Streaming String Transducers

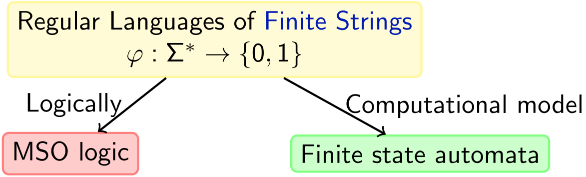
Towards a Theory of Regular Transformations

Ashutosh Trivedi

Department of Computer Science and Engineering,
IIT Bombay

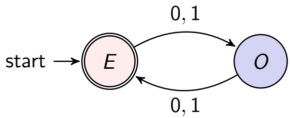
Formal Methods Update Meeting, IIT Delhi (July 27—28, 2013)

Theory of Regular Languages

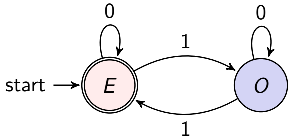


Finite State Automata

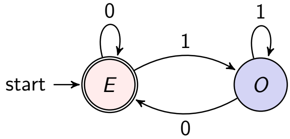
Automaton accepting strings of even length:



Automaton accepting strings with an even number of 1's:



Automaton accepting even strings (multiple of 2):



Monadic Second Order Logic (MSO) over Graphs

– The structure is

$$(\boxed{N}, \boxed{E}, \boxed{L_a, L_b, \ldots, L_k})$$

The domain (set of nodes)

Edge relation $E \subseteq N \times N$

Some unary predicates $L_a, L_b, \ldots, L_k \subseteq N$ partitioning N

Monadic Second Order Logic (MSO) over Graphs

– The structure is

$$(\boxed{N} , \boxed{E} , \boxed{L_a, L_b, \ldots, L_k})$$

The domain (set of nodes)

Edge relation $E \subseteq N \times N$

Some unary predicates $L_a, L_b, \ldots, L_k \subseteq N$ partitioning N

– Strings are interpreted structures: e.g. $(\{1, \ldots, 10\}, E, L_a, L_b, L_c)$

$$\begin{array}{lllllllllll}
s = & a & b & b & a & b & c & a & b & c & c \\
L_a = \{ & 1, & & & 4, & & & 7\} & & & \\
L_b = \{ & & 2, & 3, & & 5, & & & 8\} & & \\
L_c = \{ & & & & & & 6, & & & 9, & 10\}
\end{array}$$

Monadic Second Order Logic (MSO) over Graphs

- The structure is

$$(\boxed{N}, \boxed{E}, \boxed{L_a, L_b, \ldots, L_k})$$

The domain (set of nodes)

Edge relation $E \subseteq N \times N$

Some unary predicates $L_a, L_b, \ldots, L_k \subseteq N$ partitioning N

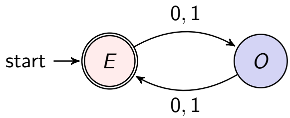
- Strings are interpreted structures: e.g. $(\{1, \ldots, 10\}, E, L_a, L_b, L_c)$

$$\begin{array}{lllllllllll} s = & a & b & b & a & b & c & a & b & c & c \\ L_a = \{ & 1, & & & 4, & & & 7\} & & & \\ L_b = \{ & & 2, & 3, & & 5, & & & 8\} & & \\ L_c = \{ & & & & & & 6, & & & 9, & 10\} \end{array}$$

- Formulas are defined inductively:
  - first-order variables: x, y, z ranging over nodes
  - second-order variables: X, Y, Z ranging over node sets
  - Atomic formulas: $E(x, y)$, $L_a(x)$, $x = y$ and $x \in X$, ...
  - Boolean connectives: $\varphi_1 \wedge \varphi_2$, $\neg\varphi_3$, ...
  - First-order quantification: $\exists x.\varphi$
  - Second-order quantification: $\exists X.\varphi$

Examples

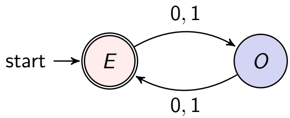
Set of strings with an even number of letters:



- Consider two sets of positions Even and Odd.
- Both sets are disjoint.
- First position is in Odd and the last position is in Even.
- For each position in Even the next position (if exists) is in Odd and vice-versa.

Examples

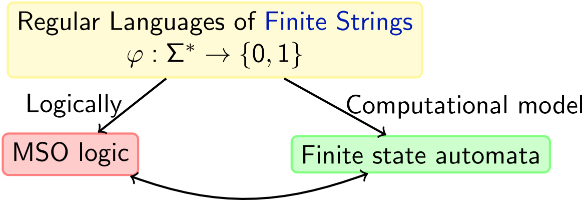
Set of strings with an even number of letters:



- Consider two sets of positions Even and Odd.
- Both sets are disjoint.
- First position is in Odd and the last position is in Even.
- For each position in Even the next position (if exists) is in Odd and vice-versa.

$$
\begin{aligned}
&\exists Odd.\exists Even.\\
&(\forall x.((x \in Odd) \rightarrow \neg(x \in Even) \wedge ((x \in Even) \rightarrow \neg(x \in Odd)))\\
&\wedge First(x) \rightarrow (x \in Odd)\\
&\wedge Last(x) \rightarrow (x \in Even)\\
&\forall x \forall y((x \in Odd) \wedge E(x, y)) \rightarrow y \in Even\\
&\forall x \forall y((x \in Even) \wedge E(x, y)) \rightarrow y \in Odd).
\end{aligned}
$$

Theory of Regular Languages

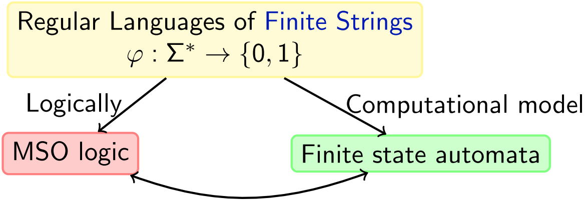


Equi-expressiveness [Büchi, 1960, Elgot, 1961, Trakhtenbrot, 1962]

Theorem ([Büchi, 1960, Elgot, 1961, Trakhtenbrot, 1962])

A language of finite strings is accepted by a finite state automaton iff it is MSO-definable.

Theory of Regular Languages



Equi-expressiveness [Büchi, 1960, Elgot, 1961, Trakhtenbrot, 1962]

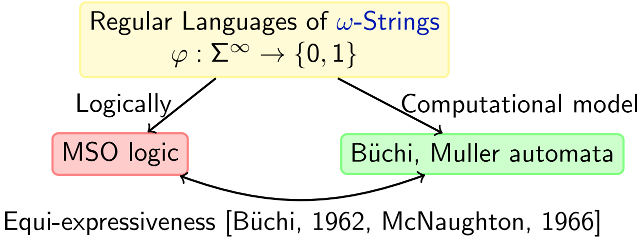
Theorem ([Büchi, 1960, Elgot, 1961, Trakhtenbrot, 1962])

A language of finite strings is accepted by a finite state automaton iff it is MSO-definable.

Why bother?

- new tools to solve problems in logic
- revolutionized the field of automata theory as Büchi initiated the study of equivalent finite state models for MSO over infinite strings.

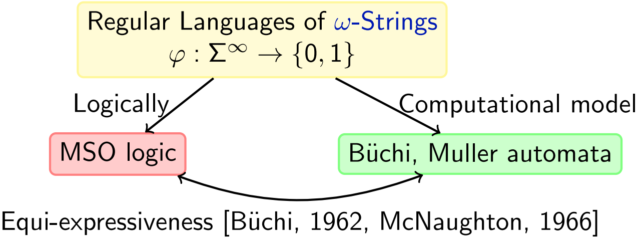
Theory of Regular Languages



Theorem ([Büchi, 1962, McNaughton, 1966])

An language of infinite strings is accepted by a Muller automaton iff it is MSO-definable.

Theory of Regular Languages

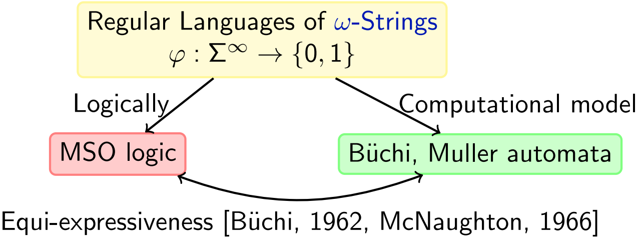


Theorem ([Büchi, 1962, McNaughton, 1966])

An language of infinite strings is accepted by a Muller automaton iff it is MSO-definable.

Since then the theory of regular languages has been lifted to languages of Trees [Rabin, 1969], partial-orders [Thomas, 1995], and more.

Theory of Regular Languages



Theorem ([Büchi, 1962, McNaughton, 1966])

An language of infinite strings is accepted by a Muller automaton iff it is MSO-definable.

Since then the theory of regular languages has been lifted to languages of Trees [Rabin, 1969], partial-orders [Thomas, 1995], and more.

Can we go beyond Languages!

Theory of Regular Transformations

Regular Transformations of Finite Strings
$\varphi : \Sigma^* \to \Gamma^*$

Logically

MSO logic [Courcelle, 1994]

Computational model

Finite state transducers

Theory of Regular Transformations

Regular Transformations of Finite Strings
$\varphi : \Sigma^* \to \Gamma^*$

Logically

Computational model

MSO logic [Courcelle, 1994]

(two-way) Finite state transducers

Equi-expressiveness [Engelfriet and Hoogeboom, 2001]

Theory of Regular Transformations

Regular Transformations of Finite Strings
$\varphi : \Sigma^* \rightarrow \Gamma^*$

Logically

Computational model

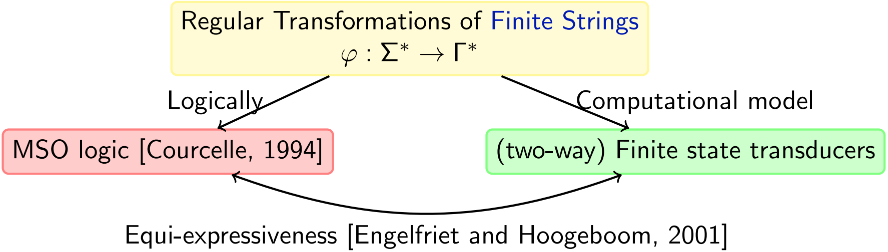
MSO logic [Courcelle, 1994]

(two-way) Finite state transducers

Equi-expressiveness [Engelfriet and Hoogeboom, 2001]

- MSO-definable transformations can be naturally extended to define transformations for more general structures
- Unfortunately, two-way finite state transducers can not naturally be generalized with such ease

Theory of Regular Transformations



- MSO-definable transformations can be naturally extended to define transformations for more general structures
- Unfortunately, two-way finite state transducers can not naturally be generalized with such ease
- Also, it would be nice to have a one-way (streaming) transducer precisely capturing the class of MSO-definable transformations

Streaming String Transducers

- Alur and Černý introduced streaming string transducers (SSTs) to model and analyze single-pass list processing programs [Alur and Černý, 2010], e.g.
  - imperative programs manipulating heap-allocated lists
  - functional programs using tail recursion
  - commonly used routines include insert, delete, and reverse.

Streaming String Transducers

- Alur and Černý introduced streaming string transducers (SSTs) to model and analyze single-pass list processing programs [Alur and Černý, 2010], e.g.
  - imperative programs manipulating heap-allocated lists
  - functional programs using tail recursion
  - commonly used routines include insert, delete, and reverse.
- decidable (PSPACE) functional equivalence and verification (pre/post condition) problem

Streaming String Transducers

- Alur and Černý introduced streaming string transducers (SSTs) to model and analyze single-pass list processing programs [Alur and Černý, 2010], e.g.
  - imperative programs manipulating heap-allocated lists
  - functional programs using tail recursion
  - commonly used routines include insert, delete, and reverse.
- decidable (PSPACE) functional equivalence and verification (pre/post condition) problem
- first one-way (streaming) transducer model that precisely captures the MSO-definable transformations

Streaming String Transducers

- Alur and Černý introduced streaming string transducers (SSTs) to model and analyze single-pass list processing programs [Alur and Černý, 2010], e.g.
  - imperative programs manipulating heap-allocated lists
  - functional programs using tail recursion
  - commonly used routines include insert, delete, and reverse.
- decidable (PSPACE) functional equivalence and verification (pre/post condition) problem
- first one-way (streaming) transducer model that precisely captures the MSO-definable transformations
- SSTs naturally generalize to model transformation of more general structures
  - string-to-tree [Alur and D'Antoni, 2012],
  - tree-to-tree [Alur and D'Antoni, 2012],
  - ω-string to ω-strings [Alur et al., 2012],
  - ω-string to ω-trees [Alur et al., 2013b].
  - strings to costs [Alur et al., 2013a]

Streaming String Transducers

- Alur and Černý introduced streaming string transducers (SSTs) to model and analyze single-pass list processing programs [Alur and Černý, 2010], e.g.
  - imperative programs manipulating heap-allocated lists
  - functional programs using tail recursion
  - commonly used routines include insert, delete, and reverse.
- decidable (PSPACE) functional equivalence and verification (pre/post condition) problem
- first one-way (streaming) transducer model that precisely captures the MSO-definable transformations
- SSTs naturally generalize to model transformation of more general structures
  - string-to-tree [Alur and D'Antoni, 2012],
  - tree-to-tree [Alur and D'Antoni, 2012],
  - ω-string to ω-strings [Alur et al., 2012],
  - ω-string to ω-trees [Alur et al., 2013b].
  - strings to costs [Alur et al., 2013a]

Theory of regular transformations

Regular Transformations of Finite Strings

Regular Transformations of Infinite Strings

Conclusion

Transformations of Finite Strings

- A transformation from Σ to Γ is a (partial) function $f : \Sigma^* \to \Gamma^*$.
- Generalizes the concept of a language $f : \Sigma^* \to \{0, 1\}$.
- Example:
  - $a^n \mapsto a^n b^n$
  - $a^n b^m \mapsto a^{2^n - 1} b^m$
  - local transformations, e.g., delete each a, repeat every b
  - reverse transformation, i.e. $a_1 a_2 \ldots a_n \mapsto a_n a_{n-1} \ldots a_1$,
  - swapping transformation, e.g. $\alpha \# \beta \mapsto \beta \# \alpha$,
  - look-ahead based transformations, e.g.
    - replace each a with b if the string contains a $\#$.
    - replace each a with b if the string contains a prime number of $\#$.

Transformations of Finite Strings

- A transformation from Σ to Γ is a (partial) function $f : \Sigma^* \to \Gamma^*$.
- Generalizes the concept of a language $f : \Sigma^* \to \{0, 1\}$.
- Example:
  - $a^n \mapsto a^n b^n$
  - $a^n b^m \mapsto a^{2^n - 1} b^m$
  - local transformations, e.g., delete each a, repeat every b
  - reverse transformation, i.e. $a_1 a_2 \dots a_n \mapsto a_n a_{n-1} \dots a_1$,
  - swapping transformation, e.g. $\alpha \# \beta \mapsto \beta \# \alpha$,
  - look-ahead based transformations, e.g.
    - replace each a with b if the string contains a $\#$.
    - replace each a with b if the string contains a prime number of $\#$.
- A transducer is an abstract machine defining a transformation.
- Transducers generalize the concept of automata

Transformations of Finite Strings

- A transformation from Σ to Γ is a (partial) function $f : \Sigma^* \to \Gamma^*$.
- Generalizes the concept of a language $f : \Sigma^* \to \{0, 1\}$.
- Example:
  - $a^n \mapsto a^n b^n$
  - $a^n b^m \mapsto a^{2^n - 1} b^m$
  - local transformations, e.g., delete each a, repeat every b
  - reverse transformation, i.e. $a_1 a_2 \ldots a_n \mapsto a_n a_{n-1} \ldots a_1$,
  - swapping transformation, e.g. $\alpha \# \beta \mapsto \beta \# \alpha$,
  - look-ahead based transformations, e.g.
    - replace each a with b if the string contains a $\#$.
    - replace each a with b if the string contains a prime number of $\#$.
- A transducer is an abstract machine defining a transformation.
- Transducers generalize the concept of automata
- Similar to languages, a transformation can also be defined using logic, most notably Monadic second-order logic (MSO) over finite strings.

MSO-definable Transformations

Definition (Defining Transformation using MSO)

A transformation using MSO is specified by:

- input and output alphabets;
- an MSO formula specifying the domain of the transformation;
- output string is specified using a finite number of copies of nodes of input string graph;
- the node labels are specified using MSO formulas; and
- the existence of edges between nodes of various copies is specified using MSO formulas

MSO-definable Transformations

Definition (Defining Transformation using MSO)

A transformation using MSO is specified by:

- input and output alphabets;
- an MSO formula specifying the domain of the transformation;
- output string is specified using a finite number of copies of nodes of input string graph;
- the node labels are specified using MSO formulas; and
- the existence of edges between nodes of various copies is specified using MSO formulas

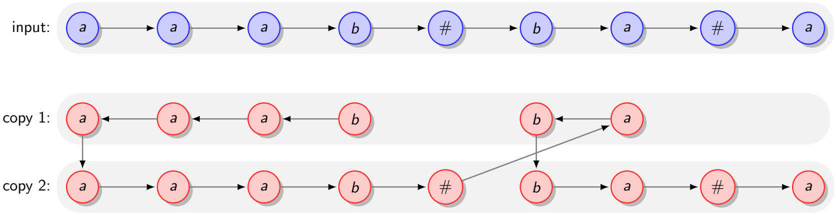
Example

Let $\Sigma = \{a, b, \#\}$. Consider a transformation $f_1 : \Sigma^* \to \Sigma^*$

$$u_1 \# u_2 \# \dots u_{n-1} \# u_n \# v \mapsto \overline{u_1} u_1 \# \dots \# \overline{u_n} u_n \# v.$$

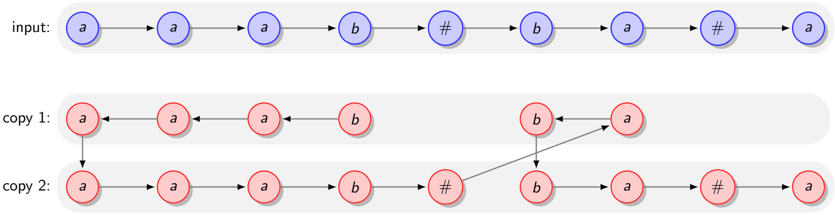
where $\overline{u}$ is reverse of u.

MSO-definable Transformations



- $\Sigma = \Gamma = \{a, b, \#\}$, $C = \{1, 2\}$, and
- Node Label Formulas
  - $\mathrm{Label}^{c1}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x) \wedge \neg\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x)$
  - $\mathrm{Label}^{c2}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x)$
- Edge Label Formulas
  - $\mathrm{Edge}^{c1,c1}(x, y) = \mathrm{Edge}^{\mathrm{inp}}(y, x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(y)$.
  - $\mathrm{Edge}^{c2,c2}(x, y) =$
    $\mathrm{Edge}^{\mathrm{inp}}(x, y) \wedge (\neg\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \vee (\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \neg\mathrm{reach}_{\#}(x)))$
  - $\mathrm{Edge}^{1,2}(x, y) = (x = y) \wedge (\mathrm{first}(x) \vee \exists z (\mathrm{Label}^{\mathrm{inp}}_{\#}(z) \wedge \mathrm{Edge}^{\mathrm{inp}}(z, x)))$
  - $\mathrm{Edge}^{2,1}(x, y) = \mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x) \wedge (\exists z (\mathrm{Edge}^{\mathrm{inp}}(y, z) \wedge$
    $\mathrm{Label}^{\mathrm{inp}}_{\#}(z))) \wedge (\forall z ((\mathrm{path}(x, z) \wedge \mathrm{path}(z, y)) \rightarrow \neg\mathrm{Label}^{\mathrm{inp}}_{\#}(z)))$

MSO-definable Transformations



- $\Sigma = \Gamma = \{a, b, \#\}$, $C = \{1, 2\}$, and
- Node Label Formulas
  - $\mathrm{Label}^{c1}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x) \wedge \neg\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x)$
  - $\mathrm{Label}^{c2}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x)$
- Edge Label Formulas
  - $\mathrm{Edge}^{c1,c1}(x, y) = \mathrm{Edge}^{\mathrm{inp}}(y, x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(y)$.
  - $\mathrm{Edge}^{c2,c2}(x, y) =$ $\mathrm{Edge}^{\mathrm{inp}}(x, y) \wedge (\neg\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \vee (\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \neg\mathrm{reach}_{\#}(x)))$
  - $\mathrm{Edge}^{1,2}(x, y) = (x = y) \wedge (\mathrm{first}(x) \vee \exists z(\mathrm{Label}^{\mathrm{inp}}_{\#}(z) \wedge \mathrm{Edge}^{\mathrm{inp}}(z, x)))$
  - $\mathrm{Edge}^{2,1}(x, y) = \mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x) \wedge (\exists z(\mathrm{Edge}^{\mathrm{inp}}(y, z) \wedge \mathrm{Label}^{\mathrm{inp}}_{\#}(z))) \wedge (\forall z((\mathrm{path}(x, z) \wedge \mathrm{path}(z, y)) \rightarrow \neg\mathrm{Label}^{\mathrm{inp}}_{\#}(z)))$

MSO-definable Transformations

- $a^n \mapsto a^n b^n$ ✓
- $a^n b^m \mapsto a^{2^n - 1} b^m$
- local transformations, e.g., delete each a, repeat every b ✓
- reverse transformation, i.e. $a_1 a_2 \ldots a_n \mapsto a_n a_{n-1} \ldots a_1$, ✓
- swapping transformation, e.g. $\alpha \# \beta \mapsto \beta \# \alpha$, ✓
- look-ahead based transformations, e.g.
  - replace each a with b if the string contains a $\#$ ✓
  - replace each a with b if the string contains a prime number of $\#$

MSO-definable Transformations

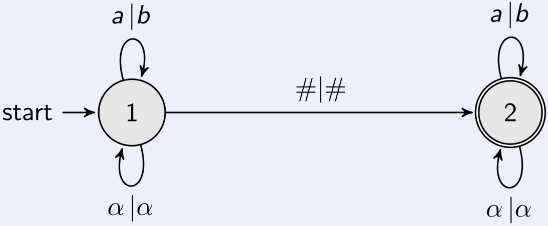
- $a^n \mapsto a^n b^n$ ✓
- $a^n b^m \mapsto a^{2^n - 1} b^m$
- local transformations, e.g., delete each a, repeat every b ✓
- reverse transformation, i.e. $a_1 a_2 \ldots a_n \mapsto a_n a_{n-1} \ldots a_1$, ✓
- swapping transformation, e.g. $\alpha \# \beta \mapsto \beta \# \alpha$, ✓
- look-ahead based transformations, e.g.
  - replace each a with b if the string contains a $\#$ ✓
  - replace each a with b if the string contains a prime number of $\#$

Regular Transformations

Which transducers accept same class of transformations?

Deterministic Generalized Sequential Machines

Example: For all strings containing a $\#$, replace all a with b.

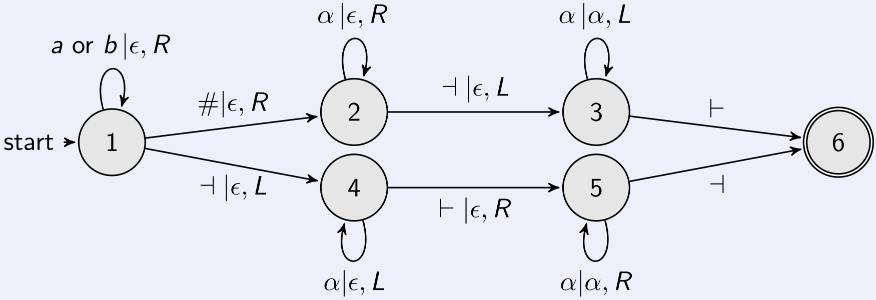


†Here α stands for any symbol other than a.

- Extend finite automata with output
- Can express local transformations
- Can not express reverse, swap, or regular look-ahead
- Non-deterministic variants can express regular look-ahead

2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains a $\#$ then $\overline{u}$ else u

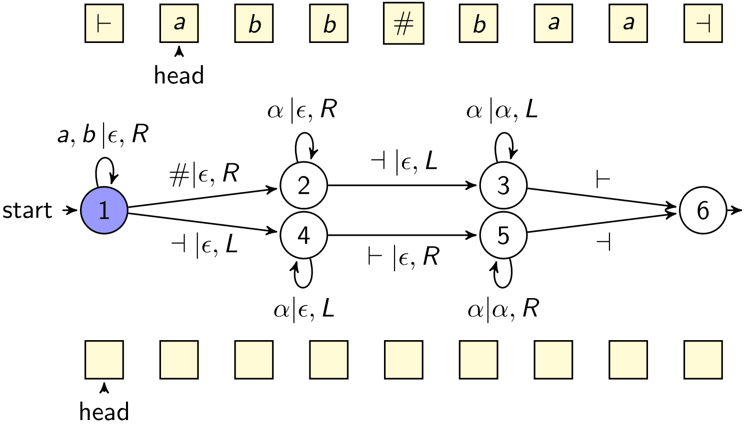


†Here α stands for any symbol except end markers.

- Extend two-way finite automata with output
- Allowing transitions based on regular look-ahead do not increase expressiveness (Chytil and Jakl [1977])
- Two-way finite-state transducers capture the same class of MSO-definable transformations (Engelfriet and Hoogeboom [2001])

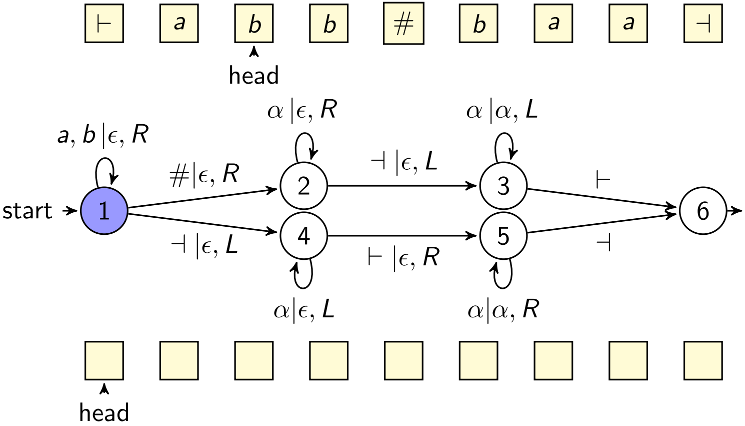
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



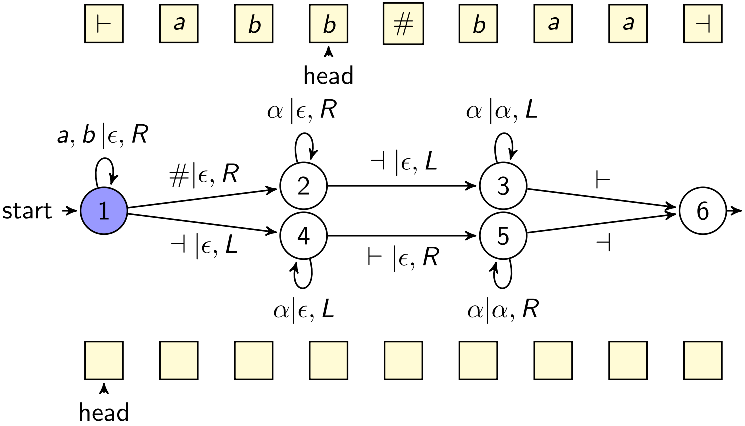
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



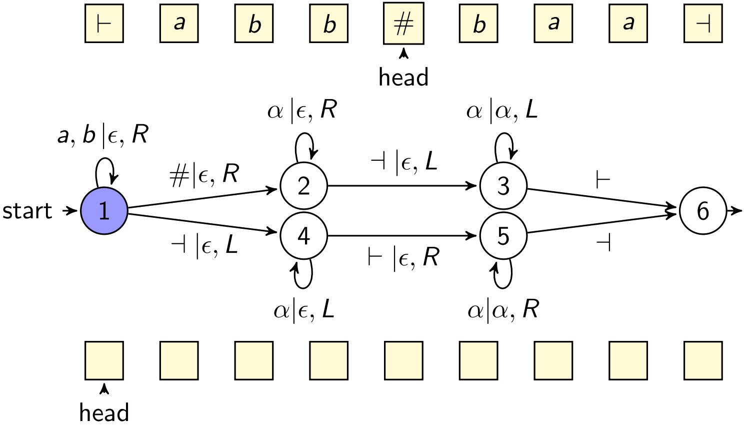
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



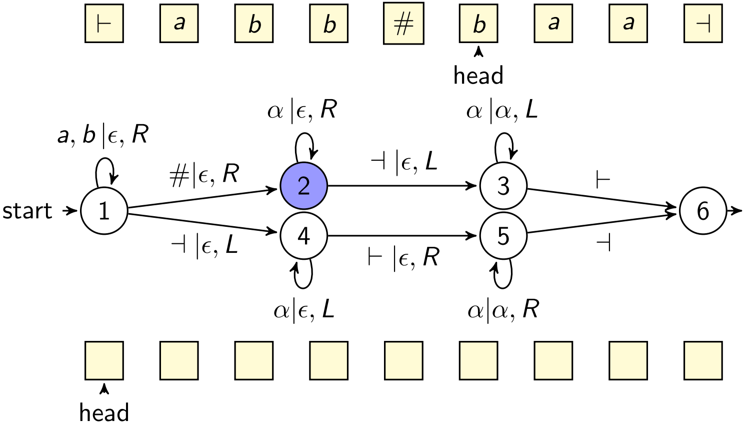
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



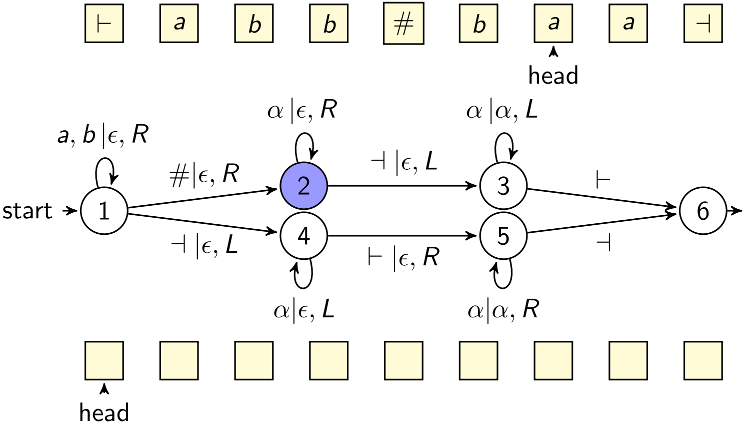
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



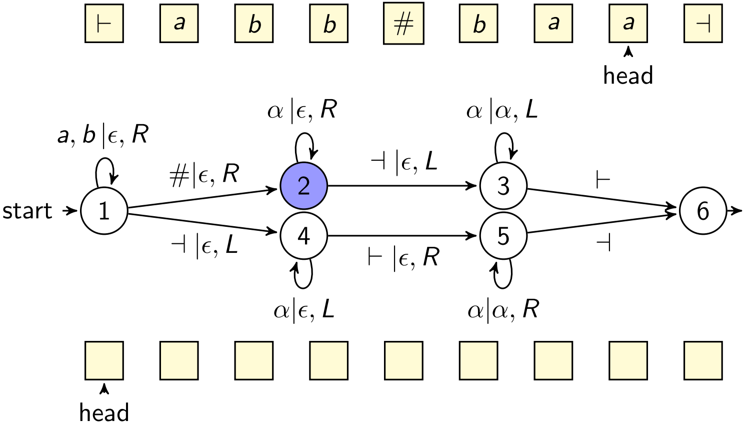
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



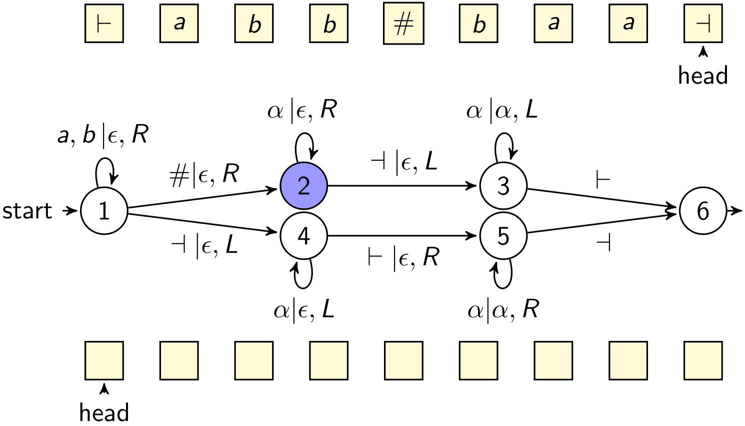
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



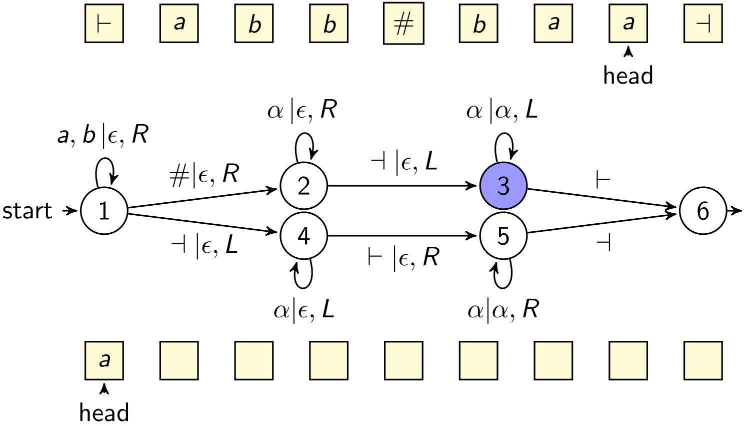
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



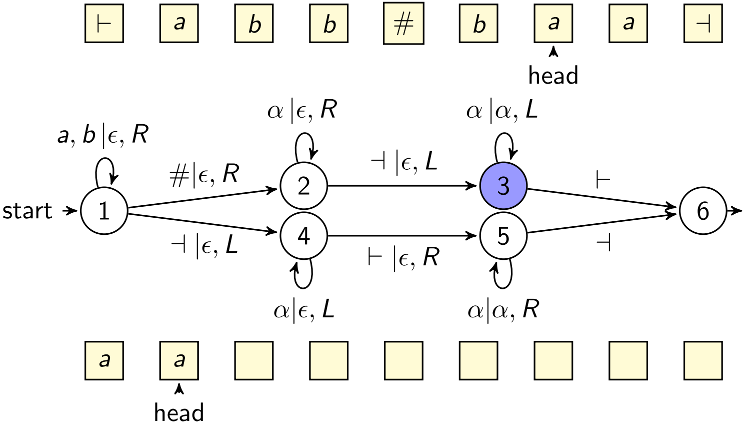
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



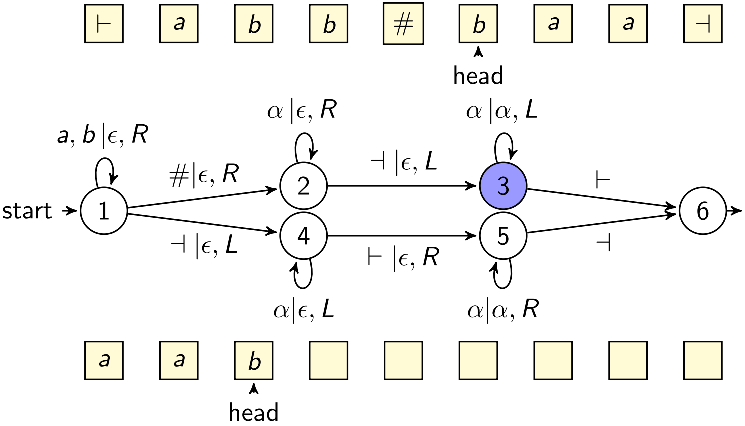
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



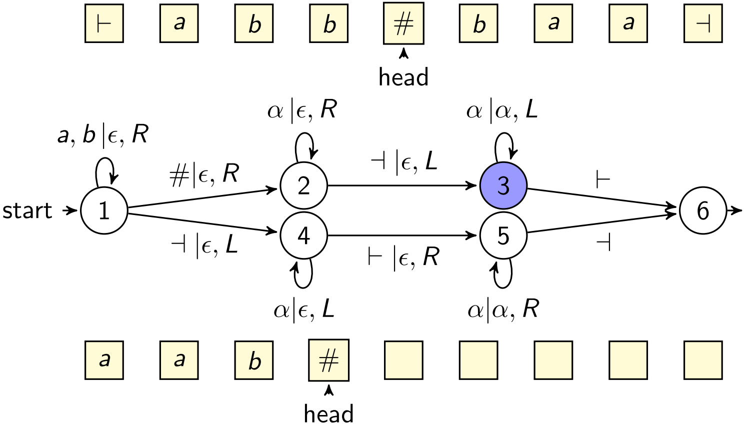
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



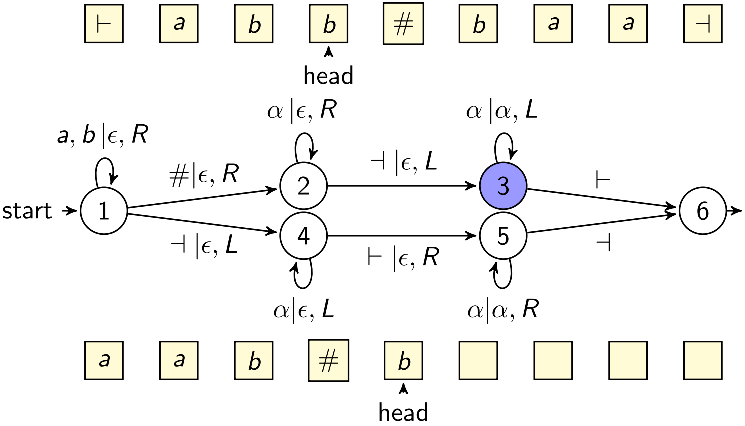
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



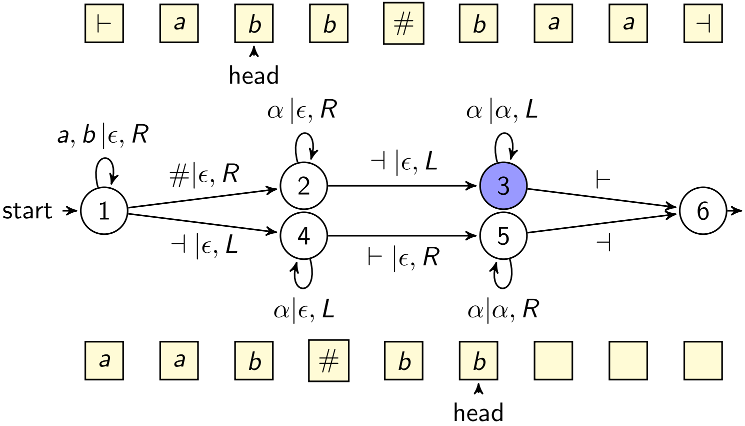
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



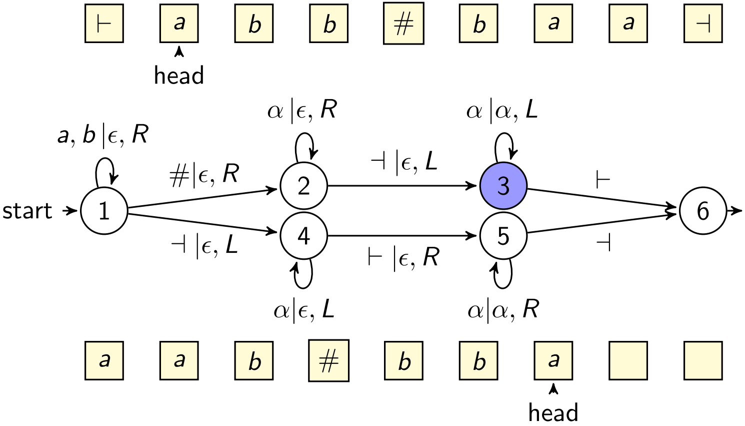
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



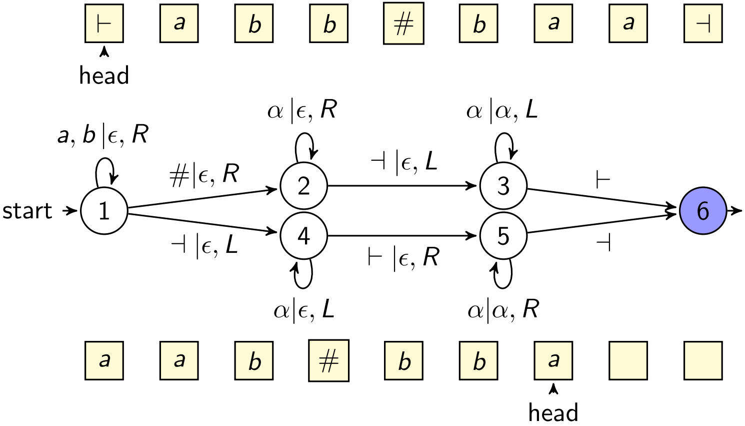
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



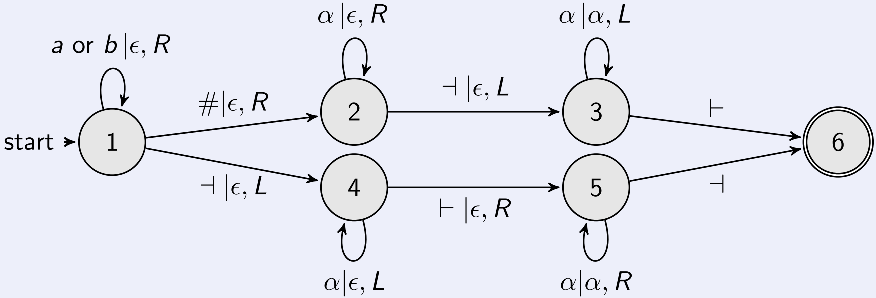
2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains $\#$ then $\overline{u}$ else u



2-Way Deterministic Finite State Transducers

Example: $u \mapsto$ if u contains a $\#$ then $\overline{u}$ else u

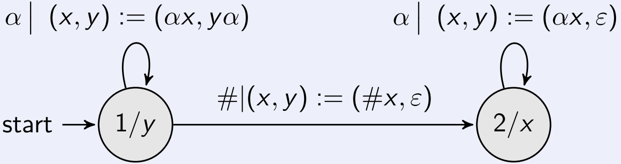


†Here α stands for any symbol except end markers.

- Extend two-way finite automata with output
- Allowing transitions based on regular look-ahead do not increase expressiveness (Chytil and Jakl [1977])
- Two-way finite-state transducers capture the same class of MSO-definable transformations (Engelfriet and Hoogeboom [2001])

Transducers: Streaming String Transducers

Example: $u \mapsto$ if u contains a $\#$ then $\overline{u}$ else u



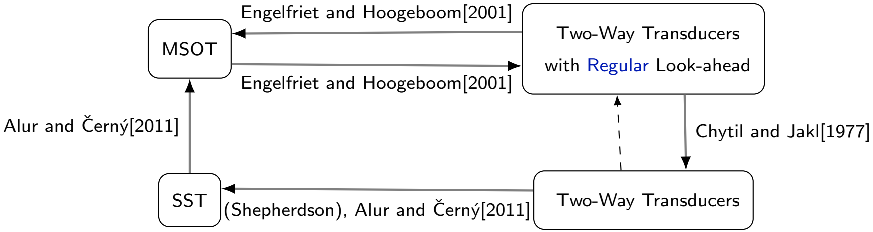
[†] Here α stands for any symbol except end markers.

- Extend deterministic finite-state automata with string variables
- String variables are updated in a copyless fashion
- Output is given as a function of states to copyless concatenation of string variables

Expressiveness of Streaming String Transducers

Theorem ([Alur and Černý, 2011])

A transformation of finite strings is accepted by a streaming string transducer iff it is MSO-definable.



Properties of Regular Transformations

- Characterized by
  - MSO,
  - (deterministic) two-way finite-state transducers, and
  - (deterministic) streaming string transducers.
- They are closed under sequential composition
- Equivalence problem, deciding the equivalence of two regular transformations, is decidable.
- Type checking problem, deciding whether image of a given regular set I under a regular transformation T is contained in another given regular set O i.e. $T(I) \subseteq O$, is decidable.
- Both problems are in PSPACE for streaming-string transducers [Alur and Černý, 2011]

Transformations of Infinite Strings

- A transformation from Σ to Γ is a (partial) function $f : \Sigma^\omega \to \Gamma^\omega$.
- Generalizes the concept of an ω-language $f : \Sigma^\omega \to \{0, 1\}$.
- Example:
  - $a^n \#^\omega \mapsto a^n b^n \#^\omega$
  - $a^n b^\omega \mapsto a^{2^n - 1} b^\omega$
  - local transformations, e.g., delete each a, repeat every b
  - reverse transformation, i.e. $a_1 a_2 \ldots a_n \# u \mapsto a_n a_{n-1} \ldots a_1 \# u$,
  - swapping transformation, e.g. $\alpha \# \beta \# u \mapsto \beta \# \alpha \# u$,
  - look-ahead based transformations,
    - replace each a with b if the string contains a $\#$.
    - replace each a with b if the string contains a prime number of $\#$.

Transformations of Infinite Strings

- A transformation from Σ to Γ is a (partial) function $f : \Sigma^\omega \to \Gamma^\omega$.
- Generalizes the concept of an ω-language $f : \Sigma^\omega \to \{0,1\}$.
- Example:
  - $a^n \#^\omega \mapsto a^n b^n \#^\omega$
  - $a^n b^\omega \mapsto a^{2^n - 1} b^\omega$
  - local transformations, e.g., delete each a, repeat every b
  - reverse transformation, i.e. $a_1 a_2 \ldots a_n \# u \mapsto a_n a_{n-1} \ldots a_1 \# u$,
  - swapping transformation, e.g. $\alpha \# \beta \# u \mapsto \beta \# \alpha \# u$,
  - look-ahead based transformations,
    - replace each a with b if the string contains a $\#$.
    - replace each a with b if the string contains a prime number of $\#$.
- MSO on infinite strings can be used to define transformations on infinite strings [Courcelle, 1994]

Transformations of Infinite Strings

- A transformation from Σ to Γ is a (partial) function $f : \Sigma^\omega \to \Gamma^\omega$.
- Generalizes the concept of an ω-language $f : \Sigma^\omega \to \{0, 1\}$.
- Example:
  - $a^n \#^\omega \mapsto a^n b^n \#^\omega$
  - $a^n b^\omega \mapsto a^{2^n - 1} b^\omega$
  - local transformations, e.g., delete each a, repeat every b
  - reverse transformation, i.e. $a_1 a_2 \dots a_n \# u \mapsto a_n a_{n-1} \dots a_1 \# u$,
  - swapping transformation, e.g. $\alpha \# \beta \# u \mapsto \beta \# \alpha \# u$,
  - look-ahead based transformations,
    - replace each a with b if the string contains a $\#$.
    - replace each a with b if the string contains a prime number of $\#$.
- MSO on infinite strings can be used to define transformations on infinite strings [Courcelle, 1994]
- What classes of **finite-state transducers** have equal expressive power?
- What **decision problems** about MSO-definable transformations of infinite strings can be solved?

MSO-definable Transformations

Definition (Defining Transformation using MSO)

A transformation using MSO is specified by:

- input and output alphabets;
- an MSO formula specifying the domain of the transformation;
- output string is specified using a finite number of copies of nodes of input string graph;
- the node labels are specified using MSO formulas; and
- the existence of edges between nodes of various copies is specified using MSO formulas

MSO-definable Transformations

Definition (Defining Transformation using MSO)

A transformation using MSO is specified by:

- input and output alphabets;
- an MSO formula specifying the domain of the transformation;
- output string is specified using a finite number of copies of nodes of input string graph;
- the node labels are specified using MSO formulas; and
- the existence of edges between nodes of various copies is specified using MSO formulas

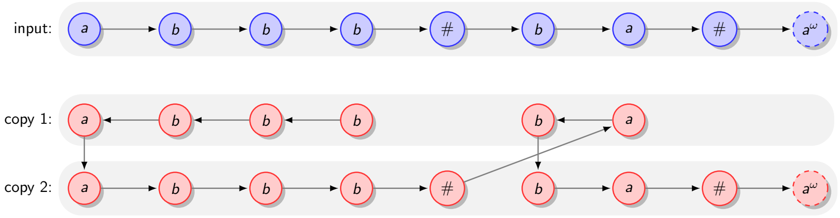
Example

Let $\Sigma = \{a, b, \#\}$. Consider a transformation $f_2 : \Sigma^\omega \rightarrow \Sigma^\omega$

$$u_1 \# u_2 \# \ldots u_{n-1} \# u_n \# v \mapsto \overline{u_1} u_1 \# \ldots \# \overline{u_n} u_n \# v.$$

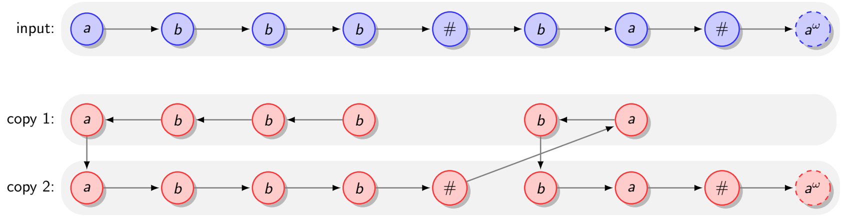
where $\overline{u}$ is reverse of u and $v \in \{a, b\}^\omega$.

MSO-definable Transformations



- $\Sigma = \Gamma = \{a, b, \#\}$, $C = \{1, 2\}$, and
- Node Label Formulas
  - $\mathrm{Label}^{c1}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x) \wedge \neg\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x)$
  - $\mathrm{Label}^{c2}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x)$
- Edge Label Formulas
  - $\mathrm{Edge}^{c1,c1}(x, y) = \mathrm{Edge}^{\mathrm{inp}}(y, x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(y)$.
  - $\mathrm{Edge}^{c2,c2}(x, y) =$
    $\mathrm{Edge}^{\mathrm{inp}}(x, y) \wedge (\neg\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \vee (\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \neg\mathrm{reach}_{\#}(x)))$
  - $\mathrm{Edge}^{1,2}(x, y) = (x = y) \wedge (\mathrm{first}(x) \vee \exists z(\mathrm{Label}^{\mathrm{inp}}_{\#}(z) \wedge \mathrm{Edge}^{\mathrm{inp}}(z, x)))$
  - $\mathrm{Edge}^{2,1}(x, y) = \mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x) \wedge (\exists z(\mathrm{Edge}^{\mathrm{inp}}(y, z) \wedge$
    $\mathrm{Label}^{\mathrm{inp}}_{\#}(z))) \wedge (\forall z((\mathrm{path}(x, z) \wedge \mathrm{path}(z, y)) \rightarrow \neg\mathrm{Label}^{\mathrm{inp}}_{\#}(z)))$

MSO-definable Transformations



- $\Sigma = \Gamma = \{a, b, \#\}$, $C = \{1, 2\}$, and
- Node Label Formulas
  - $\mathrm{Label}^{c1}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x) \wedge \neg \mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x)$
  - $\mathrm{Label}^{c2}_{\alpha}(x) = \mathrm{Label}^{\mathrm{inp}}_{\alpha}(x)$
- Edge Label Formulas
  - $\mathrm{Edge}^{c1,c1}(x, y) = \mathrm{Edge}^{\mathrm{inp}}(y, x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(x) \wedge \mathrm{Label}^{\mathrm{inp}}_{\star}(y)$.
  - $\mathrm{Edge}^{c2,c2}(x, y) =$ $\mathrm{Edge}^{\mathrm{inp}}(x, y) \wedge (\neg \mathrm{Label}^{\mathrm{inp}}_{\#}(x) \vee (\mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \neg \mathrm{reach}_{\#}(x)))$
  - $\mathrm{Edge}^{1,2}(x, y) = (x = y) \wedge (\mathrm{first}(x) \vee \exists z (\mathrm{Label}^{\mathrm{inp}}_{\#}(z) \wedge \mathrm{Edge}^{\mathrm{inp}}(z, x)))$
  - $\mathrm{Edge}^{2,1}(x, y) = \mathrm{Label}^{\mathrm{inp}}_{\#}(x) \wedge \mathrm{reach}_{\#}(x) \wedge (\exists z (\mathrm{Edge}^{\mathrm{inp}}(y, z) \wedge \mathrm{Label}^{\mathrm{inp}}_{\#}(z))) \wedge (\forall z ((\mathrm{path}(x, z) \wedge \mathrm{path}(z, y)) \rightarrow \neg \mathrm{Label}^{\mathrm{inp}}_{\#}(z)))$

MSO-definable Transformations

- $a^n \#^\omega \mapsto a^n b^n \#^\omega$ ✓
- $a^n b^\omega \mapsto a^{2^n - 1} b^\omega$
- local transformations, e.g., delete each a, repeat every b ✓
- reverse transformation, i.e. $a_1 a_2 \dots a_n \# u \mapsto a_n a_{n-1} \dots a_1 \# u$, ✓
- swapping transformation, e.g. $\alpha \# \beta \# u \mapsto \beta \# \alpha \# u$, ✓
- look-ahead based transformations, e.g.
  - replace each a with b if the string contains a $\#$ ✓
  - replace each a with b if the string contains a prime number of $\#$

MSO-definable Transformations

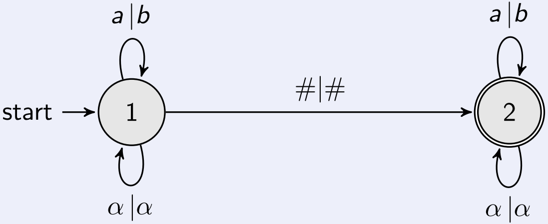
- $a^n \#^\omega \mapsto a^n b^n \#^\omega$ ✓
- $a^n b^\omega \mapsto a^{2^n - 1} b^\omega$
- local transformations, e.g., delete each a, repeat every b ✓
- reverse transformation, i.e. $a_1 a_2 \ldots a_n \# u \mapsto a_n a_{n-1} \ldots a_1 \# u$, ✓
- swapping transformation, e.g. $\alpha \# \beta \# u \mapsto \beta \# \alpha \# u$, ✓
- look-ahead based transformations, e.g.
  - replace each a with b if the string contains a $\#$ ✓
  - replace each a with b if the string contains a prime number of $\#$

Regular Transformations on Infinite Strings

Which transducers accept the same class of transformations?

Deterministic Generalized Sequential Machines

Example: For all strings containing a $\#$, replace all a with b.

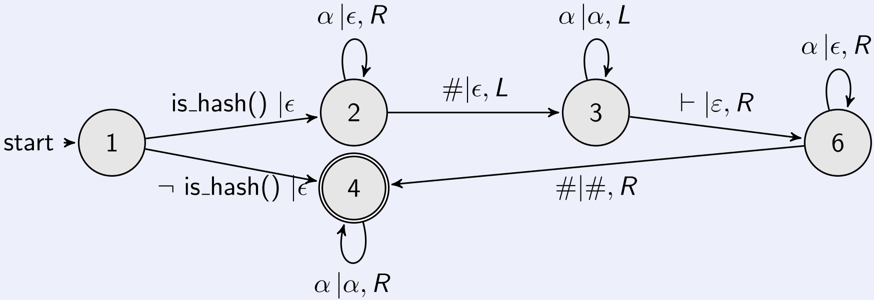


†Here α stands for any symbol other than a.

- Extend Muller automata with output
- Can express local transformations
- Can not express transformations such as reverse or swap

2-Way Transducers with Look-Ahead

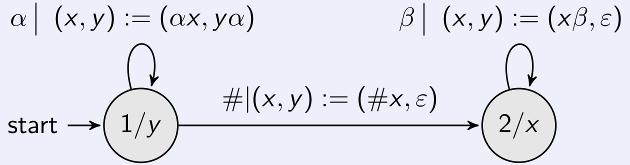
Example: Reverse the sub-string before the first



- Extend two-way Muller automata with output
- Allowing ω-regular look-ahead increases expressiveness
- Two-way finite-state transducers with ω-regular look-ahead capture the same class of transformations as MSO.

SSTs with Muller Acceptance Condition

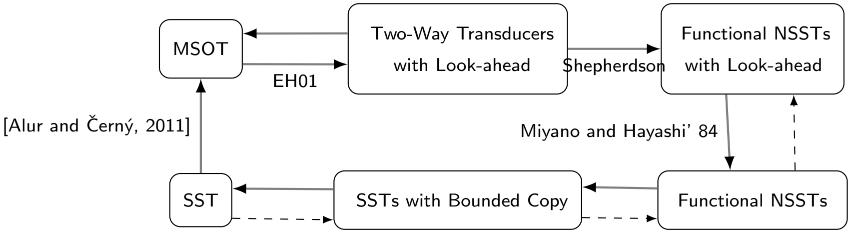
Example: Reverse the sub-string before the first



[†]Here α is any symbol except #, while β is any symbol.

- Extend Muller automata with string variables
- String variables are updated in a copyless fashion
- Output is given as a function of set of states to copyless concatenation of string variables
- We enforce syntactic restrictions that ascertain that output string is always an infinite string

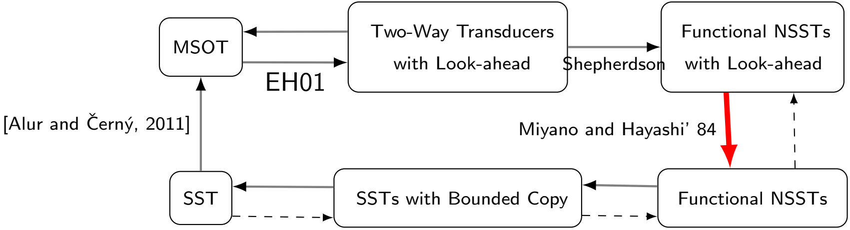
Expressiveness of Streaming String Transducers



Proof Sketch

Theorem

A transformation of infinite strings is accepted by a streaming string transducer iff it is MSO-definable.

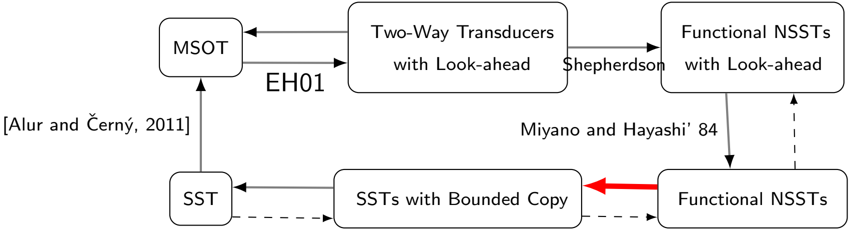


- simulate all look-aheads in parallel
- look-ahead $\sim$ universal transitions in an alternating Muller automaton
- use Miyano-Hayashi like construction to remove universality

Proof Sketch

Theorem

A transformation of infinite strings is accepted by a streaming string transducer iff it is MSO-definable.

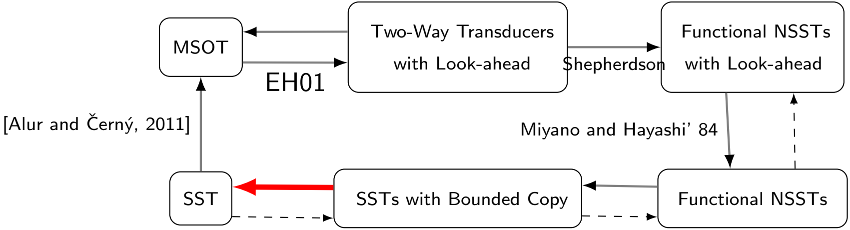


- simulate all runs in parallel
- functionality ⇒ at most $|Q|$ runs have to be simulated in parallel
- use $|Q|$ copies of each variable $x \in X$
- may introduce variable copy

Proof Sketch

Theorem

A transformation of infinite strings is accepted by a streaming string transducer iff it is MSO-definable.

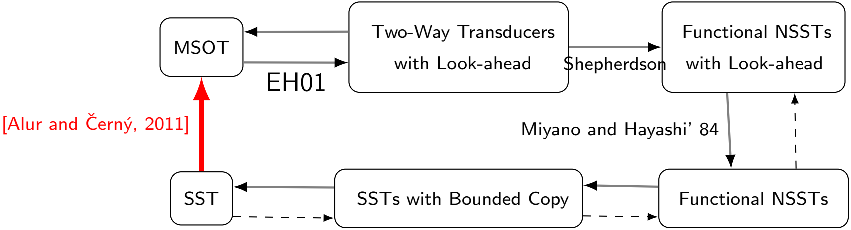


- most technical result
- based on the notion of dependency graphs
- states are sufficient abstractions of dependency graphs

Proof Sketch

Theorem

A transformation of infinite strings is accepted by a streaming string transducer iff it is MSO-definable.



- simple extension of the finite string case
- uses two domain copies for each variable

Equivalence Problem

Theorem

Equivalence problem is decidable in PSPACE for streaming-string transducers on infinite strings.

Equivalence Problem

Theorem

Equivalence problem is decidable in PSPACE for streaming-string transducers on infinite strings.

T_1 and T_2 are inequivalent iff $dom(T_1) \neq dom(T_2)$ **or**

$dom(T_1) = dom(T_2)$ and $\exists u \in dom(T_1)$, $\exists i \geq 0$ such that $T_1(u)[i] \neq T_2(u)[i]$

1. domain equivalence can be checked in PSPACE.
2. if domains are equivalent, then check existence of u
   - reduction to emptiness of reversal-bounded counter machines (NLogSpace, Ibarra)
   - product construction to simulate runs of T_1 and T_2 on the same inputs
   - guess a position i and check that there is a mismatch
   - as outputs are not produced synchronously, counters are used to retrieve the letters at position i in both outputs
   - construction ensures that finite runs can be extended to infinite accepting runs that do not modify the letters at position i

Type-Checking Problem

Theorem

Type-checking, deciding whether image of a given regular set I under a regular transformation T is contained in another given regular set O i.e. $T(I) \subseteq O$, is decidable in PSPACE for streaming-string transducers on infinite strings.

- Check whether T is defined for all strings of u, i.e. $\mathrm{dom}(T) \subseteq I$.
- A Muller automaton recognizing the domain of T can be constructed in linear time, and therefore $I \subseteq \mathrm{dom}(T)$ can be checked in PSPACE.

Type-Checking Problem

Theorem

Type-checking, deciding whether image of a given regular set I under a regular transformation T is contained in another given regular set O i.e. $T(I) \subseteq O$, is decidable in PSPACE for streaming-string transducers on infinite strings.

- Check whether T is defined for all strings of u, i.e. $\mathrm{dom}(T) \subseteq I$.
- A Muller automaton recognizing the domain of T can be constructed in linear time, and therefore $I \subseteq \mathrm{dom}(T)$ can be checked in PSPACE.
- Next we check the language $L = \{u \in \Sigma^\omega \mid u \in I, T(u) \notin O\}$ for emptiness.
- The language L can be defined by a Muller automaton A_L that simulates automaton A_I and T on the input string, and $A_{\overline{O}}$ on the output of T.

Type-Checking Problem

Theorem

Type-checking, deciding whether image of a given regular set I under a regular transformation T is contained in another given regular set O i.e. $T(I) \subseteq O$, is decidable in PSPACE for streaming-string transducers on infinite strings.

- Check whether T is defined for all strings of u, i.e. $\mathrm{dom}(T) \subseteq I$.
- A Muller automaton recognizing the domain of T can be constructed in linear time, and therefore $I \subseteq \mathrm{dom}(T)$ can be checked in PSPACE.
- Next we check the language $L = \{u \in \Sigma^\omega \mid u \in I, T(u) \notin O\}$ for emptiness.
- The language L can be defined by a Muller automaton A_L that simulates automaton A_I and T on the input string, and $A_{\overline{O}}$ on the output of T.
- This can be done by computing functions τ such that for all states q of $A_{\overline{O}}$ and all variables $x \in X$, $\tau(q, x)$ is the state of $A_{\overline{O}}$ after evaluating the current value of x, starting from state q.
- The size of A_L is exponential in A_I, A_O and T, and its emptiness can be decided in PSPACE.

Properties of Regular Transformations

- Characterized by
  - MSO,
  - two-way finite-state transducers with ω-regular look-ahead, and
  - streaming string transducers
- They are closed under sequential composition

Properties of Regular Transformations

- Characterized by
  - MSO,
  - two-way finite-state transducers with ω-regular look-ahead, and
  - streaming string transducers
- They are closed under sequential composition

Theorem

Equivalence and type-checking problems are decidable in PSPACE for streaming-string transducers on infinite strings.

Properties of Regular Transformations

- Characterized by
  - MSO,
  - two-way finite-state transducers with ω-regular look-ahead, and
  - streaming string transducers
- They are closed under sequential composition

Theorem

Equivalence and type-checking problems are decidable in PSPACE for streaming-string transducers on infinite strings.

Corollary

Equivalence of MSO-transducers on infinite strings is decidable.

Regular Transformations of Finite Strings

Regular Transformations of Infinite Strings

Conclusion

Summary

- Introduction of streaming string transducers renewed the interest in the study of regular transformations
- Streaming string transducers naturally extend from strings to more general structures, while conserving MSO equivalence.
- Streaming-string transducer models are robust: closed under bounded copy, functional nondeterminism, and regular look-ahead.
- Important verification problems like functional equivalence and pre/post condition type-checking are decidable for streaming string transducers.

Summary

- Introduction of streaming string transducers renewed the interest in the study of regular transformations
- Streaming string transducers naturally extend from strings to more general structures, while conserving MSO equivalence.
- Streaming-string transducer models are robust: closed under bounded copy, functional nondeterminism, and regular look-ahead.
- Important verification problems like functional equivalence and pre/post condition type-checking are decidable for streaming string transducers.
- A number of open problems!

Summary

- Introduction of streaming string transducers renewed the interest in the study of regular transformations
- Streaming string transducers naturally extend from strings to more general structures, while conserving MSO equivalence.
- Streaming-string transducer models are robust: closed under bounded copy, functional nondeterminism, and regular look-ahead.
- Important verification problems like functional equivalence and pre/post condition type-checking are decidable for streaming string transducers.
- A number of open problems!

Thank You!

Alur, A. and Černý, P. (2011).
Streaming transducers for algorithmic verification of single-pass list-processing programs.
In *POPL*, pages 599–610.

Alur, R. and Černý, P. (2010).
Expressiveness of streaming string transducers.
In *FSTTCS*, volume 8, pages 1–12.

Alur, R. and D'Antoni, L. (2012).
Streaming tree transducers.
In *ICALP (2)*, pages 42–53.

Alur, R., D'Antoni, L., Deshmukh, J. V., Raghothaman, M., and Yuan, Y. (2013a).
Regular functions and cost register automata.
In *LICS*.

Alur, R., Durand-Gasselin, A., and Trivedi, A. (2013b).
From monadic second-order definable string transformations to transducers.
In *LICS*.

Alur, R., Filiot, E., and Trivedi, A. (2012).

Regular transformations of infinite strings.
In *LICS*, pages 65–74.

Büchi, J. R. (1960).
Weak second-order arithmetic and finite automata.
Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, 6(1–6):66–92.

Büchi, J. R. (1962).
On a decision method in restricted second-order arithmetic.
In *Int. Congr. for Logic Methodology and Philosophy of Science*, pages 1–11. Standford University Press, Stanford.

Courcelle, B. (1994).
Monadic second-order definable graph transductions: a survey.
Theoretical Computer Science, 126(1):53–75.

Elgot, C. C. (1961).
Decision problems of finite automata design and related arithmetics.
In Transactions of the American Mathematical Society, 98(1):21–51.

Engelfriet, J. and Hoogeboom, H. J. (2001).
MSO definable string transductions and two-way finite-state transducers.
ACM Trans. Comput. Logic, 2:216–254.

McNaughton, R. (1966).
Testing and generating infinite sequences by a finite automaton.
Inform. Contr., 9:521–530.

Rabin, M. O. (1969).
Decidability of second-order theories and automata on infinite trees.
Transactions of the American Mathematical Society, 1(35).

Thomas, W. (1995).
On the synthesis of strategies in infinite games.
In *STACS*, volume 900 of *LNCS*, pages 1–13. Springer.

Trakhtenbrot, B. A. (1962).
Finite automata and monadic second order logic.
Siberian Mathematical Journal, 3:101–131.