Comparing nondeterministic and quasideterministic finite-state transducers built from morphological dictionaries*

Alicia Garrido-Alenda and Mikel L. Forcada
Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant
E-03071 Alacant, Spain

SEPLN 2002, Valladolid

*Funded by Caja de Ahorros del Mediterráneo, Universitat d'Alacant and CICyT (project TIC2000-1599-C02-02).

Index

Index

Lexical transformations in NLP systems

Aligned and unaligned dictionaries

Transducers: quasideterministic and nondeterministic

Building transducers from dictionaries

Comparing quasi- and non-deterministic transducers

Closing comments

Index

Index

Index

Lexical transformations in NLP systems

Aligned and unaligned dictionaries

Transducers: quasideterministic and nondeterministic

Building transducers from dictionaries

Comparing quasi- and non-deterministic transducers

Closing comments

Index

Lexical transformations in NLP systems

Aligned and unaligned dictionaries

Transducers: quasideterministic and nondeterministic

Building transducers from dictionaries

Comparing quasi- and non-deterministic transducers

Closing comments

Index

Lexical transformations in NLP systems

Aligned and unaligned dictionaries

Transducers: quasideterministic and nondeterministic

Building transducers from dictionaries

Comparing quasi- and non-deterministic transducers

Closing comments

Lexical transformations

Lexical transformations

Lexical transformations in NLP systems:

Lexical transformations

Lexical transformations in NLP systems:

- Morphological analysis: surface form $\rightarrow$ lexical form(s) [lemma $+$ PoS $+$ inflection info.]

Lexical transformations

Lexical transformations in NLP systems:

- Morphological analysis: surface form $\longrightarrow$ lexical form(s) [lemma $+$ PoS $+$ inflection info.]
- Morphological generation: lexical form $\longrightarrow$ surface form.

Lexical transformations

Lexical transformations in NLP systems:

- Morphological analysis: surface form $\longrightarrow$ lexical form(s) [lemma + PoS + inflection info.]
- Morphological generation: lexical form $\longrightarrow$ surface form.
- Lexical transfer (in MT): source lexical form $\longrightarrow$ target lexical form.

Lexical transformations

Lexical transformations in NLP systems:

- Morphological analysis: surface form $\longrightarrow$ lexical form(s) [lemma $+$ PoS $+$ inflection info.]
- Morphological generation: lexical form $\longrightarrow$ surface form.
- Lexical transfer (in MT): source lexical form $\longrightarrow$ target lexical form.

Transformations usually specified in terms of (morphological, bilingual) dictionaries.

Aligned and unaligned dictionaries

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

(`recordáis`, `recordar<vblex><pri><2><pl>`)

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

(`recordáis`, `recordar<vblex><pri><2><pl>`)
(`recuerdo`, `recordar<vblex><pri><1><sg>`)

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

(`recordáis`, `recordar<vblex><pri><2><pl>`)
(`recuerdo`, `recordar<vblex><pri><1><sg>`)
(`recuerdo`, `recuerdo<n><m><sg>`)

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

```
(recordáis, recordar<vblex><pri><2><pl>)
(recuerdo, recordar<vblex><pri><1><sg>)
(recuerdo, recuerdo<n><m><sg>)
```

Aligned dictionary: list of sequences of (input substring, output substring) pairs expressing linguistic regularities.

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

(recordáis, recordar<vblex><pri><2><pl>)
(recuerdo, recordar<vblex><pri><1><sg>)
(recuerdo, recuerdo<n><m><sg>)

Aligned dictionary: list of sequences of (input substring, output substring) pairs expressing linguistic regularities.

(re, re)(c, c)(o, o)(rd, rd)(áis, ar<vblex><2><pl>)

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

(recordáis, recordar<vblex><pri><2><pl>)
(recuerdo, recordar<vblex><pri><1><sg>)
(recuerdo, recuerdo<n><m><sg>)

Aligned dictionary: list of sequences of (input substring, output substring) pairs expressing linguistic regularities.

(re, re)(c, c)(o, o)(rd, rd)(áis, ar<vblex><2><pl>)
(re, re)(c, c)(ue, o)(rd, rd)(o, ar<vblex><1><sg>)

Aligned and unaligned dictionaries

Unaligned dictionary: simple list of (input string, output string) pairs.

(recordáis, recordar<vblex><pri><2><pl>)
(recuerdo, recordar<vblex><pri><1><sg>)
(recuerdo, recuerdo<n><m><sg>)

Aligned dictionary: list of sequences of (input substring, output substring) pairs expressing linguistic regularities.

(re, re)(c, c)(o, o)(rd, rd)(áis, ar<vblex><2><pl>)
(re, re)(c, c)(ue, o)(rd, rd)(o, ar<vblex><1><sg>)
(re, re)(c, c)(ue, ue)(rd, rd)(o, o<n><m><sg>)

Transducers: quasi- and non-deterministic/1

Transducers: quasi- and non-deterministic/1

Many lexical transformations in Indoeuropean languages may be performed sequentially using transducers:

Transducers: quasi- and non-deterministic/1

Many lexical transformations in Indoeuropean languages may be performed sequentially using transducers:

- reading the input left to right;

Transducers: quasi- and non-deterministic/1

Many lexical transformations in Indoeuropean languages may be performed sequentially using transducers:

- reading the input left to right;
- incrementally building:

Transducers: quasi- and non-deterministic/1

Many lexical transformations in Indoeuropean languages may be performed sequentially using transducers:

- reading the input left to right;
- incrementally building:
  - a prefix of the output (deterministic transducers), or

Transducers: quasi- and non-deterministic/1

Many lexical transformations in Indoeuropean languages may be performed sequentially using transducers:

- reading the input left to right;
- incrementally building:
  - a prefix of the output (deterministic transducers), or
  - a set of candidate prefixes of the output (nondeterministic transducers).

Transducers: quasi- and non-deterministic/1

Many lexical transformations in Indoeuropean languages may be performed sequentially using transducers:

- reading the input left to right;
- incrementally building:
  - a prefix of the output (deterministic transducers), or
  - a set of candidate prefixes of the output (nondeterministic transducers).

Sequential processing possible because inputs sharing a prefix correspond to outputs sharing a nontrivial prefix.

Transducers: quasi- and non-deterministic/2

Transducers: quasi- and non-deterministic/2

Deterministic, incremental processing: deliver the longest common output prefix corresponding to all inputs sharing the current input prefix.

Transducers: quasi- and non-deterministic/2

Deterministic, incremental processing: deliver the longest common output prefix corresponding to all inputs sharing the current input prefix.

In deterministic ("earliest p-subsequential" transducers):

Transducers: quasi- and non-deterministic/2

Deterministic, incremental processing: deliver the longest common output prefix corresponding to all inputs sharing the current input prefix.

In deterministic ("earliest p-subsequential" transducers):

- states represent sets of prefixes sharing a common output behavior;

Transducers: quasi- and non-deterministic/2

Deterministic, incremental processing: deliver the longest common output prefix corresponding to all inputs sharing the current input prefix.

In deterministic ("earliest p-subsequential" transducers):

- states represent sets of prefixes sharing a common output behavior;
- a single state is reached for each state and input symbol;

Transducers: quasi- and non-deterministic/2

Deterministic, incremental processing: deliver the longest common output prefix corresponding to all inputs sharing the current input prefix.

In deterministic (“earliest p-subsequential” transducers):

- states represent sets of prefixes sharing a common output behavior;
- a single state is reached for each state and input symbol;
- output is associated to state-to-state transitions: the longest common output prefix is built incrementally.

Transducers: quasi- and non-deterministic/2

Deterministic, incremental processing: deliver the longest common output prefix corresponding to all inputs sharing the current input prefix.

In deterministic ("earliest p-subsequential" transducers):

- states represent sets of prefixes sharing a common output behavior;
- a single state is reached for each state and input symbol;
- output is associated to state-to-state transitions: the longest common output prefix is built incrementally.

Dictionary alignments ignored: "deterministic alignment"

[Details]

Transducers: quasi- and non-deterministic/3

Transducers: quasi- and non-deterministic/3

Full determinism impossible (hence the name quasideterministic) due to one-to-many (many $\leq p$) correspondences:

Transducers: quasi- and non-deterministic/3

Full determinism impossible (hence the name quasideterministic) due to one-to-many (many $\leq p$) correspondences:

- only the longest common output prefix of all outputs (a proper prefix) can be output at the end of the input

Transducers: quasi- and non-deterministic/3

Full determinism impossible (hence the name quasideterministic) due to one-to-many (many $\leq p$) correspondences:

- only the longest common output prefix of all outputs (a proper prefix) can be output at the end of the input

$$\begin{aligned} \tau(\texttt{recuerdo}) &= \{\texttt{recordar<vblex>}\ldots,\texttt{recuerdo<n>}\ldots\} \\ \mathsf{LCP}(\tau(\texttt{recuerdo})) &= \texttt{rec} \end{aligned}$$

Transducers: quasi- and non-deterministic/3

Full determinism impossible (hence the name quasideterministic) due to one-to-many (many $\leq p$) correspondences:

- only the longest common output prefix of all outputs (a proper prefix) can be output at the end of the input

$$\begin{aligned} \tau(\texttt{recuerdo}) &= \{\texttt{recordar<vblex>}\ldots,\texttt{recuerdo<n>}\ldots\} \\ \mathrm{LCP}(\tau(\texttt{recuerdo})) &= \texttt{rec} \end{aligned}$$

- (at most p) output suffixes have to be appended at acceptance states.

Transducers: quasi- and non-deterministic/3

Full determinism impossible (hence the name quasideterministic) due to one-to-many (many $\leq p$) correspondences:

- only the longest common output prefix of all outputs (a proper prefix) can be output at the end of the input

$$\begin{aligned}\tau(\texttt{recuerdo}) &= \{\texttt{recordar<vblex>}\ldots,\texttt{recuerdo<n>}\ldots\}\\ \mathsf{LCP}(\tau(\texttt{recuerdo})) &= \texttt{rec}\end{aligned}$$

- (at most p) output suffixes have to be appended at acceptance states.

$$(\mathsf{rec})^{-1}\tau(\mathsf{recuerdo}) = \{\texttt{ordar<vblex>}\ldots,\texttt{uerdo<n>}\ldots\}$$

Transducers: quasi- and non-deterministic/4

Transducers: quasi- and non-deterministic/4

Disadvantages of quasideterministic transducers:

Transducers: quasi- and non-deterministic/4

Disadvantages of quasideterministic transducers:

- Any linguistic knowledge encoded in dictionary alignments is thrown away.

Transducers: quasi- and non-deterministic/4

Disadvantages of quasideterministic transducers:

- Any linguistic knowledge encoded in dictionary alignments is thrown away.
- For large dictionaries, irregularities may lead to very short longest common output prefixes and very long output suffixes.

Transducers: quasi- and non-deterministic/4

Disadvantages of quasideterministic transducers:

- Any linguistic knowledge encoded in dictionary alignments is thrown away.
- For large dictionaries, irregularities may lead to very short longest common output prefixes and very long output suffixes.
- Adding a new dictionary entry may force a complete reconstruction (longest common output prefixes may change)

Transducers: quasi- and non-deterministic/5

Transducers: quasi- and non-deterministic/5

Nondeterministic transducers avoid this by maintaining several output prefix candidates for each input:

Transducers: quasi- and non-deterministic/5

Nondeterministic transducers avoid this by maintaining several output prefix candidates for each input:

- more than one state may be reached for each state and input symbol;

Transducers: quasi- and non-deterministic/5

Nondeterministic transducers avoid this by maintaining several output prefix candidates for each input:

- more than one state may be reached for each state and input symbol;
- output is associated to state-to-state transitions so that a set of output prefix candidates is built incrementally by maintaining a set of alive state-output pairs during processing;

Transducers: quasi- and non-deterministic/5

Nondeterministic transducers avoid this by maintaining several output prefix candidates for each input:

- more than one state may be reached for each state and input symbol;
- output is associated to state-to-state transitions so that a set of output prefix candidates is built incrementally by maintaining a set of alive state-output pairs during processing;
- output suffixes are no longer necessary.

Transducers: quasi- and non-deterministic/6

Transducers: quasi- and non-deterministic/6

Advantages of nondeterministic transducers:

Transducers: quasi- and non-deterministic/6

Advantages of nondeterministic transducers:

- May be very compact! (when linguists are good at finding regularities to align inputs and outputs) (see later).

Transducers: quasi- and non-deterministic/6

Advantages of nondeterministic transducers:

- May be very compact! (when linguists are good at finding regularities to align inputs and outputs) (see later).
- When expressed as finite-state letter transducers (with transitions reading or writing at most one symbol), they may be determinized and minimized similarly to finite automata.

Transducers: quasi- and non-deterministic/6

Advantages of nondeterministic transducers:

- May be very compact! (when linguists are good at finding regularities to align inputs and outputs) (see later).
- When expressed as finite-state letter transducers (with transitions reading or writing at most one symbol), they may be determinized and minimized similarly to finite automata.
- New entries may be added and removed without realignment and maintaining minimality (Garrido et al., TMI-2002).

[Details]

Building transducers from dictionaries/1

Building quasideterministic transducers from unaligned dictionaries [Details]

Building transducers from dictionaries/1

Building quasideterministic transducers from unaligned dictionaries [Details]

1. Build a **trie** for the input strings of the dictionary (each prefix in the input vocabulary is a state)

Building transducers from dictionaries/1

Building quasideterministic transducers from unaligned dictionaries [Details]

1. Build a **trie** for the input strings of the dictionary (each prefix in the input vocabulary is a state)
2. Using the output strings, compute the longest common output prefix (LCOP) for each prefix

Building transducers from dictionaries/1

Building quasideterministic transducers from unaligned dictionaries [Details]

1. Build a **trie** for the input strings of the dictionary (each prefix in the input vocabulary is a state)
2. Using the output strings, compute the longest common output prefix (LCOP) for each prefix
3. Associate as output of each transition the suffix necessary to get the arrival state LCOP from the departure state LCOP

Building transducers from dictionaries/1

Building quasideterministic transducers from unaligned dictionaries [Details]

1. Build a **trie** for the input strings of the dictionary (each prefix in the input vocabulary is a state)
2. Using the output strings, compute the longest common output prefix (LCOP) for each prefix
3. Associate as output of each transition the suffix necessary to get the arrival state LCOP from the departure state LCOP
4. Compute the remaining output suffixes necessary to complete the output at each acceptance state from the LCOP of that state

Building transducers from dictionaries/1

Building quasideterministic transducers from unaligned dictionaries [Details]

1. Build a **trie** for the input strings of the dictionary (each prefix in the input vocabulary is a state)
2. Using the output strings, compute the longest common output prefix (LCOP) for each prefix
3. Associate as output of each transition the suffix necessary to get the arrival state LCOP from the departure state LCOP
4. Compute the remaining output suffixes necessary to complete the output at each acceptance state from the LCOP of that state
5. Minimize the resulting transducer

Building transducers from dictionaries/2

Building nondeterministic **transducers from aligned dictionaries** [Details]

Building transducers from dictionaries/2

Building nondeterministic **transducers from aligned dictionaries** [Details]

1. Build a **state path** from the start state to an acceptance state for each aligned pair in the dictionary (with transitions reading or writing zero or one characters)

Building transducers from dictionaries/2

Building nondeterministic transducers from aligned dictionaries [Details]

1. Build a **state path** from the start state to an acceptance state for each aligned pair in the dictionary (with transitions reading or writing zero or one characters)
2. Determinize as a finite automaton using the input-output pairs as the alphabet

Building transducers from dictionaries/2

Building nondeterministic transducers from aligned dictionaries [Details]

1. Build a **state path** from the start state to an acceptance state for each aligned pair in the dictionary (with transitions reading or writing zero or one characters)
2. Determinize as a finite automaton using the input-output pairs as the alphabet
3. Minimize in the same way

Comparing quasi- and non-deterministic transducers/1 [Details]

Comparing quasi- and non-deterministic transducers/1 [Details]

- Build both kinds of transducers from a set of representative dictionaries

Comparing quasi- and non-deterministic transducers/1 [Details]

- Build both kinds of transducers from a set of representative dictionaries
- Convert quasideterministic transducers also into finite-state letter transducers

Comparing quasi- and non-deterministic transducers/1 [Details]

- Build both kinds of transducers from a set of representative dictionaries
- Convert quasideterministic transducers also into finite-state letter transducers
  - unfolding transitions with outputs longer than 1

Comparing quasi- and non-deterministic transducers/1 [Details]

- Build both kinds of transducers from a set of representative dictionaries
- Convert quasideterministic transducers also into finite-state letter transducers
  - unfolding transitions with outputs longer than 1
  - creating letter-by-letter state paths for output suffixes at acceptance states

Comparing quasi- and non-deterministic transducers/1 [Details]

- Build both kinds of transducers from a set of representative dictionaries
- Convert quasideterministic transducers also into finite-state letter transducers
  - unfolding transitions with outputs longer than 1
  - creating letter-by-letter state paths for output suffixes at acceptance states
- Determinize and minimize the resulting letter transducers

Comparing quasi- and non-deterministic transducers/1 [Details]

- Build both kinds of transducers from a set of representative dictionaries
- Convert quasideterministic transducers also into finite-state letter transducers
  - unfolding transitions with outputs longer than 1
  - creating letter-by-letter state paths for output suffixes at acceptance states
- Determinize and minimize the resulting letter transducers
- Compare (unfair without conversion: LTs are more “rudimentary”)

Comparing quasi- and non-deterministic transducers/2

Results:

Comparing quasi- and non-deterministic transducers/2

Results:

- Without conversion, both kinds of transducers have roughly the same number of states (comparison unfair to LT)

Comparing quasi- and non-deterministic transducers/2

Results:

- Without conversion, both kinds of transducers have roughly the same number of states (comparison unfair to LT)
- After conversion, nondeterministic transducers are consistently 2.5 times more compact than quasideterministic transducers

Comparing quasi- and non-deterministic transducers/2

Results:

- Without conversion, both kinds of transducers have roughly the same number of states (comparison unfair to LT)
- After conversion, nondeterministic transducers are consistently 2.5 times more compact than quasideterministic transducers
- Observed nondeterminism (average number of ASOPs) is of the order of corpus-computed ambiguity in dictionaries: quasidet., 1.3; nondet., 1.5–1.9 (slightly worse)

Concluding remarks

Concluding remarks

For lexical transformations, nondeterministic transducers are a viable alternative to quasideterministic transducers:

Concluding remarks

For lexical transformations, nondeterministic transducers are a viable alternative to quasideterministic transducers:

- they are compact

Concluding remarks

For lexical transformations, nondeterministic transducers are a viable alternative to quasideterministic transducers:

- they are compact
- their nondeterminism is limited

Concluding remarks

For lexical transformations, nondeterministic transducers are a viable alternative to quasideterministic transducers:

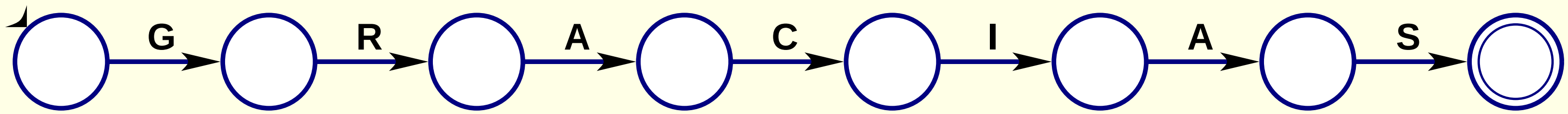
- they are compact
- their nondeterminism is limited
- they are easily maintained

Concluding remarks

For lexical transformations, nondeterministic transducers are a viable alternative to quasideterministic transducers:

- they are compact
- their nondeterminism is limited
- they are easily maintained

Nondeterministic letter transducers are in use in `www.interNOSTRUM.com` (a Spanish–Catalan MT system)


G
R
A
C
I
A
S

Finite-state letter transducers/1

Finite-state letter transducers/1

A (nondeterministic) finite-state letter transducer is

$$T = (Q, L, \delta, q_I, F),$$

Finite-state letter transducers/1

A (nondeterministic) finite-state letter transducer is

$$T = (Q, L, \delta, q_I, F),$$

- Q: finite set of states

Finite-state letter transducers/1

A (nondeterministic) finite-state letter transducer is

$$T = (Q, L, \delta, q_I, F),$$

- Q: finite set of states
- $L = (\Sigma \cup \{\theta\}) \times (\Gamma \cup \{\theta\})$: label alphabet ($\Sigma$: input alphabet, Γ: output alphabet, θ: "empty symbol")

Finite-state letter transducers/1

A (nondeterministic) finite-state letter transducer is

$$T = (Q, L, \delta, q_I, F),$$

- Q: finite set of states
- $L = (\Sigma \cup \{\theta\}) \times (\Gamma \cup \{\theta\})$: label alphabet ($\Sigma$: input alphabet, Γ: output alphabet, θ: "empty symbol")
- $\delta : Q \times L \to 2^Q$: transition function

Finite-state letter transducers/1

A (nondeterministic) finite-state letter transducer is

$$T = (Q, L, \delta, q_I, F),$$

- Q: finite set of states
- $L = (\Sigma \cup \{\theta\}) \times (\Gamma \cup \{\theta\})$: label alphabet ($\Sigma$: input alphabet, Γ: output alphabet, θ: "empty symbol")
- $\delta : Q \times L \to 2^Q$: transition function
- $q_I \in Q$: initial state

Finite-state letter transducers/1

A (nondeterministic) finite-state letter transducer is

$$T = (Q, L, \delta, q_I, F),$$

- Q: finite set of states
- $L = (\Sigma \cup \{\theta\}) \times (\Gamma \cup \{\theta\})$: label alphabet ($\Sigma$: input alphabet, Γ: output alphabet, θ: "empty symbol")
- $\delta : Q \times L \to 2^Q$: transition function
- $q_I \in Q$: initial state
- $F \subseteq Q$: acceptance states

[back]

Finite-state letter transducers/2

Finite-state letter transducers/2

State-to-state arrows have input–output labels (σ, γ):

Finite-state letter transducers/2

State-to-state arrows have input–output labels (σ, γ):

- Input σ can be an input symbol from Σ or nothing (θ)

Finite-state letter transducers/2

State-to-state arrows have input–output labels (σ, γ):

- Input σ can be an input symbol from Σ or nothing (θ)

- Output γ can be an output symbol from Γ or nothing (θ)

Finite-state letter transducers/2

State-to-state arrows have input–output labels (σ, γ):

- Input σ can be an input symbol from Σ or nothing (θ)
- Output γ can be an output symbol from Γ or nothing (θ)

Clearly, (θ, θ) arrows do nothing may be avoided.

[back]

Finite-state letter transducers/3

Finite-state letter transducers/3

Using FSLT: keep a set of alive state–output pairs (SASOP), updated after reading each input symbol from $w = \sigma[1]\sigma[2]\ldots\sigma[|w|]$.

Finite-state letter transducers/3

Using FSLT: keep a set of alive state–output pairs (SASOP), updated after reading each input symbol from $w = \sigma[1]\sigma[2]\ldots\sigma[|w|]$.

$t = 0$, initial SASOP: $\mathcal{V}^{[0]} = \{(q, z) : q \in \delta^*(q_I, (\epsilon, z))\}$, where δ^* is the extension of δ to input–output string pairs

Finite-state letter transducers/3

Using FSLT: keep a set of alive state–output pairs (SASOP), updated after reading each input symbol from $w = \sigma[1]\sigma[2]\ldots\sigma[|w|]$.

$t = 0$, initial SASOP: $\mathcal{V}^{[0]} = \{(q, z) : q \in \delta^*(q_I, (\epsilon, z))\}$, where δ^* is the extension of δ to input–output string pairs

$t \to t + 1$ (after reading $\sigma[t]$):

$$\mathcal{V}^{[t]} = \{(q, z\gamma) : q \in \delta^*(q', (\sigma[t], \gamma)) \wedge \ (q', z) \in \mathcal{V}^{[t-1]}\}$$

Finite-state letter transducers/3

Using FSLT: keep a set of alive state–output pairs (SASOP), updated after reading each input symbol from $w = \sigma[1]\sigma[2]\ldots\sigma[|w|]$.

$t = 0$, initial SASOP: $\mathcal{V}^{[0]} = \{(q,z) : q \in \delta^*(q_I, (\epsilon, z))\}$, where δ^* is the extension of δ to input–output string pairs

$t \to t+1$ (after reading $\sigma[t]$):

$$\mathcal{V}^{[t]} = \{(q, z\gamma) : q \in \delta^*(q', (\sigma[t], \gamma)) \wedge \ (q', z) \in \mathcal{V}^{[t-1]}\}$$

$t = |w|$ (at the end of w): $\tau(w) = \{z : (q,z) \in \mathcal{V}^{[|w|]} \wedge q \in F\}$.

[back]

Longest common output prefix

The longest common output prefix for input w is

$$\mathsf{LCOP}(w) = \mathsf{LCP}(\tau(ww^{-1}E))$$

where

- $E \subset \Sigma^*$ is the vocabulary of inputs,
- $\tau : E \to 2^{\Gamma^*}$ is the transformation function, and
- $ww^{-1}E = \{x \in E \ : \ w \in \mathsf{Pr}(x)\}$.

[back]

Building quasideterministic transducers: details/1

[back]

Building quasideterministic transducers: details/1

Build a p-subsequential transducer $T = (Q, \Sigma, \Gamma, \delta, \lambda, q_I, \psi)$:

[back]

Building quasideterministic transducers: details/1

Build a p-subsequential transducer $T = (Q, \Sigma, \Gamma, \delta, \lambda, q_I, \psi)$:

- With a trie structure: $Q = \mathsf{Pr}(E) \cup \{\perp\}$ ($\perp$ is the absorption state), $q_I = \epsilon$, and

$$\delta(x, \sigma) = \begin{cases} x\sigma & \text{if } x, x\sigma \in \mathsf{Pr}(E) \\ \perp & \text{otherwise} \end{cases}$$

[back]

Building quasideterministic transducers: details/1

Build a p-subsequential transducer $T = (Q, \Sigma, \Gamma, \delta, \lambda, q_I, \psi)$:

- With a trie structure: $Q = \mathsf{Pr}(E) \cup \{\perp\}$ ($\perp$ is the absorption state), $q_I = \epsilon$, and

$$\delta(x, \sigma) = \begin{cases} x\sigma & \text{if } x, x\sigma \in \mathsf{Pr}(E) \\ \perp & \text{otherwise} \end{cases}$$

- With transition outputs $\lambda(x, \sigma) = (\mathsf{LCOP}(x))^{-1}\mathsf{LCOP}(x\sigma)$ for $x, x\sigma \in \mathsf{Pr}(E)$, and undefined otherwise.

[back]

Building quasideterministic transducers: details/1

Build a p-subsequential transducer $T = (Q, \Sigma, \Gamma, \delta, \lambda, q_I, \psi)$:

- With a trie structure: $Q = \mathrm{Pr}(E) \cup \{\perp\}$ ($\perp$ is the absorption state), $q_I = \epsilon$, and

$$\delta(x, \sigma) = \begin{cases} x\sigma & \text{if } x, x\sigma \in \mathrm{Pr}(E) \\ \perp & \text{otherwise} \end{cases}$$

- With transition outputs $\lambda(x, \sigma) = (\mathrm{LCOP}(x))^{-1}\mathrm{LCOP}(x\sigma)$ for $x, x\sigma \in \mathrm{Pr}(E)$, and undefined otherwise.
- With output suffix sets $\psi(w) = (\mathrm{LCOP}(w))^{-1}\tau(w)$.

[back]

Building quasideterministic transducers: details/1

Build a p-subsequential transducer $T = (Q, \Sigma, \Gamma, \delta, \lambda, q_I, \psi)$:

- With a trie structure: $Q = \mathsf{Pr}(E) \cup \{\perp\}$ ($\perp$ is the absorption state), $q_I = \epsilon$, and

$$\delta(x, \sigma) = \begin{cases} x\sigma & \text{if } x, x\sigma \in \mathsf{Pr}(E) \\ \perp & \text{otherwise} \end{cases}$$

- With transition outputs $\lambda(x, \sigma) = (\mathsf{LCOP}(x))^{-1}\mathsf{LCOP}(x\sigma)$ for $x, x\sigma \in \mathsf{Pr}(E)$, and undefined otherwise.
- With output suffix sets $\psi(w) = (\mathsf{LCOP}(w))^{-1}\tau(w)$.

[back]

Building quasideterministic transducers: details/2

The resulting transducer is minimized using the equivalence class algorithm (which iteratively refines a partition of Q).

Two different states q and r are not equivalent if

- $\psi(q) \neq \psi(r)$
- for some σ, $\delta(q, \sigma)$ not in the same class as $\delta(r, \sigma)$
- for some σ, $\lambda(q, \sigma) \neq \lambda(r, \sigma)$.

[back]

Building nondeterministic transducers: details/1

Building nondeterministic transducers: details/1

For each dictionary entry $(a_1, b_1)(a_2, b_2)\ldots(a_N, b_N)$ ...

Building nondeterministic transducers: details/1

For each dictionary entry $(a_1, b_1)(a_2, b_2)\ldots(a_N, b_N)$ ...

... build a path $q_I \overset{(a_1,b_1)}{\longrightarrow} s_1 \overset{(a_2,b_2)}{\longrightarrow} s_2 \ldots \overset{(a_N,b_N)}{\longrightarrow} q_F \ldots$

Building nondeterministic transducers: details/1

For each dictionary entry $(a_1, b_1)(a_2, b_2) \ldots (a_N, b_N)$ ...

... build a path $q_I \stackrel{(a_1,b_1)}{\longrightarrow} s_1 \stackrel{(a_2,b_2)}{\longrightarrow} s_2 \ldots \stackrel{(a_N,b_N)}{\longrightarrow} q_F \ldots$

... from initial state q_I to acceptance state q_F.

Building nondeterministic transducers: details/1

For each dictionary entry $(a_1, b_1)(a_2, b_2) \ldots (a_N, b_N)$ ...

... build a path $q_I \stackrel{(a_1,b_1)}{\longrightarrow} s_1 \stackrel{(a_2,b_2)}{\longrightarrow} s_2 \ldots \stackrel{(a_N,b_N)}{\longrightarrow} q_F \ldots$

... from initial state q_I to acceptance state q_F.

For example, $(\texttt{haces}, \texttt{haz<n><m><pl>}) \ldots$

Building nondeterministic transducers: details/1

For each dictionary entry $(a_1, b_1)(a_2, b_2) \ldots (a_N, b_N)$ ...

... build a path $q_I \stackrel{(a_1,b_1)}{\longrightarrow} s_1 \stackrel{(a_2,b_2)}{\longrightarrow} s_2 \ldots \stackrel{(a_N,b_N)}{\longrightarrow} q_F \ldots$

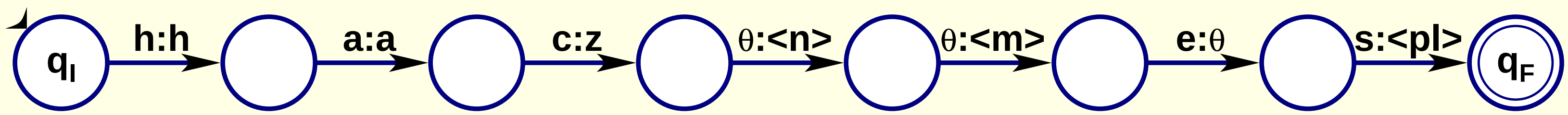
... from initial state q_I to acceptance state q_F.

For example, $(\texttt{haces}, \texttt{haz<n><m><pl>}) \ldots$

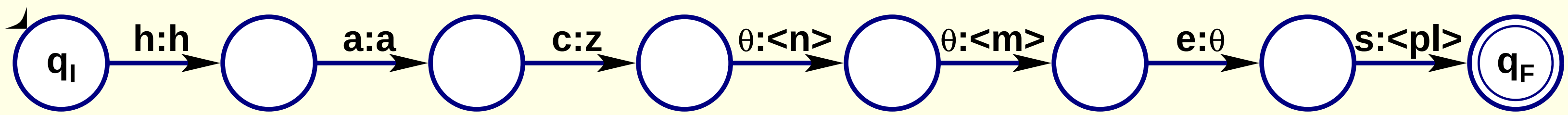
... may be aligned as $(\texttt{h}, \texttt{h})(\texttt{a}, \texttt{a})(\texttt{c}, \texttt{z})(\theta, \texttt{<n>})(\theta, \texttt{<m>})(\texttt{e}, \theta)(\texttt{s}, \texttt{<pl>})$.

[back]

Building nondeterministic transducers: details/2



Building nondeterministic transducers: details/2



The resulting transducer is determinized and minimized.

[back]

Converting into letter transducers: details

Converting into letter transducers: details

- Transitions $q \xrightarrow{(\sigma, \gamma_1 \gamma_2 \ldots \gamma_n)} q'$ with $n > 1$ . . .

Converting into letter transducers: details

- Transitions $q \xrightarrow{(\sigma,\gamma_1\gamma_2\ldots\gamma_n)} q'$ with $n > 1$ . . .

  . . . are unfolded into state paths $q \xrightarrow{(\sigma,\gamma_1)} s_1 \xrightarrow{(\theta,\gamma_2)} s_2 \ldots \xrightarrow{(\theta,\gamma_n)} q'$

Converting into letter transducers: details

- Transitions $q \xrightarrow{(\sigma,\gamma_1\gamma_2\ldots\gamma_n)} q'$ with $n > 1$ ...

  ... are unfolded into state paths $q \xrightarrow{(\sigma,\gamma_1)} s_1 \xrightarrow{(\theta,\gamma_2)} s_2 \ldots \xrightarrow{(\theta,\gamma_n)} q'$

- For each state q and for each tail $\gamma_1\gamma_2\ldots\gamma_n \in \psi(q)$, ...

Converting into letter transducers: details

- Transitions $q \xrightarrow{(\sigma,\gamma_1\gamma_2\ldots\gamma_n)} q'$ with $n > 1$ . . .

  . . . are unfolded into state paths $q \xrightarrow{(\sigma,\gamma_1)} s_1 \xrightarrow{(\theta,\gamma_2)} s_2 \ldots \xrightarrow{(\theta,\gamma_n)} q'$

- For each state q and for each tail $\gamma_1\gamma_2\ldots\gamma_n \in \psi(q)$, . . .

  ldots build an inputless state path $q \xrightarrow{(\theta,\gamma_1)} s_1 \xrightarrow{(\theta,\gamma_2)} s_2 \ldots \xrightarrow{(\theta,\gamma_n)} q_F$

Converting into letter transducers: details

- Transitions $q \xrightarrow{(\sigma,\gamma_1\gamma_2\ldots\gamma_n)} q'$ with $n > 1$ . . .

  . . . are unfolded into state paths $q \xrightarrow{(\sigma,\gamma_1)} s_1 \xrightarrow{(\theta,\gamma_2)} s_2 \ldots \xrightarrow{(\theta,\gamma_n)} q'$

- For each state q and for each tail $\gamma_1\gamma_2\ldots\gamma_n \in \psi(q)$, . . .

  ldots build an inputless state path $q \xrightarrow{(\theta,\gamma_1)} s_1 \xrightarrow{(\theta,\gamma_2)} s_2 \ldots \xrightarrow{(\theta,\gamma_n)}$ q_F (the only source of input nondeterminism).

[back]