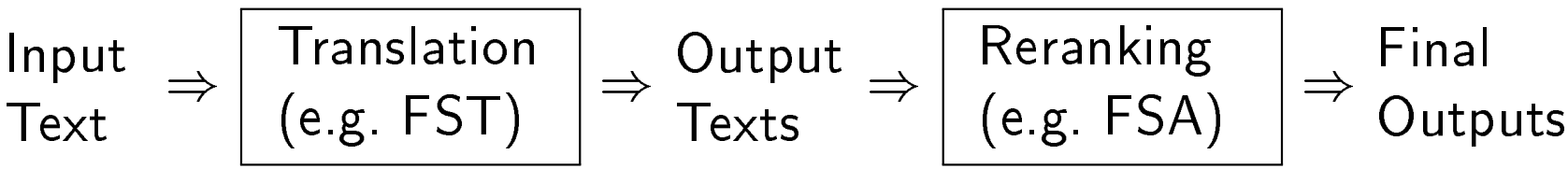
Minimization of Deterministic Weighted Tree Automata

Andreas Maletti

March 14, 2008

Syntax-based MT (e.g. TIBURON)

Overview

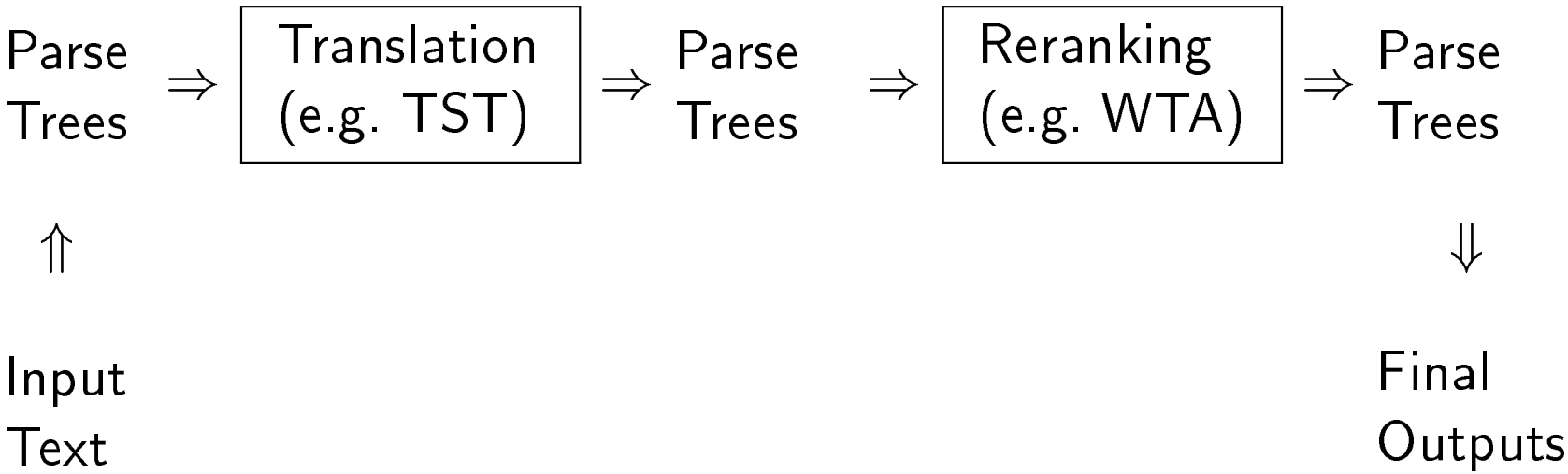


Abbreviations

- FST = Finite State Transducer
- FSA = Finite State Automaton

Syntax-based MT (e.g. TIBURON)

Overview



Abbreviations

- TST = Tree Series Transducer
- WTA = Weighted Tree Automaton

Table of Contents

Table of Contents

Overview

Tree Series

- Assigns weight (e.g. a probability) to each tree
- Weight drawn from semiring; e.g. $(\mathbb{R}, +, \cdot, 0, 1)$

Overview

Tree Series

- Assigns weight (e.g. a probability) to each tree
- Weight drawn from semiring; e.g. $(\mathbb{R}, +, \cdot, 0, 1)$

Weighted Tree Automaton

- Finitely represents tree series
- Defines the recognizable tree series

Overview

Tree Series

- Assigns weight (e.g. a probability) to each tree
- Weight drawn from semiring; e.g. $(\mathbb{R}, +, \cdot, 0, 1)$

Weighted Tree Automaton

- Finitely represents tree series
- Defines the recognizable tree series

Application

- Re-ranker for parse trees
- Representation of parses

Syntax

Definition

Weighted tree automaton (wta) is tuple $(Q, \Sigma, \mathcal{A}, F, \mu)$ where

- Q: finite set of *states*
- Σ: ranked alphabet of *input symbols*
- $\mathcal{A} = (A, +, \cdot, 0, 1)$: semiring of *weights*
- $F \subseteq Q$: *final states*

Syntax

Definition

Weighted tree automaton (wta) is tuple $(Q, \Sigma, \mathcal{A}, F, \mu)$ where

- Q: finite set of *states*
- Σ: ranked alphabet of *input symbols*
- $\mathcal{A} = (A, +, \cdot, 0, 1)$: semiring of *weights*
- $F \subseteq Q$: *final states*
- $\mu = (\mu_k)_{k \in \mathbb{N}}$ with $\mu_k \colon \Sigma_k \to A^{Q \times Q^k}$

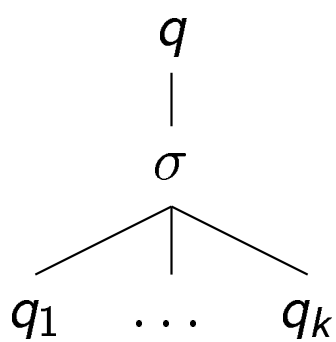
Syntax

Definition

Weighted tree automaton (wta) is tuple $(Q, \Sigma, \mathcal{A}, F, \mu)$ where

- Q: finite set of *states*
- Σ: ranked alphabet of *input symbols*
- $\mathcal{A} = (A, +, \cdot, 0, 1)$: semiring of *weights*
- $F \subseteq Q$: *final states*
- $\mu = (\mu_k)_{k \in \mathbb{N}}$ with $\mu_k \colon \Sigma_k \to A^{Q \times Q^k}$

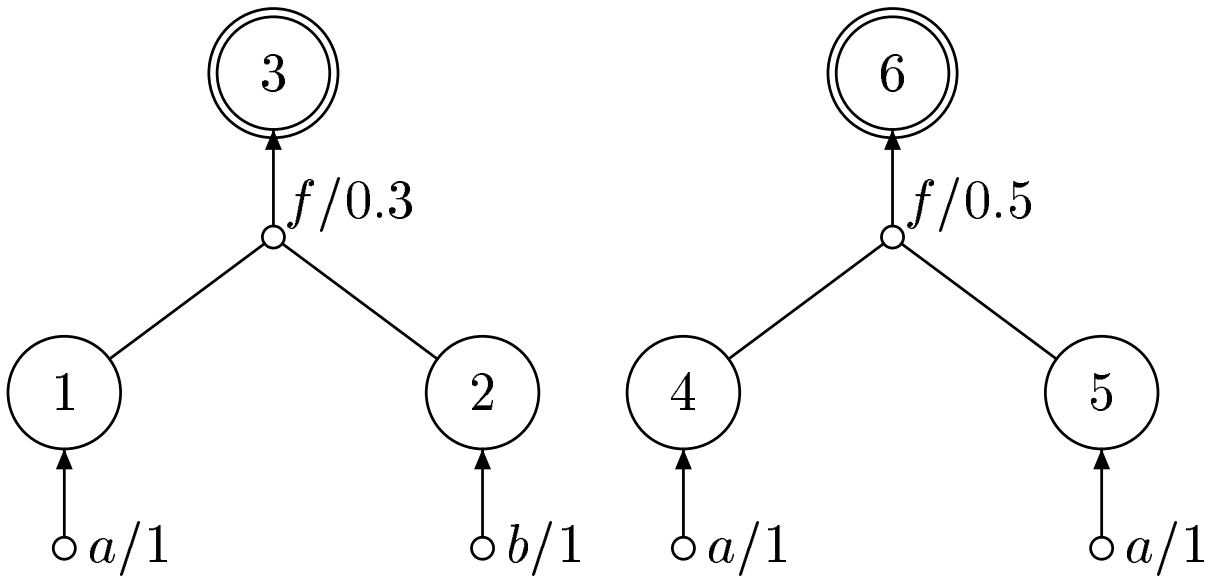
Sample Transition



with weight $\mu_k(\sigma)_{q,q_1 \cdots q_k}$

Syntax — Illustration

Sample Automaton



Semantics

Definition

Let $t \in T_\Sigma(Q)$ and $W = \mathrm{pos}(t)$.

- Run on t: map $r\colon W \to Q$ with $r(w) = t(w)$ if $t(w) \in Q$

Semantics

Definition

Let $t \in T_\Sigma(Q)$ and $W = \mathrm{pos}(t)$.

- Run on t: map $r\colon W \to Q$ with $r(w) = t(w)$ if $t(w) \in Q$
- Weight of r

$$\mathrm{wt}(r) = \prod_{\substack{w \in W \\ t(w) \in \Sigma}} \mu_k(t(w))_{r(w), r(w1) \cdots r(wk)}$$

Semantics

Definition

Let $t \in T_\Sigma(Q)$ and $W = \mathrm{pos}(t)$.

- Run on t: map $r\colon W \to Q$ with $r(w) = t(w)$ if $t(w) \in Q$
- Weight of r

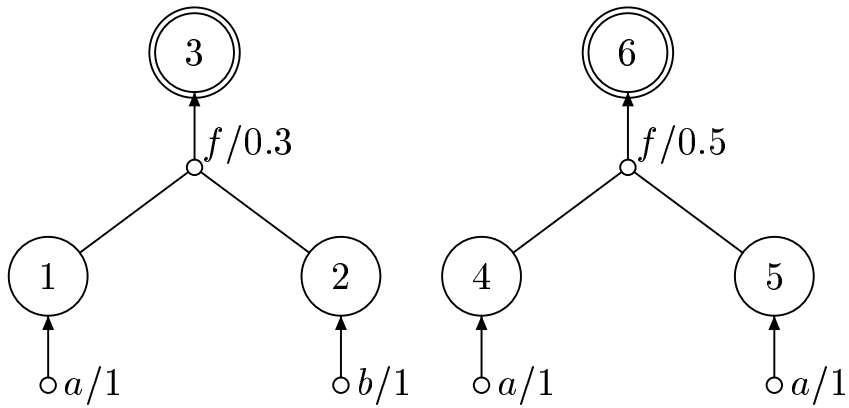
$$\mathrm{wt}(r) = \prod_{\substack{w \in W \\ t(w) \in \Sigma}} \mu_k(t(w))_{r(w), r(w1) \cdots r(wk)}$$

- Recognized tree series

$$(\|M\|, t) = \sum_{\substack{r \text{ run on } t \\ r(\varepsilon) \in F}} \mathrm{wt}(r)$$

Semantics — Illustration

Sample Automaton



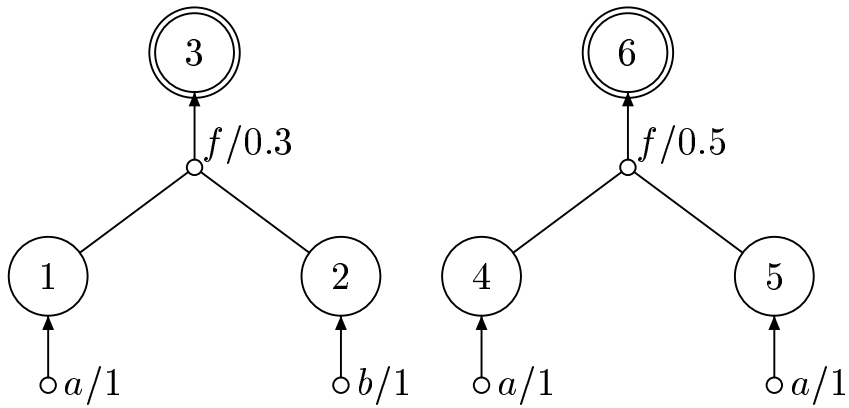
Sample Runs

Input tree: f with children a and b

Runs: 6 with weight 0 — with children 1 and 2

Semantics — Illustration

Sample Automaton



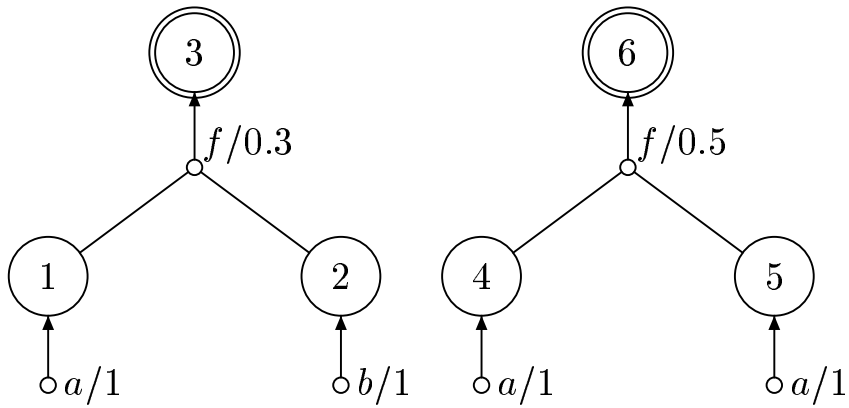
Sample Runs

Input tree: $f(a, b)$

Runs: $f(a, b)$ with weight

Semantics — Illustration

Sample Automaton

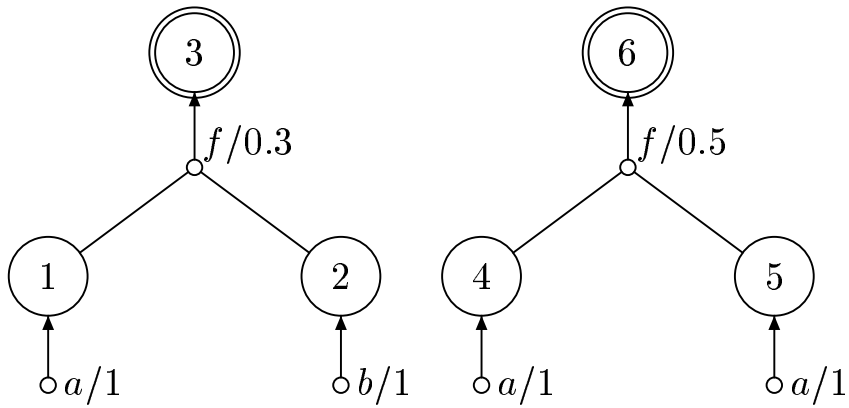


Sample Runs

Input tree: $f(a, b)$ Runs: $f(1, b)$ with weight 1

Semantics — Illustration

Sample Automaton



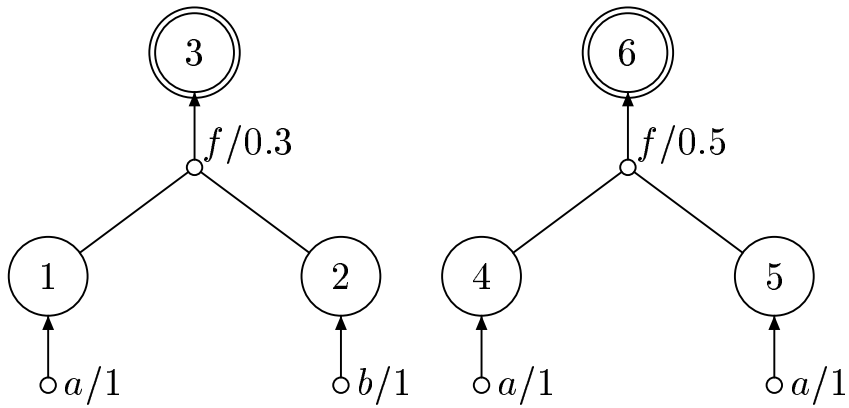
Sample Runs

Input tree: f with children a and b

Runs: f with children 1 and 2 with weight 1

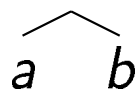
Semantics — Illustration

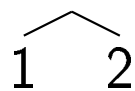
Sample Automaton



Sample Runs

Input tree: f Runs: 3 with weight 0.3





Determinism

Definition

Deterministic wta: for every $\sigma \in \Sigma_k$ and $w \in Q^k$ there exists exactly one $q \in Q$ such that $\mu_k(\sigma)_{q,w} \neq 0$

Determinism

Definition

Deterministic wta: for every $\sigma \in \Sigma_k$ and $w \in Q^k$ there exists exactly one $q \in Q$ such that $\mu_k(\sigma)_{q,w} \neq 0$

Notes

- Deterministic wta does not use addition

Determinism

Definition

Deterministic wta: for every $\sigma \in \Sigma_k$ and $w \in Q^k$ there exists exactly one $q \in Q$ such that $\mu_k(\sigma)_{q,w} \neq 0$

Notes

- Deterministic wta does not use addition
- Recognizable $\neq$ deterministically recognizable

Determinism

Definition

Deterministic wta: for every $\sigma \in \Sigma_k$ and $w \in Q^k$ there exists exactly one $q \in Q$ such that $\mu_k(\sigma)_{q,w} \neq 0$

Notes

- Deterministic wta does not use addition
- Recognizable $\neq$ deterministically recognizable
- Determinization possible in locally-finite semirings [BV03]

Determinism

Definition

Deterministic wta: for every $\sigma \in \Sigma_k$ and $w \in Q^k$ there exists exactly one $q \in Q$ such that $\mu_k(\sigma)_{q,w} \neq 0$

Notes

- Deterministic wta does not use addition
- Recognizable $\neq$ deterministically recognizable
- Determinization possible in locally-finite semirings [BV03]
- Partial determinization for probabilities [MK06]

Table of Contents

Overview

Applicability

- Deterministic wta
- Commutative semifield (i.e. multiplicative inverses)

Roadmap

- Myhill-Nerode congruence relation [Bor03]
- Determine signs of life
- Refinement

Myhill-Nerode congruence

Definition

$p \equiv q$: there exists nonzero a such that for every context C

$$(\|M\|, C[p]) = a \cdot (\|M\|, C[q])$$

Myhill-Nerode congruence

Definition

$p \equiv q$: there exists nonzero a such that for every context C

$$(\|M\|, C[p]) = a \cdot (\|M\|, C[q])$$

Notes

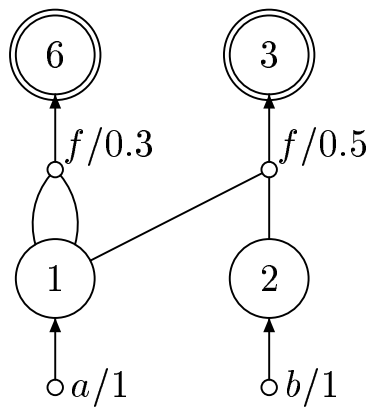
- Semifields are zero-divisor free
- Element a is unique if p is not dead

Signs of Life

Definition

Sign of life of $q \in Q$: context C such that $(\|M\|, C[q]) \neq 0$

Example



State	Sign of life	State	Sign of life
1	$f(\square, b)$	2	$f(1, \square)$
3	$\square$	6	$\square$

Compute Signs of Life

Algorithm

$D \leftarrow Q \setminus F$
2: $\mathsf{sol} \leftarrow \{(q, \square) \mid q \in F\}$
$T \leftarrow \{(w, \sigma, q) \in \delta \mid q \in F\}$
4: **while** $T \neq \emptyset$ **do**
$\tau = ((q_1, \ldots, q_k), \sigma, q) \in T$
6: $I \leftarrow \{i \in [1, k] \mid q_i \in D, \forall j \in [1, i-1] \colon q_j \neq q_i\}$
$\mathsf{sol} \leftarrow \mathsf{sol} \cup \{(q_i, \mathsf{sol}(q)[\sigma(q_1, \ldots, \square, \ldots, q_k)]) \mid i \in I\}$
8: $P \leftarrow \{q_i \mid i \in I\}$
$D \leftarrow D \setminus P$
10: $T \leftarrow (T \setminus \{\tau\}) \cup \{(w, \gamma, p) \in \delta \mid p \in P\}$
end while
12: $\Pi \leftarrow \{\{q \in Q \setminus D \mid \mathsf{ht}(\mathsf{sol}(q)) = i\} \mid i \geq 1\} \cup \{D\}$
return (Π, sol, D)

Compute Signs of Life

Algorithm

$D \leftarrow Q \setminus F$
2: $\mathsf{sol} \leftarrow \{(q, \square) \mid q \in F\}$
$T \leftarrow \{(w, \sigma, q) \in \delta \mid q \in F\}$
4: **while** $T \neq \emptyset$ **do**
$\tau = ((q_1, \ldots, q_k), \sigma, q) \in T$
6: $I \leftarrow \{i \in [1, k] \mid q_i \in D, \forall j \in [1, i-1] \colon q_j \neq q_i\}$
$\mathsf{sol} \leftarrow \mathsf{sol} \cup \{(q_i, \mathsf{sol}(q)[\sigma(q_1, \ldots, \square, \ldots, q_k)]) \mid i \in I\}$
8: $P \leftarrow \{q_i \mid i \in I\}$
$D \leftarrow D \setminus P$
10: $T \leftarrow (T \setminus \{\tau\}) \cup \{(w, \gamma, p) \in \delta \mid p \in P\}$
end while
12: $\Pi \leftarrow \{\{q \in Q \setminus D \mid \mathsf{ht}(\mathsf{sol}(q)) = i\} \mid i \geq 1\} \cup \{D\}$
return (Π, sol, D)

Compute Signs of Life

Algorithm

$D \leftarrow Q \setminus F$
2: $\mathsf{sol} \leftarrow \{(q, \square) \mid q \in F\}$
$T \leftarrow \{(w, \sigma, q) \in \delta \mid q \in F\}$
4: **while** $T \neq \emptyset$ **do**
$\tau = ((q_1, \ldots, q_k), \sigma, q) \in T$
6: $I \leftarrow \{i \in [1, k] \mid q_i \in D, \forall j \in [1, i-1] \colon q_j \neq q_i\}$
$\mathsf{sol} \leftarrow \mathsf{sol} \cup \{(q_i, \mathsf{sol}(q)[\sigma(q_1, \ldots, \square, \ldots, q_k)]) \mid i \in I\}$
8: $P \leftarrow \{q_i \mid i \in I\}$
$D \leftarrow D \setminus P$
10: $T \leftarrow (T \setminus \{\tau\}) \cup \{(w, \gamma, p) \in \delta \mid p \in P\}$
end while
12: $\Pi \leftarrow \{\{q \in Q \setminus D \mid \mathsf{ht}(\mathsf{sol}(q)) = i\} \mid i \geq 1\} \cup \{D\}$
return (Π, sol, D)

Compute Signs of Life

Algorithm

$D \leftarrow Q \setminus F$
2: $\mathsf{sol} \leftarrow \{(q, \square) \mid q \in F\}$
$T \leftarrow \{(w, \sigma, q) \in \delta \mid q \in F\}$
4: **while** $T \neq \emptyset$ **do**
$\tau = ((q_1, \ldots, q_k), \sigma, q) \in T$
6: $I \leftarrow \{i \in [1, k] \mid q_i \in D, \forall j \in [1, i-1] \colon q_j \neq q_i\}$
$\mathsf{sol} \leftarrow \mathsf{sol} \cup \{(q_i, \mathsf{sol}(q)[\sigma(q_1, \ldots, \square, \ldots, q_k)]) \mid i \in I\}$
8: $P \leftarrow \{q_i \mid i \in I\}$
$D \leftarrow D \setminus P$
10: $T \leftarrow (T \setminus \{\tau\}) \cup \{(w, \gamma, p) \in \delta \mid p \in P\}$
end while
12: $\Pi \leftarrow \{\{q \in Q \setminus D \mid \mathsf{ht}(\mathsf{sol}(q)) = i\} \mid i \geq 1\} \cup \{D\}$
return (Π, sol, D)

Computing Signs of Life

Definition

Dead state $q \in Q$: there exists no sign of life of q

Notation

- r: maximal rank of input symbols
- m: number of transitions
- n: number of states

Computing Signs of Life

Definition

Dead state $q \in Q$: there exists no sign of life of q

Notation

- r: maximal rank of input symbols
- m: number of transitions
- n: number of states

Notes

- Algorithm runs in $O(rm)$
- Computes signs of life of size $\leq rn$
- Computes dead states

Stages

Definition

Stage $(\Pi, \mathrm{sol}, f, r)$:

(i) $\equiv$ refinement of $\equiv_\Pi$

(ii) $\mathrm{sol}(F) = \{\square\}$

(iii) for live q with $p = r([q])$

$$(\|M\|, \mathrm{sol}(p)[q]) = f(q) \cdot (\|M\|, \mathrm{sol}(p)[p])$$

Stages

Definition

Stable stage $(\Pi, \mathrm{sol}, f, r)$:

(i) $\equiv$ refinement of $\equiv_\Pi$

(ii) $\mathrm{sol}(F) = \{\Box\}$

(iii) for live q with $p = r([q])$

$$(\|M\|, \mathrm{sol}(p)[q]) = f(q) \cdot (\|M\|, \mathrm{sol}(p)[p])$$

(iv) $\equiv_\Pi$ congruence

(v) for symbol σ and context C with live $\delta_\sigma(C[q])$

$$f(q)^{-1} \cdot c_\sigma(C[q]) \cdot f(\delta_\sigma(C[q])) = c_\sigma(C[p]) \cdot f(\delta_\sigma(C[p]))$$

where $\delta_\sigma \colon Q^k \to Q$ and $c_\sigma \colon Q^k \to A$

Completing a Stage

Algorithm

$f \leftarrow \emptyset$
$r \leftarrow \emptyset$
$\Pi \leftarrow \Pi \setminus \{D\}$
$\Pi' \leftarrow \{D\}$
while $\Pi \neq \emptyset$ and $\Pi \neq \{\emptyset\}$ **do**
  let $P \in \Pi$ and $p \in P$
  $P' \leftarrow \{q \in P \mid \delta(\mathrm{sol}(p)[q]) \in F\}$
  $f \leftarrow f \cup \{(q, c(\mathrm{sol}(p)[q]) \cdot c(\mathrm{sol}(p)[p])^{-1}) \mid q \in P'\}$
  $r \leftarrow r \cup \{(P', p)\}$
  $\Pi' \leftarrow \Pi' \cup \{P'\}$
  $\Pi \leftarrow (\Pi \setminus \{P\}) \cup \{P \setminus P'\}$
end while
return $(\Pi', \mathrm{sol}, f, r)$

Completing a Stage

Algorithm

$f \leftarrow \emptyset$
$r \leftarrow \emptyset$
$\Pi \leftarrow \Pi \setminus \{D\}$
$\Pi' \leftarrow \{D\}$
while $\Pi \neq \emptyset$ and $\Pi \neq \{\emptyset\}$ **do**
  let $P \in \Pi$ and $p \in P$
  $P' \leftarrow \{q \in P \mid \delta(\mathrm{sol}(p)[q]) \in F\}$
  $f \leftarrow f \cup \{(q, c(\mathrm{sol}(p)[q]) \cdot c(\mathrm{sol}(p)[p])^{-1}) \mid q \in P'\}$
  $r \leftarrow r \cup \{(P', p)\}$
  $\Pi' \leftarrow \Pi' \cup \{P'\}$
  $\Pi \leftarrow (\Pi \setminus \{P\}) \cup \{P \setminus P'\}$
end while
return $(\Pi', \mathrm{sol}, f, r)$

Completing a Stage

Algorithm

$f \leftarrow \emptyset$

$r \leftarrow \emptyset$

$\Pi \leftarrow \Pi \setminus \{D\}$

$\Pi' \leftarrow \{D\}$

while $\Pi \neq \emptyset$ and $\Pi \neq \{\emptyset\}$ **do**

  let $P \in \Pi$ and $p \in P$

  $P' \leftarrow \{q \in P \mid \delta(\mathrm{sol}(p)[q]) \in F\}$

  $f \leftarrow f \cup \{(q, c(\mathrm{sol}(p)[q]) \cdot c(\mathrm{sol}(p)[p])^{-1}) \mid q \in P'\}$

  $r \leftarrow r \cup \{(P', p)\}$

  $\Pi' \leftarrow \Pi' \cup \{P'\}$

  $\Pi \leftarrow (\Pi \setminus \{P\}) \cup \{P \setminus P'\}$

end while

return $(\Pi', \mathrm{sol}, f, r)$

Completing and Refining a Stage

Notes

- Algorithm runs in $O(rn^3)$
- Returns stage such that $\equiv_{\Pi'}$ is a refinement of $\equiv_{\Pi}$

Completing and Refining a Stage

Notes

- Algorithm runs in $O(rn^3)$
- Returns stage such that $\equiv_{\Pi'}$ is a refinement of $\equiv_{\Pi}$

Definition

Refinement of $(\Pi, \mathrm{sol}, f, r)$: Partition Π' with $p \equiv_{\Pi'} q$ if

(i) $p \equiv_{\Pi} q$

(ii) $\delta_\sigma(C[p]) \equiv_{\Pi} \delta_\sigma(C[q])$

(iii) if $\delta_\sigma(C[p])$ is live, then

$$f(p)^{-1} \cdot c_\sigma(C[p]) \cdot f(\delta_\sigma(C[p])) = f(q)^{-1} \cdot c_\sigma(C[q]) \cdot f(\delta_\sigma(C[q]))$$

for states p and q, symbol σ, and context C

Complete Algorithm

Algorithm

$(\Pi', \mathsf{sol}, D) \leftarrow \textsc{ComputeSoL}(M)$
repeat
  $(\Pi, \mathsf{sol}, f, r) \leftarrow \textsc{Complete}(M, \Pi', \mathsf{sol}, D)$
  $\Pi' \leftarrow \textsc{Refine}(M, \Pi, \mathsf{sol}, f, r, D)$
until $\Pi' = \Pi$
return minimized wta

Complete Algorithm

Algorithm

$(\Pi', \mathsf{sol}, D) \leftarrow \textsc{ComputeSol}(M)$
repeat
  $(\Pi, \mathsf{sol}, f, r) \leftarrow \textsc{Complete}(M, \Pi', \mathsf{sol}, D)$
  $\Pi' \leftarrow \textsc{Refine}(M, \Pi, \mathsf{sol}, f, r, D)$
until $\Pi' = \Pi$
return minimized wta

Notes

- Algorithm runs in $O(rmn^4)$
- Returns equivalent minimal deterministic wta

Table of Contents

Experiments

State Count

Original	Minimal	Reduction to
98	68	69%
394	308	78%
497	381	77%
727	515	71%
2701	1993	74%
3686	1766	48%

Experiments

State Count

Original	Minimal	Reduction to
98	68	69%
394	308	78%
497	381	77%
727	515	71%
2701	1993	74%
3686	1766	48%

State & Transition Count

Error	Original	Minimal	Reduction to
10^{-4}	$(727, 6485)$	$(629, 6131)$	$(87\%, 95\%)$
10^{-2}	$(727, 6485)$	$(525, 3425)$	$(72\%, 53\%)$

References

Björn Borchardt.
The Myhill-Nerode theorem for recognizable tree series.
In *Proc. 7th Int. Conf. Developments in Language Theory*, volume 2710 of LNCS, pages 146–158. Springer, 2003.

Björn Borchardt and Heiko Vogler.
Determinization of finite state weighted tree automata.
J. Autom. Lang. Combin., 8(3):417–463, 2003.

Jonathan May and Kevin Knight.
A better *n*-best list: Practical determinization of weighted finite tree automata.
In *Proc. HLT-NAACL*. The Association for Computational Linguistics, 2006.

The End

Thank You!