Two routes to automata minimization and the ways to reach it efficiently

Sylvain Lombardy[a] *and* *Jacques Sakarovitch*[b]

[a] *LaBRI, Bordeaux INP / Université de Bordeaux / CNRS*

[b] *IRIF, CNRS / Université Denis-Diderot and Telecom ParisTech*

CIAA 2018, 1 August 2018, Charlottetown (PEI)

Common knowledge in FA Theory

- Every regular language L has a *minimal DFA* (that is canonically associated with L)

Common knowledge in FA Theory

$\mathcal{A}$ DFA

- Every DFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is characteristic of $L(\mathcal{A})$

Common knowledge in FA Theory

$\mathcal{A}$ DFA

- Every DFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is characteristic of $L(\mathcal{A})$

- The minimal quotient of a DFA $\mathcal{A}$ may be *effectively computed* by a quadratic algorithm

Common knowledge in FA Theory

$\mathcal{A}$ DFA $\qquad$ n states

- Every DFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is characteristic of $L(\mathcal{A})$

- The minimal quotient of a DFA $\mathcal{A}$ may be effectively computed by the 'Moore' algorithm with a complexity $\mathrm{O}\big(n^2\big)$

Common knowledge in FA Theory

$\mathcal{A}$ DFA $\qquad$ n states

- Every DFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is characteristic of $L(\mathcal{A})$

- The minimal quotient of a DFA $\mathcal{A}$ may be effectively computed by the 'Moore' algorithm with a complexity $\mathrm{O}\left(n^2\right)$
- The minimal quotient of a DFA $\mathcal{A}$ may be effectively computed by the 'Hopcroft' algorithm with a complexity $\mathrm{O}(n \log n)$

What is this talk about

$\mathcal{A}$ NFA

What is this talk about

$\mathcal{A}$ NFA

- Every NFA $\mathcal{A}$ has a *minimal quotient*

What is this talk about

$\mathcal{A}$ NFA

- Every NFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is no more characteristic of $L(\mathcal{A})$

What is this talk about

$\mathcal{A}$ NFA

- Every NFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is no more characteristic of $L(\mathcal{A})$
- This quotient is sometimes called

  the *bisimulation minimal model* of $\mathcal{A}$

What is this talk about

$\mathcal{A}$ NFA $\quad$ n states $\quad$ m transitions

- Every NFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is no more characteristic of $L(\mathcal{A})$
- This quotient is sometimes called
  the *bisimulation minimal model* of $\mathcal{A}$

- The minimal quotient of an NFA $\mathcal{A}$ may be effectively computed by a ‘quadratic’ algorithm

What is this talk about

$\mathcal{A}$ NFA $\qquad n$ states $\qquad m$ transitions

- Every NFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is no more characteristic of $L(\mathcal{A})$
- This quotient is sometimes called
  the *bisimulation minimal model* of $\mathcal{A}$

- The minimal quotient of a NFA $\mathcal{A}$ may be effectively computed by the '*Forward*' algorithm with a complexity $\mathrm{O}(m\,n)$

What is this talk about

$\mathcal{A}$ NFA $\qquad$ n states $\qquad$ m transitions

- Every NFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is no more characteristic of $L(\mathcal{A})$
- This quotient is sometimes called
  the *bisimulation minimal model* of $\mathcal{A}$

- The minimal quotient of a NFA $\mathcal{A}$ may be effectively computed by the '*Forward*' algorithm with a complexity $\mathrm{O}(m\,n)$

- The minimal quotient of a NFA $\mathcal{A}$ may be effectively computed by the '*Backward*' algorithm with a complexity $\mathrm{O}(m\,n)$

What is this talk about

$\mathcal{A}$ NFA n states m transitions

- Every NFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is no more characteristic of $L(\mathcal{A})$
- This quotient is sometimes called
  the *bisimulation minimal model* of $\mathcal{A}$

- The minimal quotient of a NFA $\mathcal{A}$ may be effectively computed by the '*Forward*' algorithm with a complexity $\mathrm{O}(m\,n)$

- The minimal quotient of a NFA $\mathcal{A}$ may be effectively computed by the '*Backward*' algorithm with a complexity $\mathrm{O}(m\,n)$

- Under some hypotheses, the Backward algorithm may be improved into the '*Fast Backward*' algorithm with a complexity $\mathrm{O}(m \log n)$

What is this talk about

$\mathcal{A}$ WFA n states m transitions

- Every WFA $\mathcal{A}$ has a *minimal quotient*
- This quotient is no more characteristic of $L(\mathcal{A})$
- This quotient is sometimes called
  the *bisimulation minimal model* of $\mathcal{A}$

- The minimal quotient of a WFA $\mathcal{A}$ may be effectively computed by the '*Forward*' algorithm with a complexity $\mathrm{O}(m\,n)$

- The minimal quotient of a WFA $\mathcal{A}$ may be effectively computed by the '*Backward*' algorithm with a complexity $\mathrm{O}(m\,n)$

- Under some hypotheses, the Backward algorithm may be improved into the '*Fast Backward*' algorithm with a complexity $\mathrm{O}(m \log n)$

Examples of automata minimisation

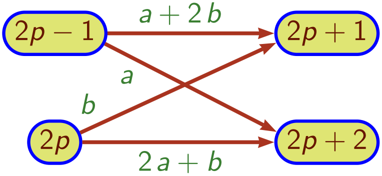
with AWALI

Benchmarks

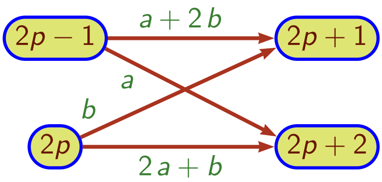
	k	14	17	20	23	26	30
	F_k	987	4 181	17 711	75 025	317 811	2 178 309
Forward	t (s)	0.42	7.37	139	–		
	$10^{-7} t/F_k^2$	4.3	4.2	4.4			
Backward	t (s)	0.010	0.045	0.257	1.36	73	257
	$10^{-7} t/k\,F_k$	7.2	6.3	7.3	7.6	6.7	7.5
Fast	t (s)	0.006	0.025	0.140	0.70	41	139
Backward	$10^{-7} t/k\,F_k$	4.2	3.5	3.9	3.8	3.5	3.7

Minimisation of $\mathcal{F}_k$

Benchmarks



Benchmarks



	n	2^{10}	2^{12}	2^{13}	2^{14}	2^{15}	2^{22}
Forward	t (s)	3.29	53.2	214	–		
	$10^{-6}t/n^2$	3.1	3.2	3.2			
Backward	t (s)	0.31	4.92	20.5	86.1	346	–
	$10^{-7}t/n^2$	3.0	2.9	3.1	3.2	3.2	
Fast	t (s)	0.008	0.030	0.061	0.12	0.24	30.8
Backward	$10^{-6}t/n$	7.8	7.3	7.4	7.3	7.3	7.3

Minimisation of Railroad(n)

The theory behind minimisation algorithms

- Automata (DFA, NFA, WFA) are (mathematical) *structures*

The theory behind minimisation algorithms

- Automata (DFA, NFA, WFA) are (mathematical) *structures*
- Structures admit *morpisms* $\varphi\colon \mathcal{A} \to \mathcal{B}$, that is,
  maps that respect the structure

The theory behind minimisation algorithms

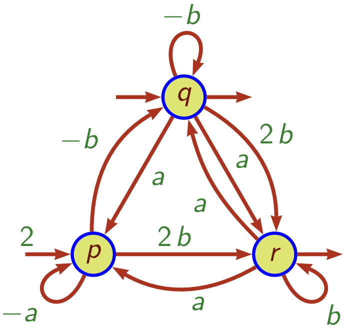
- Automata (DFA, NFA, WFA) are (mathematical) *structures*

- Structures admit *morpisms* $\varphi \colon \mathcal{A} \to \mathcal{B}$, that is,
  maps that respect the structure

- The *kernel* of $\varphi \colon \mathcal{A} \to \mathcal{B}$, that is, the equivalence map of φ is,
  a *partition* of the elements of the structure,
  here the states, that is called a *congruence*

A useful trick

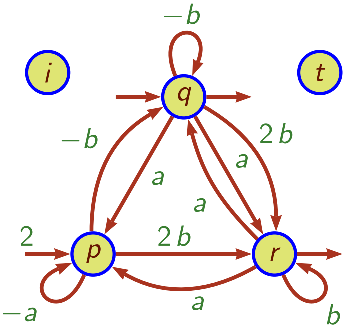
$$\mathcal{A} = \langle\, I, E, T \,\rangle$$



$$\begin{pmatrix} 2 & 1 & 0 \end{pmatrix}, \begin{pmatrix} -a & -b & 2\,b \\ a & -b & a+2\,b \\ a & a & b \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

A useful trick

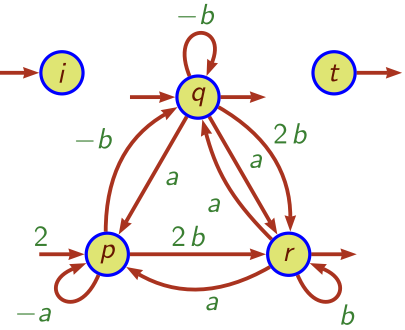
$$\mathcal{A} = \langle\, I, E, T \,\rangle$$



$$\begin{pmatrix} 2 & 1 & 0 \end{pmatrix},\ \begin{pmatrix} -a & -b & 2\,b \\ a & -b & a+2\,b \\ a & a & b \end{pmatrix},\ \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

A useful trick

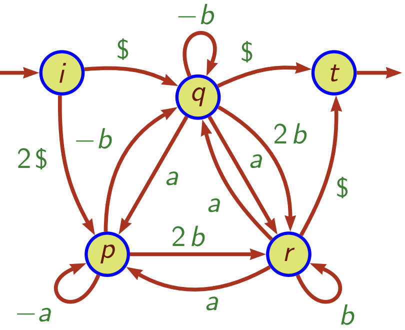
$$\mathcal{A} = \langle\, I, E, T\, \rangle$$



$$\begin{pmatrix} 2 & 1 & 0 \end{pmatrix}, \begin{pmatrix} -a & -b & 2\,b \\ a & -b & a+2\,b \\ a & a & b \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

A useful trick

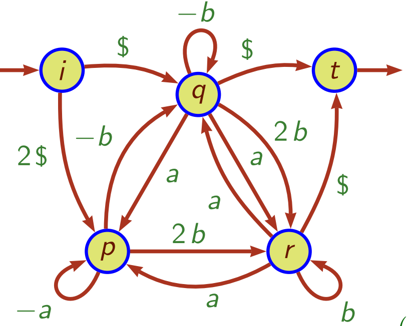
$$\mathcal{A} = \langle\, I, E, T \,\rangle$$



$$\begin{pmatrix} 2 & 1 & 0 \end{pmatrix}, \begin{pmatrix} -a & -b & 2\,b \\ a & -b & a+2\,b \\ a & a & b \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

A useful trick

$\mathcal{A} = \langle\, I, E, T \,\rangle \qquad A_{\$} = A \cup \{\$\} \qquad \mathcal{A}_{\$} = \langle\, i, E_{\$}, t \,\rangle$



$$\begin{pmatrix} 2 & 1 & 0 \end{pmatrix}, \begin{pmatrix} -a & -b & 2\,b \\ a & -b & a+2\,b \\ a & a & b \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 2\$ & \$ & 0 & 0 \\ 0 & -a & -b & 2\,b & 0 \\ 0 & a & -b & a+2\,b & \$ \\ 0 & a & a & b & \$ \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Definition

$\mathcal{A} = \langle\, Q, i, E, t \,\rangle$ $\mathbb{K}$-automaton

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}$, if:

$$\{i\} \in \mathcal{P}\,, \qquad \{t\} \in \mathcal{P}\,, \qquad \text{and}$$

$$\forall p, q \quad p\mathcal{P}q \quad \Longrightarrow \quad \forall a \in A_{\$}\,,\ \forall D \in \mathcal{P} \quad \sum_{r \in D} E(p,a,r) = \sum_{r \in D} E(q,a,r)$$

The theory behind minimisation algorithms

Definition

$\mathcal{A} = \langle\, Q, i, E, t \,\rangle$ $\mathbb{K}$-automaton

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}$, if:

$$\{i\} \in \mathcal{P}\,, \qquad \{t\} \in \mathcal{P}\,, \qquad \text{and}$$

$$\forall p, q \quad p\mathcal{P}q \quad \Longrightarrow \quad \forall a \in A_{\$}\,,\ \forall D \in \mathcal{P} \quad \sum_{r \in D} E(p, a, r) = \sum_{r \in D} E(q, a, r)$$

Theorem

Every $\mathbb{K}$-automaton $\mathcal{A}$ admits a unique coarsest congruence

The theory behind minimisation algorithms

Definition

$\mathcal{A} = \langle\, Q, i, E, t \,\rangle$ $\mathbb{K}$-automaton

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}$, if:

$$\{i\} \in \mathcal{P}\,, \qquad \{t\} \in \mathcal{P}\,, \qquad \text{and}$$

$$\forall p, q \quad p\mathcal{P}q \quad \Longrightarrow \quad \forall a \in A_{\$}\,,\ \forall D \in \mathcal{P} \quad \sum_{r \in D} E(p,a,r) = \sum_{r \in D} E(q,a,r)$$

Theorem

Every $\mathbb{K}$-automaton $\mathcal{A}$ admits a unique coarsest congruence

Definition

The quotient of $\mathcal{A}$ by its *coarsest* congruence is *the minimal quotient* of $\mathcal{A}$

The theory behind minimisation algorithms

Remark

The definition of a congruence (and of Out-morphism) is directed

The theory behind minimisation algorithms

Remark

The definition of a congruence (and of Out-morphism) is directed

The definition of *Out-morphism* coincides

- for DFA, with the classical notion of *morphism*
- for NFA, with the notion of *bisimulation*
- for WFA, with the *simulation* of Bloom-Ésik

The proto-algorithm

Definition

The *signature* of state p of $\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ with respect to $D \subseteq Q$ is the *map* $sig[p, D] \colon A_{\$} \to \mathbb{K}$ defined by:

$$sig[p, D](a) = \sum_{q \in D} E(p, a, q)$$

The proto-algorithm

Definition

The *signature* of state p of $\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ with respect to $D \subseteq Q$ is the *map* $sig[p, D] \colon A_{\$} \to \mathbb{K}$ defined by:

$$sig[p, D](a) = \sum_{q \in D} E(p, a, q)$$

Definition

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}_{\$}$, if:

$$\{i\} \in \mathcal{P}\,, \qquad \{t\} \in \mathcal{P}\,, \qquad \text{and}$$

$$\forall p, q \quad p \mathcal{P} q \quad \Longrightarrow \quad \forall a \in A_{\$}\,,\ \forall D \in \mathcal{P} \quad \sum_{r \in D} E(p, a, r) = \sum_{r \in D} E(q, a, r)$$

The proto-algorithm

Definition

The *signature* of state p of $\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ with respect to $D \subseteq Q$ is the *map* $sig[p,D]\colon A_{\$} \to \mathbb{K}$ defined by:

$$sig[p,D](a) = \sum_{q \in D} E(p,a,q)$$

Definition

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}_{\$}$, if:

$$\{i\} \in \mathcal{P}\,, \qquad \{t\} \in \mathcal{P}\,, \qquad \text{and}$$

$$\forall p,q \quad p\mathcal{P}q \quad \Longrightarrow \quad \forall a \in A_{\$}\,,\ \forall D \in \mathcal{P} \quad \sum_{r \in D} E(p,a,r) = \sum_{r \in D} E(q,a,r)$$

Definition

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}_{\$}$ if:

$$\forall C \in \mathcal{P}\,,\ \forall p,q \in C\,,\ \ \forall D \in \mathcal{P} \qquad sig[p,D] = sig[q,D]\ .$$

The proto-algorithm

Definition

The *signature* of state p of $\mathcal{A}_\$ = \langle\, Q, i, E, t\,\rangle$ with respect to $D \subseteq Q$ is the *map* $sig[p,D]\colon A_\$ \to \mathbb{K}$ defined by:

$$sig[p,D](a) = \sum_{q \in D} E(p,a,q)$$

Definition

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}_\$$ if:

$$\forall C \in \mathcal{P}\,,\ \forall p,q \in C\,,\ \ \forall D \in \mathcal{P} \qquad sig[p,D] = sig[q,D]\ .$$

The proto-algorithm

Definition

The *signature* of state p of $\mathcal{A}_\$ = \langle Q, i, E, t \rangle$ with respect to $D \subseteq Q$ is the *map* $sig[p, D] \colon A_\$ \to \mathbb{K}$ defined by:

$$sig[p, D](a) = \sum_{q \in D} E(p, a, q)$$

Definition

An equivalence $\mathcal{P}$ on Q is a *congruence* on $\mathcal{A}_\$$ if:

$$\forall C \in \mathcal{P}\,,\ \forall p, q \in C\,,\ \forall D \in \mathcal{P} \qquad sig[p, D] = sig[q, D]\ .$$

Definition

$split[C, D]$ map equivalence on C of the signature w.r.t. D

$$\forall p, q \in C \quad split[C, D](p) = split[C, D](q) \iff sig[p, D] = sig[q, D]$$

The proto-algorithm

Definition

The *signature* of state p of $\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ with respect to $D \subseteq Q$ is the *map* $\mathit{sig}[p, D] \colon A_\$ \to \mathbb{K}$ defined by:

$$\mathit{sig}[p, D](a) = \sum_{q \in D} E(p, a, q)$$

Definition

$\mathit{split}[C, D]$ map equivalence on C of the signature w.r.t. D

$$\forall p, q \in C \quad \mathit{split}[C, D](p) = \mathit{split}[C, D](q) \;\Leftrightarrow\; \mathit{sig}[p, D] = \mathit{sig}[q, D]$$

The proto-algorithm

Definition

The *signature* of state p of $\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ with respect to $D \subseteq Q$ is the *map* $sig[p, D] \colon A_\$ \to \mathbb{K}$ defined by:

$$sig[p, D](a) = \sum_{q \in D} E(p, a, q)$$

Definition

$split[C, D]$ map equivalence on C of the signature w.r.t. D

$$\forall p, q \in C \quad split[C, D](p) = split[C, D](q) \;\Leftrightarrow\; sig[p, D] = sig[q, D]$$

The proto-algorithm

$\mathcal{P} := \mathcal{P}_0$

while there exists a splitting pair (C, D) in $\mathcal{P}$

$\mathcal{P} := \mathcal{P} \wedge split[C, D]$

The Forward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ n states m transitions

The Forward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ n states m transitions

The Forward Algorithm

The Forward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Forward Algorithm $\quad$ *queue* $=$ queue of classes

The Forward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Forward Algorithm $\quad$ *queue* = queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\quad$ $Q \to$ *queue*

The Forward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Forward Algorithm

queue $=$ queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\quad$ $Q \rightarrow \mathit{queue}$
- Notion of *round* in the algorithm

The Forward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ n states m transitions

The Forward Algorithm

$queue$ = queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $Q \rightarrow queue$
- Notion of *round* in the algorithm
- At round $i+1$, for every $C \in queue$,
  - compute $split[C, D]$ for every $D \in \mathcal{P}_i$
  - put the pieces in $queue$, even if C is not split (but the singletons)

The Forward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t\,\rangle$ n states m transitions

The Forward Algorithm queue = queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $Q \to \mathit{queue}$
- Notion of *round* in the algorithm
- At round $i+1$, for every $C \in \mathit{queue}$,
  - compute $\mathit{split}[C, D]$ for every $D \in \mathcal{P}_i$
  - put the pieces in queue, even if C is not split (but the singletons)
- $\mathcal{P}_{i+1}$ = content of queue + singletons

The Forward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ n states m transitions

The Forward Algorithm

queue = queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $Q \to$ *queue*
- Notion of *round* in the algorithm
- At round $i+1$, for every $C \in$ *queue*,
  - compute *split*$[C, D]$ for every $D \in \mathcal{P}_i$
  - put the pieces in *queue*, even if C is not split (but the singletons)
- $\mathcal{P}_{i+1}$ = content of *queue* + singletons
- If no split occurs in round $i+1$, ie if $\mathcal{P}_{i+1} = \mathcal{P}_i$, stop

The Forward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Forward Algorithm $\quad$ *queue* $=$ queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\quad$ $Q \rightarrow \mathit{queue}$
- Notion of *round* in the algorithm
- At round $i+1$, for every $C \in \mathit{queue}$,
  - compute $\mathit{split}[C, D]$ for every $D \in \mathcal{P}_i$
  - put the pieces in *queue*, even if C is not split (but the singletons)
- $\mathcal{P}_{i+1} =$ content of *queue* $+$ singletons
- If no split occurs in round $i+1$, ie if $\mathcal{P}_{i+1} = \mathcal{P}_i$, stop

Theorem

Forward Algorithm computes the coarsest congruence in $\mathrm{O}(n(m+n))$

The Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Backward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ $\quad n$ states $\quad m$ transitions

The Backward Algorithm

The Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Backward Algorithm $\quad$ *queue* $=$ queue of classes

The Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ n states m transitions

The Backward Algorithm queue = queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\{t\} \rightarrow \mathit{queue}$ $Q \rightarrow \mathit{queue}$

The Backward Algorithm

$\mathcal{A}_{\$} = \langle Q, i, E, t \rangle$ n states m transitions

The Backward Algorithm

queue $=$ queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\{t\} \to queue$ $Q \to queue$
- for every $D \in queue$,
  - for every $C \in \mathcal{P}_i$ that is not a singleton,
    and that contains a *predecessor* of a state in $D \in queue$
    - compute $split[C, D]$
    - if it is a true split, put pieces in *queue* (even singletons)
  - $\mathcal{P}_{i+1} = \mathcal{P}_i \wedge \cup\, split[C, D]$

The Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ n states m transitions

The Backward Algorithm

queue = queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\{t\} \to queue$ $Q \to queue$
- for every $D \in queue$,
  - for every $C \in \mathcal{P}_i$ that is not a singleton, and that contains a *predecessor* of a state in $D \in queue$
    - compute $split[C, D]$
    - if it is a true split, put pieces in *queue* (even singletons)
  - $\mathcal{P}_{i+1} = \mathcal{P}_i \wedge \cup\, split[C, D]$
- stop when *queue* is empty

The Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ n states m transitions

The Backward Algorithm $queue$ $=$ queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\{t\} \to queue$ $Q \to queue$
- for every $D \in queue$,
  - for every $C \in \mathcal{P}_i$ that is not a singleton, and that contains a *predecessor* of a state in $D \in queue$
    - compute $split[C, D]$
    - if it is a true split, put pieces in *queue* (even singletons)
  - $\mathcal{P}_{i+1} = \mathcal{P}_i \wedge \cup\, split[C, D]$
- stop when $queue$ is empty

Theorem
Backward Algorithm computes the coarsest congruence in $\mathrm{O}(n(m+n))$

The Fast Backward Algorithm

Hopcroft's algorithm is an improvement of Backward Algorithm for complete DFA

It implements indeed the strategy 'all but the largest' described by Tarjan and Paige

The Fast Backward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Fast Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ $\qquad$ n states $\qquad$ m transitions

Signatures are equipped with a pointwise addition

$$D \cap D' = \emptyset \quad \Longrightarrow \quad sig[p, D \cup D'] = sig[p, D] + sig[p, D']$$

The Fast Backward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ $\qquad$ n states $\qquad$ m transitions

Signatures are equipped with a pointwise addition

$$D \cap D' = \emptyset \quad \Longrightarrow \quad sig[p, D \cup D'] = sig[p, D] + sig[p, D']$$

Definition

$\mathcal{A}$ has *simplifiable signatures* if $\forall D \subseteq Q \quad \forall C \subseteq D \quad \forall p, q \in Q$

$sig[p, D] = sig[q, D]$ and $sig[p, C] = sig[q, C] \Longrightarrow sig[p, D \backslash C] = sig[q, D \backslash C]$.

The Fast Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ $\quad n$ states $\quad m$ transitions

Signatures are equipped with a pointwise addition

$$D \cap D' = \emptyset \quad \Longrightarrow \quad sig[p, D \cup D'] = sig[p, D] + sig[p, D']$$

Definition

$\mathcal{A}$ has *simplifiable signatures* if $\forall D \subseteq Q \quad \forall C \subseteq D \quad \forall p, q \in Q$

$sig[p, D] = sig[q, D]$ and $sig[p, C] = sig[q, C] \Longrightarrow sig[p, D \setminus C] = sig[q, D \setminus C]$.

If $(\mathbb{K}, +)$ is a *cancellative monoid* (in particular if $\mathbb{K}$ is a ring), then all $\mathbb{K}$-automata have simplifiable signatures.

The Fast Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ $\qquad$ n states $\qquad$ m transitions

Signatures are equipped with a pointwise addition

$$D \cap D' = \emptyset \quad \Longrightarrow \quad \mathit{sig}[p, D \cup D'] = \mathit{sig}[p, D] + \mathit{sig}[p, D']$$

Definition
$\mathcal{A}$ has *simplifiable signatures* if $\forall D \subseteq Q \quad \forall C \subseteq D \quad \forall p, q \in Q$
$\mathit{sig}[p, D] = \mathit{sig}[q, D]$ and $\mathit{sig}[p, C] = \mathit{sig}[q, C] \Longrightarrow \mathit{sig}[p, D \setminus C] = \mathit{sig}[q, D \setminus C]$.

If $(\mathbb{K}, +)$ is a *cancellative monoid* (in particular if $\mathbb{K}$ is a ring),
then all $\mathbb{K}$-automata have simplifiable signatures.

If $\mathcal{A}$ is a *deterministic* automaton — not necessarily complete,
then $\mathcal{A}$ has simplifiable signatures.

The Fast Backward Algorithm

$\mathcal{A}_{\$} = \langle\, Q, i, E, t \,\rangle$ n states m transitions

Signatures are equipped with a pointwise addition

$$D \cap D' = \emptyset \quad \Longrightarrow \quad sig[p, D \cup D'] = sig[p, D] + sig[p, D']$$

Definition
$\mathcal{A}$ has *simplifiable signatures* if $\forall D \subseteq Q \quad \forall C \subseteq D \quad \forall p, q \in Q$
$sig[p, D] = sig[q, D]$ and $sig[p, C] = sig[q, C] \Longrightarrow sig[p, D \setminus C] = sig[q, D \setminus C]$.

If $(\mathbb{K}, +)$ is a *cancellative monoid* (in particular if $\mathbb{K}$ is a ring),
then all $\mathbb{K}$-automata have simplifiable signatures.

If $\mathcal{A}$ is a *deterministic* automaton — not necessarily complete,
then $\mathcal{A}$ has simplifiable signatures.

If $\mathcal{A}$ is a *sequential* $\mathbb{K}$-automaton,
then $\mathcal{A}$ has simplifiable signatures.

The Fast Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ $\quad$ n states $\quad$ m transitions

The Fast Backward Algorithm $\quad$ *queue* $=$ queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\quad$ $\{t\} \rightarrow \mathit{queue}$ $\quad$ $Q \rightarrow \mathit{queue}$
- for every $D \in \mathit{queue}$,
  - for every $C \in \mathcal{P}_i$ that is not a singleton,
    and that contains a *predecessor* of a state in $D \in \mathit{queue}$
    - compute $\mathit{split}[C, D]$
    - if it is a true split, put pieces in *queue* (even singletons)
      but the largest piece
  - $\mathcal{P}_{i+1} = \mathcal{P}_i \wedge \cup\, \mathit{split}[C, D]$
- stop when *queue* is empty

The Fast Backward Algorithm

$\mathcal{A}_\$ = \langle\, Q, i, E, t \,\rangle$ $\qquad$ n states $\qquad$ m transitions

The Fast Backward Algorithm $\quad$ *queue* $=$ queue of classes

- $\mathcal{P}_0 = \{i\}, Q, \{t\}$ $\qquad$ $\{t\} \to$ *queue* $\qquad$ $Q \to$ *queue*
- for every $D \in$ *queue*,
  - for every $C \in \mathcal{P}_i$ that is not a singleton, and that contains a *predecessor* of a state in $D \in$ *queue*
    - compute *split*$[C, D]$
    - if it is a true split, put pieces in *queue* (even singletons) but the largest piece
  - $\mathcal{P}_{i+1} = \mathcal{P}_i \wedge \cup\, \mathit{split}[C, D]$
- stop when *queue* is empty

Theorem

If $\mathcal{A}$ has simplifiable signatures, then Fast Backward Algorithm computes the coarsest congruence in $\mathrm{O}((m+n)\log n)$

Conclusion

- Every automaton (DFA, NFA, WFA) has a minimal quotient

Conclusion

- Every automaton (DFA, NFA, WFA) has a minimal quotient
- Two main algorithms for computing the minimal quotient

Conclusion

- Every automaton (DFA, NFA, WFA) has a minimal quotient
- Two main algorithms for computing the minimal quotient
- They both have a time complexity of $\mathrm{O}(n(m+n))$

Conclusion

- Every automaton (DFA, NFA, WFA) has a minimal quotient
- Two main algorithms for computing the minimal quotient
- They both have a time complexity of $\mathrm{O}(n(m+n))$
- Devil is in the details of the implementation
  to achieve the prescribed complexity

Conclusion

- Every automaton (DFA, NFA, WFA) has a minimal quotient
- Two main algorithms for computing the minimal quotient
- They both have a time complexity of $\mathrm{O}(n(m+n))$
- Devil is in the details of the implementation
  to achieve the prescribed complexity
- Simplifiable signatures allow a complexity of $\mathrm{O}((m+n)\log n)$

Conclusion

- Every automaton (DFA, NFA, WFA) has a minimal quotient
- Two main algorithms for computing the minimal quotient
- They both have a time complexity of $\mathrm{O}(n(m+n))$
- Devil is in the details of the implementation
  to achieve the prescribed complexity
- Simplifiable signatures allow a complexity of $\mathrm{O}((m+n)\log n)$
- This subsumes and generalises works of
  Hopcroft, Béal–Crochemore, Valmari–Lehtinen

Conclusion

- Every automaton (DFA, NFA, WFA) has a minimal quotient
- Two main algorithms for computing the minimal quotient
- They both have a time complexity of $\mathrm{O}(n(m+n))$
- Devil is in the details of the implementation
  to achieve the prescribed complexity
- Simplifiable signatures allow a complexity of $\mathrm{O}((m+n)\log n)$
- This subsumes and generalises works of
  Hopcroft, Béal–Crochemore, Valmari–Lehtinen
- Open problem:
  lower bound for minimisation of Boolean and $\mathbb{Z}$min-automata