coalgebra, gaussian elimination, ...
- Solvers are implemented directly in the interpreter, as transformers from an abstract syntax tree to another abstract syntax tree.
- Future: to provide tools to manipulate the abstract syntax tree allowing programmers to easily specify their solver.

CoCaml

- CoCaml offers new program constructs and functionalities to implement functions on coinductive structures.
- Examples illustrate the need for new constructs
- New constructs enable allow definitions very much in the style of standard recursive functions.

`http://www.cs.cornell.edu/Projects/CoCaml/`

Thanks!