Towards Solving String Constraints with Real-world Regular Expressions

Zhilei Han[1]

School of Software, Tsinghua University

CCF Chinasoft, 24. December, 2021

1. Based on the POPL22 paper **Solving String Constraints with Regex-Dependent Functions through Transducers with Priorities and Variables** by Taolue Chen, Alejandro Flores-Lamas, Matthew Hague, Zhilei Han, Denghang Hu, Shuanglong Kan, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu.

Motivation

- The **string** type is ubiquitous in practical programs.
- Abundant operations for manipulating strings are provided
  - `replace`, `extract`, `match` . . .
  - `split`, `join`, `indexof` . . .

Motivation

- The **string** type is ubiquitous in practical programs.
- Abundant operations for manipulating strings are provided
- Sadly, strings are vulnerable to attacks[2]

Injection

```
String query = "SELECT * FROM accounts WHERE custID='"
        + request.getParameter("id") + "'";
```

Cross-Site Scripting (XSS)

```
String page += "<input name='creditcard' type='TEXT' value='"
        + request.getParameter("CC") + "'>";
```

Insecure Deserialization

2. https://owasp.org/www-project-top-ten/2017/Top_10

Motivation

Q1: How to analyze and verify string-manipulating programs?

One powerful method is *symbolic execution*.

```
// XSS vulnerable
function instantiate(info) {
  var template =
  "<h1>User<span onMouseOver="popupText('{{bio}}')">{{userName}}</span></h1>"
  var result = template.replace("{{bio}}", info.bio);
  result = template.replace("{{userName}}", info.username);
  return result;
}
```

$\Rightarrow$ $x_1 = \mathrm{replaceAll}(\mathrm{temp}, \text{“\{\{bio\}\}”}, \mathrm{bio}) \wedge x_2 = \mathrm{replaceAll}(x_1, \text{“\{\{userName\}\}”}, \mathrm{user}) \wedge x_2 \in R$

with *attack pattern* R represented as a regular language.

Motivation

Q2: Are existing string theory enough for verifying practical programs?

`No.`

One of the major reasons is: the semantics of regular expressions in real-world programming languages are different from classical regular expressions

- greedy/lazy matching: `a*` versus `a*?`
- capturing groups and references:

```
var t = replace(s, /((ab*?)+)/g, $2);
```

- anchors:

```
s.match(/^a+(b*)c+$/);
```

Table of Content

Real-world Regular Expression

Definition 1. (Real-world Regular Expression, regex)

A real-world regular expression is defined as:

$$
\begin{aligned}
e \xlongequal{\text{def}}\ & \emptyset \mid \varepsilon \mid a \mid [e + e] \mid [e \cdot e] \mid \\
& (e) \mid && \texttt{Capturing Group} \\
& [e^*] \mid [e^{*?}] \mid && \texttt{Kleene Star} \\
& [e^+] \mid [e^{+?}] \mid && \texttt{Kleene Plus}
\end{aligned}
$$

where a *is a letter in alphabet* Σ

Note loop operator has **greedy** and **lazy** variants.

Nonstandard Semantics

The semantics of regex is nonstandard in matching (though its language is regular).

Example. (Greedy/Lazy Matching)

In Javascript, the following statement:

```
var result = “<script>foo</script>”.match(/<(.*)>/)[1]
```

returns “script>foo</script”, while

```
var result = “<script>foo</script>”.match(/<(.*?)>/)[1]
```

returns “script”.

Nonstandard Semantics

Example. (Nested Repetition)

In Javascript, some operators' behaviour depends on its context. For example, the following statement:

var result = "aaa".match(/(a*?)/)[1]

returns "ε", while

var result = "aaa".match(/(a*?)*/)[1]

returns "a".

(The ECMAScript standard[3] prohibits the match of e in e^* to be ε)

3. https://262.ecma-international.org/12.0/#sec-runtime-semantics-repeatmatcher-abstract-operation

Nonstandard Semantics

Example. (Nested Repetition)

In Javascript, some operators' behaviour depends on its context. For example, the following statement:

```
var result = "aaa".match(/(a*)*/)[1]
```

returns "aaa", while

```
var result = "aaa".match(/(a*?)*/)[1]
```

returns "a".

(The ECMAScript standard prohibits the match of e as in e^* to be ε)

It's hard to give a denotational semantics to regex!

Operational semantics?

PSST

PSST extends finite state transducer (Mealy machine) with:

- Priorities: nondeterministic transitions are ordered
- Memory: a fixed number of memory cell containing unbounded string.

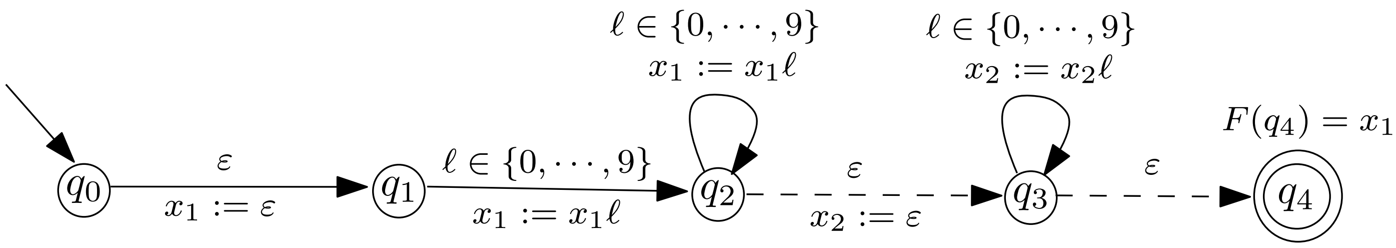

Figure 1. PSST $\mathcal{T}$ to extract the matching of the first capturing group in (\d+)(\d*)

A run of $\mathcal{T}$ on input string 2050 is:

$$q_0 \xrightarrow[\varepsilon]{x_1:=\varepsilon} q_1 \xrightarrow[2]{x_1:=x_1 2} q_2 \xrightarrow[0]{x_1:=x_1 0} q_2 \xrightarrow[5]{x_1:=x_1 5} q_2 \xrightarrow[0]{x_1:=x_1 0} q_2 \xrightarrow[\varepsilon]{x_2:=\varepsilon} q_3 \xrightarrow{\varepsilon} q_4,$$

Construction of PSST

We construct a PSST for each regex inductively as its operational semantics.

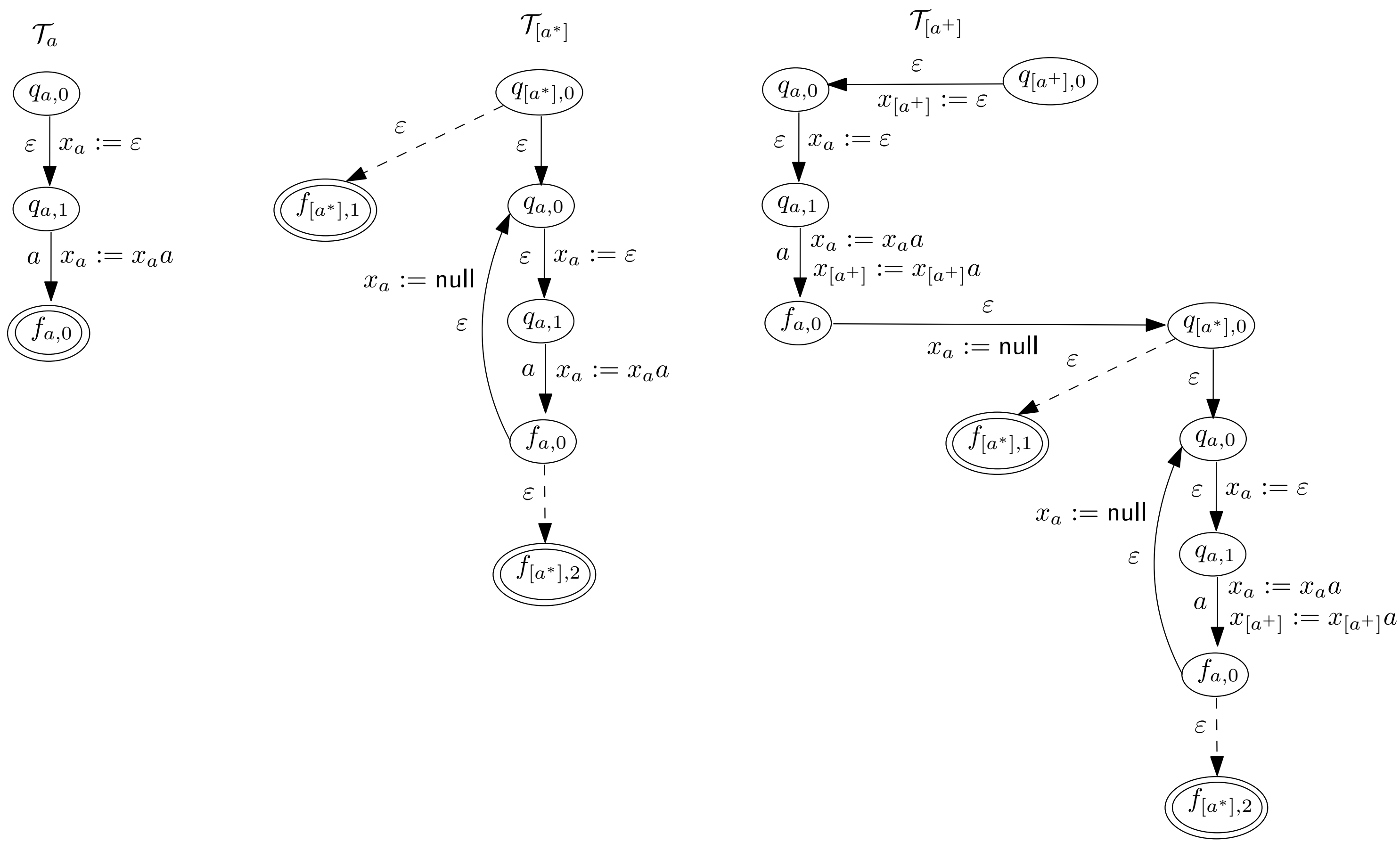

Figure 2. PSST for the RegEx a, a^* and a^+

Our String Logic

Definition 2. (STR)

The well-formed formula of the theory STR is defined as:

$$
\begin{array}{rll}
\varphi \overset{\text{def}}{=\!=} & x = y & \\
& \mid z = x \cdot y & \text{concatenation} \\
& \mid x \in e & \text{regular constraint} \\
& \mid y = \mathsf{extract}_{i,e}(x) & \text{extraction} \\
& \mid y = \mathsf{replaceAll}_{\mathrm{pat},\mathrm{rep}}(x) & \text{replacement} \\
& \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi &
\end{array}
$$

where x, y, z are string variables, $i \in \mathbb{N}$ is the index of capturing groups, e, $\mathrm{pat} \in \mathrm{regex}$ is the match pattern, and $\mathrm{rep} \in \Re$ is the replacement string. $\Re$ is defined as the concatenation of letters in Σ and references $\$i$, for $i \in \mathbb{N}$.

Semantics of regex-dependent functions

1. The $\mathsf{extract}_{i,e}(x)$ function returns matched substring of the i-th capturing group of e, if $x \in L(e)$. Otherwise, the return value is undefined.

   Example. $\mathsf{extract}_{i,e}(x)$ can be used to model many string functions like `str.match(reg)` and `reg.exec(str)` in Javascript.

   var y = “aba”.match(/(a+b)*/)[1]

   can be modeled as $y = \mathsf{extract}_{1,\Sigma^{*?}(a+b)^*\Sigma^*}(''\mathrm{aba}'')$

2. The $\mathsf{replaceAll}_{\mathrm{pat},\mathrm{rep}}(x)$ function identifies all matches of `pat` in x and replace them with string specified by `rep`. Each reference $\$i$ in `rep` will be replaced by the matching of the i-th capturing group in `pat`.

Straight-line Fragment

We introduce the straight-line fragment since the general STR is undecidable.

Definition 3. *A STR formula φ is said to be **straight-line**, if*

1. *it contains neither negation nor disjunction*
2. *φ can be ordered into a sequence of equations $x_1 = t_1, x_2 = t_2, \ldots, x_n = t_n$ plus regular constraints, such that $x_1, \ldots, x_n$ are mutually distinct, and for each $i \in \{1, \ldots, n\}$, x_i does not occur in $t_1, \ldots, t_{i-1}$.*

Let $\mathrm{STR}_{\mathrm{SL}}$ denote the set of straight-line STR formulas.

$$z = \mathsf{replaceAll}_{(a+b)^*,\$1}(x) \wedge x = y \cdot z \qquad \times$$
$$z = \mathsf{replaceAll}_{(a+b)^*,\$1}(y) \wedge x = y \cdot z \wedge y \in ab^* \qquad \checkmark$$

Decision Procedure for $\mathrm{STR}_{\mathrm{SL}}$

Theorem 1. *For each constraint* $y = \mathsf{extract}_{i,\mathrm{pat}}(x)$ *and* $y = \mathsf{replaceAll}_{\mathrm{pat},\mathrm{rep}}(x)$*. an equivalent PSST* $\mathcal{T}$ *can be constructed. (Lemma 4.7)*

Then, every $\mathrm{STR}_{\mathrm{SL}}$ formula φ can be simplified into conjunctions of formulas of the form $z = x \cdot y$, $y = \mathcal{T}(x)$ and $x \in A$, where $\mathcal{T}$ is a PSST and A is an FA.

Decision Procedure for $\mathrm{STR}_{\mathrm{SL}}$

Example. The following constraint:

$$
\begin{aligned}
x \in A_x &\ \wedge \\
y = \mathcal{T}(x) &\ \wedge\ y \in A_y \\
z = x \cdot y &\ \wedge\ z \in A_z
\end{aligned}
$$

is straight-line. We decide its satisfiability by iteratively computing **pre-images** of regular constraints under $\cdot$ (concatenation) and $\mathcal{T}$.

Step 1. For $z = x \cdot y$ and $z \in A_z$, we compute $\mathrm{Pre}(\cdot, A_z) = \cdot^{-1}(A_z) = \{(x, y) \mid x \cdot y \in A_z\}$.

It is shown in previous work[4] that $\mathrm{Pre}(\cdot, A_z)$ can be decomposed, i.e. $\mathrm{Pre}(\cdot, A_z) = A'_x \times A'_y$ for some A'_x and A'_y.

4. T. Chen, Y. Chen, M. Hague, A. W. Lin, and Z. Wu, 'What is decidable about string constraints with the ReplaceAll function', *PACMPL*, vol. 2, no. POPL, p. 3:1-3:29, 2018

Decision Procedure for $\mathrm{STR}_{\mathrm{SL}}$

Example. The remaining constraint:

$$\begin{aligned} x \in A_x \cap A'_x &\ \wedge \\ y = \mathcal{T}(x) &\ \wedge\ y \in A_y \cap A'_y \\ \cancel{z = x \cdot y} &\ \cancel{\wedge}\ \cancel{z \in A_z} \end{aligned}$$

Step 2. For $y = \mathcal{T}(x)$ and $y \in A_y \cap A'_y$, we compute $A''_x = \mathrm{Pre}(\mathcal{T}, A_y \cap A'_y) = \mathcal{T}^{-1}(A_y \cap A'_y) = \{x \mid \mathcal{T}(x) \in A_y \cap A'_y\}$.

Theorem 2. (Regularity-Preserving Property of PSST)

Given a PSST $\mathcal{T}$ and an FA A, we can compute an FA B in exponential time such that $B = \mathcal{T}^{-1}(A)$. (Lemma 5.5)

Decision Procedure for $\mathrm{STR}_{\mathrm{SL}}$

Example. The remaining constraint:

$$
\begin{aligned}
&x \in A_x \cap A'_x \cap A''_x \\
&\qquad\qquad \cancel{y = T(x)} \;\; \cancel{\wedge} \;\; \cancel{y \in A_y \cap A'_y}
\end{aligned}
$$

Step 3. We check the emptiness of $A_x \cap A'_x \cap A''_x$. If the language is empty, the constraint is unsatisfiable. Otherwise, it's satisfiable.

Our String Solver

OSTRICH: Optimistic STRIng Constraint Handler[5]

Version 1.1 now supports solving constraints with real-world regular expressions.

- The first and yet the only solver with such support
- We evaluate OSTRICH on over 195 000 string constraints.

  It greatly increase precision and efficiency compared to previous approximation-based methods.
- The implementation supports more features like anchors.

5. https://github.com/uuverifiers/ostrich

Takeaway

- String functions dependent on Real-world Regular Expressions can be modeled by PSST
- The pre-image of a PSST under regular language is computable, thus the straight-line fragment is decidable.
- Our solver can be used with software verification techniques (e.g. symbolic execution) to efficiently verify real-world string-manipulating programs.

For more formalism and proofs, check out our POPL22 paper:

> `Solving String Constraints with Regex-Dependent Functions through Transducers with Priorities and Variables`

available at https://arxiv.org/abs/2111.04298

Definition. (Prioritized Streaming String Transducers) *A* prioritized streaming string transducer *is an octuple* $\mathcal{T} = (Q, q_0, \Sigma, X, \delta, \tau, E, F)$, *where*

- Q *is a finite set of states,* $q_0 \in Q$ *is the initial state*
- Σ *is the input and output alphabet*
- X *is a finite set of string variables*
- $\delta \in Q \times \Sigma \to \bar{Q}$ *defines the non-*ε *transitions as well as their priorities (from highest to lowest)*
- $\tau \in Q \to \bar{Q} \times \bar{Q}$ *such that for every* $q \in Q$, *if* $\tau(q) = (P_1; P_2)$, *then* $P_1 \cap P_2 = \emptyset$, *(Intuitively,* $\tau(q) = (P_1; P_2)$ *specifies the* ε*-transitions at* q. ε*-transitions to the states in* P_1 *(resp.* P_2*) have higher (resp. lower) priorities than non-*ε*-transitions out of* q.*)*
- E *associates with each transition a string-variable assignment function, i.e.,* E *is partial function from* $Q \times \Sigma^{\varepsilon} \times Q$ *to* $X \to (X \cup \Sigma)^*$ *such that its domain is the set of tuples* (q, a, q') *satisfying that either* $a \in \Sigma$ *and* $q' \in \delta(q, a)$ *or* $a = \varepsilon$ *and* $q' \in \tau(q)$
- F *is the output function, which is a partial function from* Q *to* $(X \cup \Sigma)^*$

Definition. (Prioritized Streaming String Transducers) *A* prioritized streaming string transducer *is an octuple* $\mathcal{T} = (Q, q_0, \Sigma, X, \delta, \tau, E, F)$, *where*

- Q *is a finite set of states,* $q_0 \in Q$ *is the initial state*
- Σ *is the input and output alphabet*
- X *is a finite set of string variables*
- $\delta \in Q \times \Sigma \to \bar{Q}$ *defines the non-*ε *transitions as well as their priorities (from highest to lowest)*
- $\tau \in Q \to \bar{Q} \times \bar{Q}$ *such that for every* $q \in Q$, *if* $\tau(q) = (P_1; P_2)$, *then* $P_1 \cap P_2 = \emptyset$, *(Intuitively,* $\tau(q) = (P_1; P_2)$ *specifies the* ε*-transitions at* q. ε*-transitions to the states in* P_1 *(resp.* P_2*) have higher (resp. lower) priorities than non-*ε*-transitions out of* q.*)*
- E *associates with each transition a string-variable assignment function, i.e.,* E *is partial function from* $Q \times \Sigma^{\varepsilon} \times Q$ *to* $X \to (X \cup \Sigma)^*$ *such that its domain is the set of tuples* (q, a, q') *satisfying that either* $a \in \Sigma$ *and* $q' \in \delta(q, a)$ *or* $a = \varepsilon$ *and* $q' \in \tau(q)$
- F *is the output function, which is a partial function from* Q *to* $(X \cup \Sigma)^*$

Definition. (Prioritized Streaming String Transducers) *A* prioritized streaming string transducer *is an octuple* $\mathcal{T} = (Q, q_0, \Sigma, X, \delta, \tau, E, F)$, *where*

- Q *is a finite set of states,* $q_0 \in Q$ *is the initial state*
- Σ *is the input and output alphabet*
- X *is a finite set of string variables*
- $\delta \in Q \times \Sigma \to \bar{Q}$ *defines the non-*ε *transitions as well as their priorities (from highest to lowest)*
- $\tau \in Q \to \bar{Q} \times \bar{Q}$ *such that for every* $q \in Q$*, if* $\tau(q) = (P_1; P_2)$*, then* $P_1 \cap P_2 = \emptyset$*, (Intuitively,* $\tau(q) = (P_1; P_2)$ *specifies the* ε*-transitions at* q*.* ε*-transitions to the states in* P_1 *(resp.* P_2*) have higher (resp. lower) priorities than non-*ε*-transitions out of* q*.)*
- E *associates with each transition a string-variable assignment function, i.e.,* E *is partial function from* $Q \times \Sigma^{\varepsilon} \times Q$ *to* $X \to (X \cup \Sigma)^*$ *such that its domain is the set of tuples* (q, a, q') *satisfying that either* $a \in \Sigma$ *and* $q' \in \delta(q, a)$ *or* $a = \varepsilon$ *and* $q' \in \tau(q)$
- F *is the output function, which is a partial function from* Q *to* $(X \cup \Sigma)^*$

Definition. (Prioritized Streaming String Transducers) *A* prioritized streaming string transducer *is an octuple* $\mathcal{T} = (Q, q_0, \Sigma, X, \delta, \tau, E, F)$, *where*

- Q *is a finite set of states,* $q_0 \in Q$ *is the initial state*
- Σ *is the input and output alphabet*
- X *is a finite set of string variables*
- $\delta \in Q \times \Sigma \to \bar{Q}$ *defines the non-*ε *transitions as well as their priorities (from highest to lowest)*
- $\tau \in Q \to \bar{Q} \times \bar{Q}$ *such that for every* $q \in Q$, *if* $\tau(q) = (P_1; P_2)$, *then* $P_1 \cap P_2 = \emptyset$, *(Intuitively,* $\tau(q) = (P_1; P_2)$ *specifies the* ε*-transitions at* q. ε*-transitions to the states in* P_1 *(resp.* P_2*) have higher (resp. lower) priorities than non-*ε*-transitions out of* q.*)*
- E *associates with each transition a string-variable assignment function, i.e.,* E *is partial function from* $Q \times \Sigma^\varepsilon \times Q$ *to* $X \to (X \cup \Sigma)^*$ *such that its domain is the set of tuples* (q, a, q') *satisfying that either* $a \in \Sigma$ *and* $q' \in \delta(q, a)$ *or* $a = \varepsilon$ *and* $q' \in \tau(q)$
- F *is the output function, which is a partial function from* Q *to* $(X \cup \Sigma)^*$

Definition. (Prioritized Streaming String Transducers) *A* prioritized streaming string transducer *is an octuple* $\mathcal{T} = (Q, q_0, \Sigma, X, \delta, \tau, E, F)$*, where*

- Q *is a finite set of states,* $q_0 \in Q$ *is the initial state*
- Σ *is the input and output alphabet*
- X *is a finite set of string variables*
- $\delta \in Q \times \Sigma \to \bar{Q}$ *defines the non-*ε *transitions as well as their priorities (from highest to lowest)*
- $\tau \in Q \to \bar{Q} \times \bar{Q}$ *such that for every* $q \in Q$*, if* $\tau(q) = (P_1; P_2)$*, then* $P_1 \cap P_2 = \emptyset$*, (Intuitively,* $\tau(q) = (P_1; P_2)$ *specifies the* ε*-transitions at* q*.* ε*-transitions to the states in* P_1 *(resp.* P_2*) have higher (resp. lower) priorities than non-*ε*-transitions out of* q*.)*
- E *associates with each transition a string-variable assignment function, i.e.,* E *is partial function from* $Q \times \Sigma^{\varepsilon} \times Q$ *to* $X \to (X \cup \Sigma)^*$ *such that its domain is the set of tuples* (q, a, q') *satisfying that either* $a \in \Sigma$ *and* $q' \in \delta(q, a)$ *or* $a = \varepsilon$ *and* $q' \in \tau(q)$
- F *is the output function, which is a partial function from* Q *to* $(X \cup \Sigma)^*$

Definition. (Prioritized Streaming String Transducers) *A* prioritized streaming string transducer *is an octuple $\mathcal{T} = (Q, q_0, \Sigma, X, \delta, \tau, E, F)$, where*

- Q *is a finite set of states, $q_0 \in Q$ is the initial state*
- Σ *is the input and output alphabet*
- X *is a finite set of string variables*
- $\delta \in Q \times \Sigma \to \bar{Q}$ *defines the non-ε transitions as well as their priorities (from highest to lowest)*
- $\tau \in Q \to \bar{Q} \times \bar{Q}$ *such that for every $q \in Q$, if $\tau(q) = (P_1; P_2)$, then $P_1 \cap P_2 = \emptyset$, (Intuitively, $\tau(q) = (P_1; P_2)$ specifies the ε-transitions at q. ε-transitions to the states in P_1 (resp. P_2) have higher (resp. lower) priorities than non-ε-transitions out of q.)*
- E *associates with each transition a string-variable assignment function, i.e., E is partial function from $Q \times \Sigma^\varepsilon \times Q$ to $X \to (X \cup \Sigma)^*$ such that its domain is the set of tuples (q, a, q') satisfying that either $a \in \Sigma$ and $q' \in \delta(q, a)$ or $a = \varepsilon$ and $q' \in \tau(q)$*
- F *is the output function, which is a partial function from Q to $(X \cup \Sigma)^*$*

Definition. (Prioritized Streaming String Transducers) *A* prioritized streaming string transducer *is an octuple* $\mathcal{T} = (Q, q_0, \Sigma, X, \delta, \tau, E, F)$, *where*

- Q *is a finite set of states,* $q_0 \in Q$ *is the initial state*
- Σ *is the input and output alphabet*
- X *is a finite set of string variables*
- $\delta \in Q \times \Sigma \to \bar{Q}$ *defines the non-*ε *transitions as well as their priorities (from highest to lowest)*
- $\tau \in Q \to \bar{Q} \times \bar{Q}$ *such that for every* $q \in Q$, *if* $\tau(q) = (P_1; P_2)$, *then* $P_1 \cap P_2 = \emptyset$, *(Intuitively,* $\tau(q) = (P_1; P_2)$ *specifies the* ε*-transitions at* q. ε*-transitions to the states in* P_1 *(resp.* P_2*) have higher (resp. lower) priorities than non-*ε*-transitions out of* q*.)*
- E *associates with each transition a string-variable assignment function, i.e.,* E *is partial function from* $Q \times \Sigma^\varepsilon \times Q$ *to* $X \to (X \cup \Sigma)^*$ *such that its domain is the set of tuples* (q, a, q') *satisfying that either* $a \in \Sigma$ *and* $q' \in \delta(q, a)$ *or* $a = \varepsilon$ *and* $q' \in \tau(q)$
- F *is the output function, which is a partial function from* Q *to* $(X \cup \Sigma)^*$