Statically Typed String Sanitation Inside a Python (Technical Report)

Nathan Fulton **Cyrus Omar** **Jonathan Aldrich**

December 2014
CMU-ISR-14-112

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This report contains supporting evidence for claims put forth and explained in the paper "Statically Typed String Sanitation Inside a Python" [**?**], including proofs of lemmas and theorems asserted in the paper, examples, additional discussion of the paper's technical content, and errata.

This work was supported by the National Security Agency lablet contract #H98230-14-C-0140.

Keywords: type systems; regular languages; input sanitation; string sanitation

Contents

List of Figures

1 Terminology and Notation

Theorems and lemmas appearing in [**?**] are numbered correspondingly, while supporting facts appearing only in the Technical Report are lettered. Throughout this technical report, we use a small step semantics corresponding to the big step semantics given in [**?**].

2 Regular Expressions

The syntax of regular expressions over some alphabet Σ is shown in Figure **??**.

Assumption A (Regular Expression Congruences)**.** *We assume regular expressions are implicitly identified up to the following congruences:*

$$\begin{aligned} \epsilon \cdot r &\equiv r \\ r \cdot \epsilon &\equiv r \\ (r_1 \cdot r_2) \cdot r_3 &\equiv r_1 \cdot (r_2 \cdot r_3) \\ r_1 + r_2 &\equiv r_2 + r_1 \\ (r_1 + r_2) + r_3 &\equiv r_1 + (r_2 + r_3) \\ \epsilon^* &\equiv \epsilon \end{aligned}$$

Assumption B (Properties of Regular Languages)**.** *We assume the following properties:*

1. *If $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$ then $s_1 s_2 \in \mathcal{L}\{r_1 \cdot r_2\}$.*
2. *For all strings s and regular expressions r, either $s \in \mathcal{L}\{r\}$ or $s \notin \mathcal{L}\{r\}$.*
3. *Regular languages are closed under reversal.*

3 λ_{RS}

The syntax of λ_{RS} is specified in Figure **??**.

3.1 Static Semantics

The static semantics of λ_{RS} is specified in Figure **??**. The typing context obeys the standard structural properties of weakening, exchange and contraction.

3.1.1 Case Analysis

The following correctness conditions must hold for any definition of $\mathsf{lhead}(r)$ and $\mathsf{ltail}(r)$.

Condition C (Correctness of Head)**.** *If $c_1 s' \in \mathcal{L}\{r\}$, then $c_1 \in \mathcal{L}\{\mathsf{lhead}(r)\}$.*

Condition D (Correctness of Tail)**.** *If $c_1 s' \in \mathcal{L}\{r\}$ then $s' \in \mathcal{L}\{\mathsf{ltail}(r)\}$.*

For example, we conjecture (but do not here prove) that the definitions below satisfy these conditions. Note that these are slightly amended relative to the published paper.

Definition 1 (Definition of $\mathsf{lhead}(r)$)**.** We first define an auxiliary relation that determines the set of characters that the head might be, tracking the remainder of any sequences that appear:

$$\begin{aligned}
\mathsf{lhead}(\epsilon, \epsilon) &= \emptyset \\
\mathsf{lhead}(\epsilon, r') &= \mathsf{lhead}(r', \epsilon) \\
\mathsf{lhead}(a, r') &= \{a\} \\
\mathsf{lhead}(r_1 \cdot r_2, r') &= \mathsf{lhead}(r_1, r_2 \cdot r') \\
\mathsf{lhead}(r_1 + r_2, r') &= \mathsf{lhead}(r_1, r') \cup \mathsf{lhead}(r_2, r') \\
\mathsf{lhead}(r^*, r') &= \mathsf{lhead}(r, \epsilon) \cup \mathsf{lhead}(r', \epsilon)
\end{aligned}$$

We define $\mathsf{lhead}(r) = a_1 + a_2 + ... + a_i$ iff $\mathsf{lhead}(r, \epsilon) = \{a_1, a_2, ..., a_i\}$.

Definition 2 (Brzozowski's Derivative)**.** The *derivative of r with respect to s* is denoted by $\delta_s(r)$ and is $\delta_s(r) = \{t | st \in \mathcal{L}\{r\}\}$.

Definition 3 (Definition of $\mathsf{ltail}(r)$)**.** If $\mathsf{lhead}(r, \epsilon) = \{a_1, a_2, ..., a_i\}$, then we define $\mathsf{ltail}(r) = \delta_{a_1}(r) + \delta_{a_2}(r) + ... + \delta_{a_i}(r)$.

3.1.2 Replacement

The following correctness condition must hold for any definition of $\mathsf{lreplace}(r, r_1, r_2)$.

Condition E (Replacement Correctness)**.** *If $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$ then*

$$\mathsf{replace}(r; s_1; s_2) \in \mathcal{L}\{\mathsf{lreplace}(r, r_1, r_2)\}$$

We do not give a particular definition for $\mathsf{lreplace}(r, r_1, r_2)$ here.

3.2 Dynamic Semantics

Figure **??** specifies a small-step operational semantics for λ_{RS}.

3.2.1 Canonical Forms

Lemma F (Canonical Forms)**.** *If $\emptyset \vdash v : \sigma$ then:*

1. *If $\sigma = \mathsf{stringin}[r]$ then $v = \mathsf{rstr}[s]$ and $s \in \mathcal{L}\{r\}$.*
2. *If $\sigma = \sigma_1 \to \sigma_2$ then $v = \lambda x.e'$.*

Proof. By inspection of the static and dynamic semantics. □

3.2.2 Type Safety

Lemma G (Progress)**.** *If $\emptyset \vdash e : \sigma$ either $e = v$ or $e \mapsto e'$.*

Proof. The proof proceeds by rule induction on the derivation of $\emptyset \vdash e : \sigma$.

λ **fragment**. Cases SS-T-Var, SS-T-Abs, and SS-T-App are exactly as in a proof of progress for the simply typed lambda calculus.

S-T-Stringin-I. Suppose $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[s]$. Then $e = \mathsf{rstr}[s]$.

S-T-Concat. Suppose $\emptyset \vdash \mathsf{rconcat}(e_1; e_2) : \mathsf{stringin}[r_1 \cdot r_2]$ and $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$ and $\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$. By induction, $e_1 \mapsto e_1'$ or $e_1 = v_1$ and similarly, $e_2 \mapsto e_2'$ or $e_2 = v_2$. If e_1 steps, then SS-E-Concat-Left applies and so $\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)$. Similarly, if e_2 steps then e steps by SS-E-Concat-Right.

In the remaining case, $e_1 = v_1$ and $e_2 = v_2$. But then it follows by Canonical Forms that $e_1 = \mathsf{rstr}[s_1]$ and $e_2 = \mathsf{rstr}[s_2]$. Finally, by SS-E-Concat, $\mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]$.

S-T-Case. Suppose $e = \mathsf{rstrcase}(e_1; e_2; x, y.e_3)$ and $\emptyset \vdash e_1 : \mathsf{stringin}[r]$. By induction and Canonical Forms it follows that $e_1 \mapsto e_1'$ or $e_1 = \mathsf{rstr}[s]$. In the former case, e steps by S-E-Case-Left. In the latter case, note that $s = \epsilon$ or $s = at$ for some string t. If $s = \epsilon$ then e steps by S-E-Case-ϵ-Val, and if $s = at$ the e steps by S-E-Case-Concat.

S-T-Replace. Suppose $e = \mathsf{rreplace}[r](e_1; e_2)$, $\emptyset \vdash e : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]$ and:

$$\emptyset \vdash e_1 : \mathsf{stringin}[r_1] \tag{1}$$

$$\emptyset \vdash e_2 : \mathsf{stringin}[r_2] \tag{2}$$

By induction on (1), $e_1 \mapsto e_1'$ or $e_1 = v_1$ for some e_1'. If $e_1 \mapsto e_1'$ then e steps by SS-E-Replace-Left. Similarly, if e_2 steps then e steps by SS-E-Replace-Right. The only remaining case is where $e_1 = v_1$ and also $e_2 = v_2$. By Canonical Forms, $e_1 = \mathsf{rstr}[s_1]$ and $e_2 = \mathsf{rstr}[s_2]$. Therefore, $e \mapsto \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]$ by SS-E-Replace.

S-T-SafeCoerce. Suppose that $\emptyset \vdash \mathsf{rcoerce}[r](e_1) : \mathsf{stringin}[r]$. and $\emptyset \vdash e_1 : \mathsf{stringin}[r']$ for $\mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}$. By induction, $e_1 = v_1$ or $e_1 \mapsto e_1'$ for some e_1'. If $e_1 \mapsto e_1'$ then e steps by SS-E-SafeCoerce-Step. Otherwise, $e_1 = v$ and by Canonical Forms $e_1 = \mathsf{rstr}[s]$. In this case, $e = \mathsf{rcoerce}[r](\mathsf{rstr}[s]) \mapsto \mathsf{rstr}[s]$ by SS-E-SafeCoerce.

S-T-Check Suppose that $\emptyset \vdash \mathsf{rcheck}[r](e_0; x.e_1; e_2) : \mathsf{stringin}[r]$ and:

$$\emptyset \vdash e_0 : \mathsf{stringin}[r_0] \tag{3}$$

$$\emptyset, x : \mathsf{stringin}[r] \vdash e_1 : \sigma \tag{4}$$

$$\emptyset \vdash e_2 : \sigma \tag{5}$$

By induction, $e_0 \mapsto e_0'$ or $e_0 = v$. In the former case e steps by SS-E-Check-StepLeft. Otherwise, $e_0 = \mathsf{rstr}[s]$ by Canonical Forms. By Lemma **??** part 2, either $s \in \mathcal{L}\{r_0\}$ or $s \notin \mathcal{L}\{r_0\}$. In the former case e takes a step by SS-E-Check-Ok. In the latter case e takes a step by SS-E-Check-NotOk.

$\square$

Assumption H (Substitution)**.** *If* $\Psi, x : \sigma' \vdash e : \sigma$ *and* $\Psi \vdash e' : \sigma'$*, then* $\Psi \vdash [e'/x]e : \sigma$.

Lemma I (Preservation for Small Step Semantics)**.** *If* $\emptyset \vdash e : \sigma$ *and* $e \mapsto e'$ *then* $\emptyset \vdash e' : \sigma$.

Proof. By induction on the derivation of $e \mapsto e'$ and $\emptyset \vdash e : \sigma$.

λ **fragment**. Cases SS-E-AppLeft, SS-E-AppRight, and SS-E-AppAbs are exactly as in a proof of type safety for the simply typed lambda calculus.

S-E-Concat-Left. Suppose $e = \mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)$ and $e_1 \mapsto e_1'$. The only rule that applies is S-T-Concat, so $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$ and $\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$. By induction, $\emptyset \vdash e_1' : \mathsf{stringin}[r_1]$. Therefore, by S-T-Concat, $\emptyset \vdash \mathsf{rconcat}(e_1'; e_2) : \mathsf{stringin}[r_1 r_2]$.

S-E-Concat-Right. Suppose $e = \mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1; e_2')$ and $e_2 \mapsto e_2'$. The only rule that applies is S-T-Concat, so $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$ and $\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$. By induction, $\emptyset \vdash e_2' : \mathsf{stringin}[r_2]$. Therefore, by S-T-Concat, $\emptyset \vdash \mathsf{rconcat}(e_1; e_2') : \mathsf{stringin}[r_1 r_2]$.

S-E-Concat. Suppose $\mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]$. The only applicable rule is S-T-Concat, so $\emptyset \vdash \mathsf{rstr}[s_1] : \mathsf{stringin}[r_1]$ and $\emptyset \vdash \mathsf{rstr}[s_2] : \mathsf{stringin}[r_2]$ and $\emptyset \vdash \mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) : \mathsf{stringin}[r_1 \cdot r_2]$. By Canonical Forms, $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$ from which it follows by Lemma **??** that $s_1 s_2 \in \mathcal{L}\{r_1 \cdot r_2\}$. Therefore, $\emptyset \vdash \mathsf{rstr}[s_1 s_2] : \mathsf{stringin}[r_1 \cdot r_2]$ by S-T-Rstr.

S-E-Case-Left. Suppose $e \mapsto \mathsf{rstrcase}(e_1'; e_2; x, y.e_3)$ and $\emptyset \vdash e : \sigma$ and $e_1 \mapsto e_1'$. The only rule that applies is S-T-Case, so:

$$\emptyset \vdash e_1 : \mathsf{stringin}[r] \tag{6}$$
$$\emptyset \vdash e_2 : \sigma \tag{7}$$
$$\emptyset, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma \tag{8}$$

By (**??**) and the assumption that $e_1 \mapsto e_1'$, it follows by induction that $\emptyset \vdash e_1' : \mathsf{stringin}[r]$. This fact together with (**??**) and (**??**) implies by S-T-Case that $\emptyset \vdash \mathsf{rstrcase}(e_1'; e_2; x, y.e_3) : \sigma$.

S-E-Case-ϵ-Val. Suppose $\mathsf{rstrcase}(e_0; e_2; x, y.e_3) \mapsto e_2$. The only rule that applies is S-T-Case, so $\emptyset \vdash e_2 : \sigma$.

S-E-Case-Concat. Suppose that $e = \mathsf{rstrcase}(\mathsf{rstr}[as]; e_2; x, y.e_3) \mapsto [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3$ and that $\emptyset \vdash e : \sigma$. The only rule that applies is S-T-Case so:

$$\emptyset \vdash \mathsf{rstr}[as] : \mathsf{stringin}[r] \tag{9}$$
$$\emptyset \vdash e_2 : \sigma \tag{10}$$
$$\emptyset, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma \tag{11}$$

We know that $as \in \mathcal{L}\{r\}$ by Canonical Forms on (**??**) Therefore, $a \in \mathcal{L}\{\mathsf{lhead}(r)\}$ by Condition **??** and $s \in \mathcal{L}\{\mathsf{ltail}(r)\}$ by Condition **??**.

From these facts about a and s we know by S-T-Rstr that $\emptyset \vdash \mathsf{rstr}[a] : \mathsf{stringin}[\mathsf{lhead}(r)]$ and $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[\mathsf{ltail}(r)]$. It follows by Assumption **??** that $\emptyset \vdash [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 : \sigma$.

Case S-E-Replace-Left. Suppose that $e = \mathsf{rreplace}[r](e_1; e_2) \mapsto \mathsf{rreplace}[r](e_1'; e_2)$ when $e_1 \mapsto e_1'$. The only rule that applies is S-T-Replace, so $\emptyset \vdash e : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]$ where:

$$\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$$
$$\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$$

By induction, $\emptyset \vdash e_1' : \mathsf{stringin}[r_1]$. Therefore, $\emptyset \vdash \mathsf{rreplace}[r](e_1'; e_2) : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]$ by S-T-Replace.

Case S-E-Replace-Right. Suppose that $e = \mathsf{rreplace}[r](e_1; e_2) \mapsto \mathsf{rreplace}[r](e_1'; e_2)$ when $e_1 \mapsto e_1'$. The only rule that applies is S-T-Replace, so $\emptyset \vdash e : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]$ where:

$$\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$$
$$\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$$

By induction, $\emptyset \vdash e_1' : \mathsf{stringin}[r_1]$. Therefore, $\emptyset \vdash \mathsf{rreplace}[r](r_1'; r_2) : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]$ by S-T-Replace.

Case S-E-Replace.

Suppose $e = \mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]$. The only applicable rule is S-T-Replace, so

$$\emptyset \vdash \mathsf{rstr}[s_1] : \mathsf{stringin}[r_1]$$
$$\emptyset \vdash \mathsf{rstr}[s_2] : \mathsf{stringin}[r_2]$$

By conanical forms, $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$. Therefore,

$$\mathsf{replace}(r; s_1; s_2) \in \mathcal{L}\{\mathsf{lreplace}(r, r_1, r_2)\}$$

by Condition **??**. It is finally derivable by S-T-Rstr that:

$\emptyset \vdash \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)] : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]$.

Case S-E-SafeCoerce. Suppose that $\mathsf{rcoerce}[r](\mathsf{rstr}[s_1]) \mapsto \mathsf{rstr}[s_1]$. The only applicable rule is S-T-SafeCoerce, so $\emptyset \vdash \mathsf{rcoerce}[r](s_1) : \mathsf{stringin}[r]$ and $\emptyset \vdash \mathsf{rstr}[s_1] : \mathsf{stringin}[r']$ and $\mathcal{L}\{r'\} \subset \mathcal{L}\{r\}$. By Canonical Forms, $s' \in \mathcal{L}\{r'\}$. By the definition of subset, $s' \in \mathcal{L}\{r\}$. Therefore, by S-T-Rstr, we have that $\emptyset \vdash \mathsf{rstr}[s'] : \mathsf{stringin}[r]$.

Case S-E-Check-Ok. Suppose $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto [\mathsf{rstr}[s]/x]e_1$ and $s \in \mathcal{L}\{r\}$, and $\emptyset \vdash \mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) : \sigma$. The only rule that applies is S-T-Check, so $\emptyset, x : \mathsf{stringin}[r] \vdash e_1 : \sigma$. By S-T-Rstr, we have that $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$. By Substitution, we have that $\emptyset \vdash [\mathsf{rstr}[s]/x]e_1 : \sigma$.

Case S-E-Check-NotOk. Suppose $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto e_2$ and $s \notin \mathcal{L}\{r\}$ and $\emptyset \vdash \mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) : \sigma$. The only applicable rule is S-T-Check, so $\emptyset \vdash e_2 : \sigma$.

□

Theorem J (Type Safety for small step semantics.)**.** *If $\emptyset \vdash e : \sigma$ then either e val or $e \mapsto^* e'$ and $\emptyset \vdash e' : \sigma$.*

Proof. Follows from applying progress and preservation transitively over the multistep judgement. □

3.2.3 The Security Theorem

Theorem 4 (Correctness of Input Sanitation for λ_{RS})**.** *If $\emptyset \vdash e : \mathsf{stringin}[r]$ and $e \mapsto^* \mathsf{rstr}[s]$ then $s \in \mathcal{L}\{r\}$.*

Proof. By type safety, $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$. By canonical forms, $s \in \mathcal{L}\{r\}$. □

4 λ_P

We will define a translation to a language with only standard strings and regular expressions. The syntax of λ_P is shown in Figure **??**.

4.1 Static Semantics

The static semantics of λ_P is shown in Figure **??**. The typing context of λ_P obeys the standard structural properties of weakening, exchange and contraction.

4.2 Dynamic Semantics

The dynamic semantics of λ_P is shown in Figure **??**.

4.2.1 Canonical Forms

Lemma 5 (Canonical Forms)**.** *If* $\emptyset \vdash \dot{v} : \tau$ *then:*

- *If* $\tau = \tau_1 \rightarrow \tau_2$ *then* $\dot{v} = \lambda x : \tau.\iota$.
- *If* $\tau = \mathsf{regex}$ *then* $\dot{v} = \mathsf{rx}[r]$.
- *If* $\tau = \mathsf{string}$ *then* $\dot{v} = \mathsf{str}[s]$.

Proof. By inspection of the static and dynamic semantics. □

4.2.2 Type Safety

Theorem 6 (Progress)**.** *If* $\emptyset \vdash \iota : \tau$ *either* $\iota = \dot{v}$ *or* $\iota \mapsto \iota'$.

Proof. The proof proceeds by induction on the typing assumption.

λ **fragment**. Cases P-T-Var, P-T-Abs, and P-T-App are exactly as in a proof of progress for the simply typed lambda calculus.

P-T-String. In this case, $\iota = \mathsf{str}[s]$, which is a value.

P-T-Regex. In this case, $\iota = \mathsf{rx}[r]$, which is a value.

P-T-Concat. In this case, we have that $\emptyset \vdash \mathsf{pconcat}(\iota_1; \iota_2) : \mathsf{string}$ and $\emptyset \vdash \iota_1 : \mathsf{string}$ and $\emptyset \vdash \iota_2 : \mathsf{string}$. By the IH, we have that either $\iota_1 \rightsquigarrow \iota_1'$ or $\iota_1 = \dot{v}_1$, and similarly $\iota_2 \rightsquigarrow \iota_2'$ or $\iota_2 = \dot{v}_2$. If ι_1 steps, then we can make progress via PS-E-ConcatLeft. If ι_2 steps, then we can make progress via PS-E-ConcatRight. If both are values, then by canonical forms $\iota_1 = \mathsf{str}[s_1]$ and $\iota_2 = \mathsf{str}[s_2]$ so we can make progress by PS-E-Concat.

P-T-Case. Suppose $\emptyset \vdash \mathsf{pstrcase}(\iota_1; \iota_2; x, y.\iota_3) : \tau$ and $\emptyset \vdash \iota_1 : \mathsf{string}$. By induction and canonical forms, either $\iota_1 \mapsto \iota_1'$ or $\iota_1 = \mathsf{str}[s_1]$. If ι_1 steps then we can make progress by PS-E-CaseLeft. If it is a value, then by the definition of strings, either $s_1 = \epsilon$ or $s_1 = as$ for some string s. If s_1 is empty, then we can make progress by PS-E-Case-Epsilon. Otherwise, we can make progress by PS-E-Case-Cons.

P-T-Replace. Suppose $\emptyset \vdash \mathsf{preplace}(\iota_1; \iota_2; \iota_3) : \mathsf{string}$ and $\emptyset \vdash \iota_1 : \mathsf{regex}$ and $\emptyset \vdash \iota_2 : \mathsf{string}$ and $\emptyset \vdash \iota_3 : \mathsf{string}$. By induction and canonical forms, either $\iota_1 \mapsto \iota_1'$ or $\iota_1 = \mathsf{rx}[r]$. Similarly, $\iota_2 \mapsto \iota_2'$ or $\iota_2 = \mathsf{str}[s_2]$, and $\iota_3 \mapsto \iota_3'$ or $\iota_3 = \mathsf{str}[s_3]$. If ι_1 steps, then we can make progress by PS-E-ReplaceLeft. If ι_2 steps then we can make progress by PS-E-ReplaceMid. If ι_3 steps, then we can make progress by PS-E-ReplaceRight. If all three are values, we can make progress by PS-E-Replace.

P-T-Check. Suppose $\emptyset \vdash \mathsf{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4)$ and $\emptyset \vdash \iota_1 : \mathsf{regex}$ and $\emptyset \vdash \iota_2 : \mathsf{string}$. By induction and canonical forms, either $\iota_1 \mapsto \iota_1'$ or $\iota_1 = \mathsf{rx}[r]$. Similarly, $\iota_2 \mapsto \iota_2'$ or $\iota_2 = \mathsf{str}[s]$. If ι_1 steps, then we can make progress by PS-E-CheckLeft. If ι_2 steps, then we can make progress by PS-E-CheckRight. If both are values, then by Assumption **??**.2, either $s \in \mathcal{L}\{r\}$ or $s \notin \mathcal{L}\{r\}$. In the former case, we can make progress by PS-E-Check-OK. In the latter case, we can make progress by PS-E-Check-NotOK.

□

Assumption K (Substitution)**.** *If* $\Theta, x : \tau' \vdash \iota : \tau$ *and* $\Theta \vdash \iota' : \tau'$ *then* $\Theta \vdash [\iota'/x]\iota : \tau$.

Theorem 7 (Preservation)**.** *If* $\emptyset \vdash \iota : \tau$ *and* $\iota \mapsto \iota'$ *then* $\emptyset \vdash \iota' : \tau$.

Proof. The proof proceeds by rule induction on $\iota \mapsto \iota'$ and $\emptyset \vdash \iota : \tau$.

λ **fragment**. Cases PS-E-AppLeft, PS-E-AppRight, and PS-E-AppAbs are exactly as in a proof of type safety for the simply typed lambda calculus.

Case PS-E-ConcatLeft. Suppose $\mathsf{pconcat}(\iota_1; \iota_2) \mapsto \mathsf{pconcat}(\iota_1'; \iota_2)$ and $\iota_1 \mapsto \iota_1'$. The only applicable typing rule is P-T-Concat, so $\emptyset \vdash \iota_1 : \mathsf{string}$ and $\emptyset \vdash \iota_2 : \mathsf{string}$. By induction, $\emptyset \vdash \iota_1' : \mathsf{string}$, so $\emptyset \vdash \mathsf{rconcat}(\iota_1'; \iota_2) : \mathsf{string}$ by P-T-Concat.

Case PS-E-ConcatRight. Suppose $\mathsf{pconcat}(\mathsf{str}[s_1]; \iota_2) \mapsto \mathsf{pconcat}(\mathsf{str}[s_1]; \iota_2')$ and $\iota_2 \mapsto \iota_2'$. The only applicable typing rule is P-T-Concat, so $\emptyset \vdash \mathsf{str}[s_1] : \mathsf{string}$ and $\emptyset \vdash \iota_2 : \mathsf{string}$. By induction, $\emptyset \vdash \iota_2' : \mathsf{string}$, so $\emptyset \vdash \mathsf{rconcat}(\mathsf{str}[s_1]; \iota_2') : \mathsf{string}$ by P-T-Concat.

Case PS-E-Concat. Suppose $\mathsf{pconcat}(\mathsf{str}[s_1]; \mathsf{str}[s_2]) \mapsto \mathsf{str}[s_1 s_2]$. By P-T-String, $\emptyset \vdash \mathsf{str}[s_1 s_2] : \mathsf{string}$.

Case PS-E-CaseLeft. Suppose $\mathsf{pstrcase}(\iota_1; \iota_2; x, y.\iota_3) \mapsto \mathsf{rstrcase}(\iota_1'; \iota_2; x, y.\iota_3)$ and $\iota_1 \mapsto \iota_1'$. The only rule that applies is P-T-Case, so:

$$\begin{aligned} \emptyset &\vdash \iota_1 : \mathsf{string} \\ \emptyset &\vdash \iota_2 : \tau \\ \emptyset, x : \mathsf{string}, y : \mathsf{string} &\vdash \iota_3 : \tau \end{aligned}$$

By induction, $\emptyset \vdash \iota_1' : \mathsf{string}$. By P-T-Case, $\emptyset \vdash \mathsf{pstrcase}(\iota_1'; \iota_2; x, y.\iota_3) : \tau$.

Case PS-E-CaseEpsilon. Suppose $\mathsf{pstrcase}(\mathsf{str}[\epsilon]; \iota_2; x, y.\iota_3) \mapsto \iota_2$. The only rule that applies is P-T-Case, so $\emptyset \vdash \iota_2 : \tau$.

Case PS-E-Case-Cons. Suppose $\mathsf{pstrcase}(\mathsf{str}[as]; \iota_2; x, y.\iota_3) \mapsto [\mathsf{str}[a], \mathsf{str}[s]/x, y]\iota_3$ The only rule that applies is P-T-Case, so:

$$\begin{aligned}\emptyset &\vdash \iota_1 : \mathsf{string}\\ \emptyset &\vdash \iota_2 : \tau\\ \emptyset, x : \mathsf{string}, y : \mathsf{string} &\vdash \iota_3 : \tau\end{aligned}$$

By P-T-String, we have that $\emptyset \vdash \mathsf{str}[a] : \mathsf{string}$ and $\emptyset \vdash \mathsf{str}[s] : \mathsf{string}$. By weakening and Substitution applied twice, we have that $\emptyset \vdash [\mathsf{str}[a], \mathsf{str}[s]/x, y]\iota_3 : \tau$.

Case PS-E-ReplaceLeft. Suppose $\mathsf{preplace}(\iota_1; \iota_2; \iota_3) \mapsto \mathsf{preplace}(\iota_1'; \iota_2; \iota_3)$ and $\iota_1 \mapsto \iota_1'$. The only rule that applies is P-T-Replace, so $\tau = \mathsf{string}$ and:

$$\begin{aligned}\emptyset &\vdash \iota_1 : \mathsf{regex}\\ \emptyset &\vdash \iota_2 : \mathsf{string}\\ \emptyset &\vdash \iota_3 : \mathsf{string}\end{aligned}$$

By induction, $\emptyset \vdash \iota_1' : \mathsf{regex}$. Therefore, by P-T-Replace $\emptyset \vdash \mathsf{preplace}(\iota_1'; \iota_2; \iota_3)$.

Case PS-E-ReplaceMid. Suppose $\mathsf{preplace}(\mathsf{rx}[r]; \iota_2; \iota_3) \mapsto \mathsf{preplace}(\mathsf{rx}[r]; \iota_2'; \iota_3)$ and $\iota_2 \mapsto \iota_2'$. The only rule that applies is P-T-Replace, so $\tau = \mathsf{string}$ and:

$$\begin{aligned}\emptyset &\vdash \mathsf{rx}[r] : \mathsf{regex}\\ \emptyset &\vdash \iota_2 : \mathsf{string}\\ \emptyset &\vdash \iota_3 : \mathsf{string}\end{aligned}$$

By induction, $\emptyset \vdash \iota_2' : \mathsf{string}$. Therefore, by P-T-Replace $\emptyset \vdash \mathsf{preplace}(\mathsf{rx}[r]; \iota_2'; \iota_3)$.

Case PS-E-ReplaceRight. Suppose $\mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s]; \iota_3) \mapsto \mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s]; \iota_3')$ and $\iota_3 \mapsto \iota_3'$. The only rule that applies is P-T-Replace, so $\tau = \mathsf{string}$ and:

$$\begin{aligned}\emptyset &\vdash \mathsf{rx}[r] : \mathsf{regex}\\ \emptyset &\vdash \mathsf{str}[s] : \mathsf{string}\\ \emptyset &\vdash \iota_3 : \mathsf{string}\end{aligned}$$

By induction, $\emptyset \vdash \iota_3' : \mathsf{string}$. Therefore, by P-T-Replace $\emptyset \vdash \mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s]; \iota_3')$.

Case PS-E-Replace. Suppose $\mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s_2]; \mathsf{str}[s_3]) \mapsto \mathsf{str}[\mathsf{replace}(r; s_2; s_3)]$. The only applicalbe rule is P-T-Replace, so $\tau = \mathsf{string}$. By P-T-String, $\emptyset \vdash \mathsf{str}[\mathsf{replace}(r; s_2; s_3)] : \mathsf{string}$.

Case PS-E-CheckLeft. Suppose $\mathsf{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4) \mapsto \mathsf{pcheck}(\iota_1'; \iota_2; \iota_3; \iota_4)$ and $\iota_1 \mapsto \iota_1'$. The only applicable typing rule is P-T-Check, so:

$$\begin{aligned}\emptyset &\vdash \iota_1 : \mathsf{regex}\\ \emptyset &\vdash \iota_2 : \mathsf{string}\\ \emptyset &\vdash \iota_3 : \tau\\ \emptyset &\vdash \iota_4 : \tau\end{aligned}$$

By induction, $\emptyset \vdash \iota_1' : \mathsf{regex}$. Therefore, by P-T-Check $\emptyset \vdash \mathsf{pcheck}(\iota_1'; \iota_2; \iota_3; \iota_4) : \tau$.

Case PS-E-CheckRight. Suppose $\mathsf{pcheck}(\mathsf{rx}[r]; \iota_2; \iota_3; \iota_4) \mapsto \mathsf{pcheck}(\mathsf{rx}[r]; \iota_2'; \iota_3; \iota_4)$ and $\iota_2 \mapsto \iota_2'$. The only applicable typing rule is P-T-Check, so:

$$\begin{aligned}
&\emptyset \vdash \mathsf{rx}[r] : \mathsf{regex} \\
&\emptyset \vdash \iota_2 : \mathsf{string} \\
&\emptyset \vdash \iota_3 : \tau \\
&\emptyset \vdash \iota_4 : \tau
\end{aligned}$$

By induction, $\emptyset \vdash \iota_2' : \mathsf{string}$. Therefore, by P-T-Check $\emptyset \vdash \mathsf{pcheck}(\mathsf{rx}[r]; \iota_2'; \iota_3; \iota_4) : \tau$.

Case PS-E-Check-Ok. Suppose $\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; \iota_3; \iota_4) \mapsto \iota_3$. The only applicable typing rule is P-T-Check, so $\emptyset \vdash \iota_3 : \tau$.

Case PS-E-Check-Ok. Suppose $\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; \iota_3; \iota_4) \mapsto \iota_4$. The only applicable typing rule is P-T-Check, so $\emptyset \vdash \iota_4 : \tau$.

□

5 Translation from λ_{RS} to λ_P

The translation from λ_{RS} to λ_P is defined in Figure **??**.

Theorem 8 (Type-Preserving Translation)**.** *If* $\Psi \vdash e : \sigma$ *then* $[\![\Psi]\!] \vdash [\![\iota]\!] : [\![\sigma]\!]$

Proof. By induction on the typing relation.

Case S-T-Var. Suppose $\Psi \vdash x : \sigma$ and $x : \sigma \in \Psi$. We have by definition that $x : [\![\sigma]\!] \in [\![\Psi]\!]$ and $[\![x]\!] = x$. By P-T-Var, we have that $[\![\Psi]\!] \vdash x : [\![\sigma]\!]$.

Case S-T-Abs. Suppose $\Psi \vdash \lambda x : \sigma_1.e' : \sigma_1 \to \sigma_2$ and $\Psi, x : \sigma_1 \vdash e' : \sigma_2$. We have by definition:

$$\begin{aligned}
[\![\lambda x : \sigma_1.e']\!] &= \lambda x : [\![\sigma_1]\!].[\![e']\!] \\
[\![\sigma_1 \to \sigma_2]\!] &= [\![\sigma_1]\!] \to [\![\sigma_2]\!] \\
[\![\Psi, x : \sigma_1]\!] &= [\![\Psi]\!], x : [\![\sigma_1]\!]
\end{aligned}$$

By induction, we have that $[\![\Psi]\!], x : [\![\sigma_1]\!] \vdash [\![e']\!] : [\![\sigma_2]\!]$.

By P-T-Abs, we have that $[\![\Psi]\!] \vdash \lambda x : [\![\sigma_1]\!].[\![e']\!] : [\![\sigma_1]\!] \to [\![\sigma_2]\!]$.

Case S-T-App. Suppose $\Psi \vdash e_1(e_2) : \sigma$ and $\Psi \vdash e_1 : \sigma_2 \to \sigma$ and $\Psi \vdash e_2 : \sigma_2$. We have by definition:

$$\begin{aligned}
[\![e_1(e_2)]\!] &= [\![e_1]\!]([\![e_2]\!]) \\
[\![\sigma_2 \to \sigma]\!] &= [\![\sigma_2]\!] \to [\![\sigma]\!]
\end{aligned}$$

By induction, $[\![\Psi]\!] \vdash [\![e_1]\!] : [\![\sigma_2]\!] \to [\![\sigma]\!]$ and $[\![\Psi]\!] \vdash [\![e_2]\!] : [\![\sigma_2]\!]$. Therefore, $[\![\Psi]\!] \vdash [\![e_1]\!]([\![e_2]\!]) : [\![\sigma]\!]$ by P-T-App.

Case S-T-StringIn-I. Suppose $\Psi \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$. By definition, $[\![\mathsf{rstr}[s]]\!] = \mathsf{str}[s]$ and $[\![\mathsf{stringin}[r]]\!] = \mathsf{string}$. By P-T-String, $\Theta \vdash \mathsf{str}[s] : \mathsf{string}$.

Case S-T-Concat. Suppose $\Psi \vdash \mathsf{rconcat}(e_1; e_2) : \mathsf{stringin}[r_1 \cdot r_2]$ and $\Psi \vdash e_1 : \mathsf{stringin}[r_1]$ and $\Psi \vdash e_2 : \mathsf{stringin}[r_2]$. We have by definition:

$$\begin{aligned}
[\![\mathsf{rconcat}(e_1; e_2)]\!] &= \mathsf{pconcat}([\![e_1]\!]; [\![e_2]\!]) \\
[\![\mathsf{stringin}[r_1]]\!] &= \mathsf{string} \\
[\![\mathsf{stringin}[r_2]]\!] &= \mathsf{string} \\
[\![\mathsf{stringin}[r_1 \cdot r_2]]\!] &= \mathsf{string}
\end{aligned}$$

By induction, $[\![\Psi]\!] \vdash [\![e_1]\!] : \mathsf{string}$ and $[\![\Psi]\!] \vdash [\![e_2]\!] : \mathsf{string}$. Thus, $[\![\Psi]\!] \vdash \mathsf{pconcat}([\![e_1]\!]; [\![e_2]\!]) : \mathsf{string}$ by P-T-Concat.

Case S-T-Case. Suppose $\Psi \vdash \mathsf{rstrcase}(e_1; e_2; x, y.e_3) : \sigma$ and $\Psi \vdash e_1 : \mathsf{stringin}[r]$ and $\Psi \vdash e_2 : \sigma$ and $\Psi, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma$. We have by definition:

$$\begin{aligned}
[\![\mathsf{rstrcase}(e_1; e_2; x, y.e_3)]\!] &= \mathsf{pstrcase}([\![e_1]\!]; [\![e_2]\!]; x, y.[\![e_3]\!]) \\
[\![\mathsf{stringin}[r]]\!] &= \mathsf{string} \\
[\![\mathsf{stringin}[\mathsf{lhead}(r)]]\!] &= \mathsf{string} \\
[\![\mathsf{stringin}[\mathsf{ltail}(r)]]\!] &= \mathsf{string} \\
[\![\Psi, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)]]\!] &= [\![\Psi]\!], x : \mathsf{string}, y : \mathsf{string}
\end{aligned}$$

By induction, $[\![\Psi]\!] \vdash [\![e_1]\!] : \mathsf{string}$ and $[\![\Psi]\!] \vdash [\![e_2]\!] : [\![\sigma]\!]$, and $[\![\Psi]\!], x : \mathsf{string}, y : \mathsf{string} \vdash [\![e_3]\!] : [\![\sigma]\!]$. By P-T-Case, we have that $[\![\Psi]\!] \vdash \mathsf{pstrcase}([\![e_1]\!]; [\![e_2]\!]; x, y.[\![e_3]\!]) : [\![\sigma]\!]$.

Case S-T-Replace. Suppose $\Psi \vdash \mathsf{rreplace}[r](e_1; e_2) : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]$ and $\Psi \vdash e_1 : \mathsf{stringin}[r_1]$ and $\Psi \vdash e_2 : \mathsf{stringin}[r_2]$. We have by definition:

$$\begin{aligned}
\llbracket \mathsf{rreplace}[r](e_1; e_2) \rrbracket &= \mathsf{preplace}(\mathsf{rx}[r]; [\![e_1]\!]; [\![e_2]\!]) \\
[\![\mathsf{stringin}[r_1]]\!] &= \mathsf{string} \\
[\![\mathsf{stringin}[r_2]]\!] &= \mathsf{string} \\
[\![\mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]]\!] &= \mathsf{string}
\end{aligned}$$

By induction, we have that $[\![\Psi]\!] \vdash [\![e_1]\!] : \mathsf{string}$ and $[\![\Psi]\!] \vdash [\![e_2]\!] : \mathsf{string}$. By P-T-Regex, we have that $[\![\Psi]\!] \vdash \mathsf{rx}[r] : \mathsf{regex}$. By P-T-Replace, we have that $[\![\Psi]\!] \vdash \mathsf{preplace}(\mathsf{rx}[r]; [\![e_1]\!]; [\![e_2]\!]) : \mathsf{string}$.

Case S-T-SafeCoerce. Suppose $\Psi \vdash \mathsf{rcoerce}[r](e) : \mathsf{stringin}[r]$ and $\Psi \vdash e : \mathsf{stringin}[r']$. By definition, $[\![\mathsf{rcoerce}[r](e)]\!] = [\![e]\!]$. By induction, $[\![\Psi]\!] \vdash [\![e]\!] : [\![\mathsf{stringin}[r']]\!]$.

Case S-T-Check. Suppose $\Psi \vdash \mathsf{rcheck}[r](e_0; x.e_1; e_2) : \sigma$ where $\Psi \vdash e_0 : \mathsf{stringin}[r']$ and $\Psi, x : \mathsf{stringin}[r] \vdash e_1 : \sigma$ and $\Psi \vdash e_2 : \sigma$. We have by definition:

$$\begin{aligned}
\llbracket \mathsf{rcheck}[r](e_0; x.e_1; e_2) \rrbracket &= \mathsf{pcheck}(\mathsf{rx}[r]; [\![e_0]\!]; (\lambda x : \mathsf{string}.[\![e_1]\!])[\![e_0]\!]; [\![e_2]\!]) \\
[\![\mathsf{stringin}[r']]\!] &= \mathsf{string} \\
[\![\mathsf{stringin}[r]]\!] &= \mathsf{string} \\
[\![\Psi, x : \mathsf{stringin}[r]]\!] &= [\![\Psi]\!], x : \mathsf{string}
\end{aligned}$$

By induction, we have that $[\![\Psi]\!] \vdash [\![e_0]\!] : \mathsf{string}$ and $[\![\Psi]\!], x : \mathsf{string} \vdash [\![e_1]\!] : [\![\sigma]\!]$ and $[\![\Psi]\!] \vdash [\![e_2]\!] : [\![\sigma]\!]$.
By P-T-Regex, we have that $[\![\Psi]\!] \vdash \mathsf{rx}[r] : \mathsf{regex}$.
By P-T-Abs and P-T-App, we have that $[\![\Psi]\!] \vdash (\lambda x : \mathsf{string}.[\![e_1]\!])([\![e_0]\!]) : [\![\sigma]\!]$.
By P-T-Check, we have that $[\![\Psi]\!] \vdash \mathsf{pcheck}(\mathsf{rx}[r]; [\![e_0]\!]; (\lambda x : \mathsf{string}.[\![e_1]\!])([\![e_0]\!]); [\![e_2]\!]) : [\![\sigma]\!]$.

□

Assumption L (Multistep Closure). *The following closure properties hold:*

1. *If* $\iota_1 \mapsto^* \iota_1'$ *then* $\iota_1(\iota_2) \mapsto^* \iota_1'(\iota_2)$.
2. *If* $\iota_2 \mapsto^* \iota_2'$ *then* $\dot{v}_1(\iota_2) \mapsto^* \dot{v}_1(\iota_2')$.
3. $(\lambda x.\dot{v})(\dot{v}_1) = [\dot{v}_1/x]\dot{v}$.
4. $\mathsf{pconcat}(\mathsf{str}[s_1]; \mathsf{str}[s_2]) \mapsto^* \mathsf{str}[s_1 s_2]$
5. *If* $\iota_1 \mapsto^* \iota_1'$ *then* $\mathsf{pstrcase}(\iota_1; \iota_2; \lambda x.\lambda y.\iota_3) \mapsto^* \mathsf{pstrcase}(\iota_1'; \iota_2; \lambda x.\lambda y.\iota_3)$
6. $\mathsf{pstrcase}(\mathsf{str}[\epsilon]; \dot{v}_2; \lambda x.\lambda y.\dot{v}_3) \mapsto^* \dot{v}_2$.
7. $\mathsf{pstrcase}(\mathsf{str}[as]; \iota_2; \lambda x.\lambda y.\iota_3) \mapsto^* [a, s/x, y]\iota_3$.
8. *If* $\iota_1 \mapsto \iota_1'$ *then* $\mathsf{preplace}(\mathsf{rx}[r]; \iota_1; \iota_2) \mapsto^* \mathsf{preplace}(\mathsf{rx}[r]; \iota_1'; \iota_2)$.
9. *If* $\iota_2 \mapsto \iota_2'$ *then* $\mathsf{preplace}(\mathsf{rx}[r]; \dot{v}; \iota_2) \mapsto^* \mathsf{preplace}(\mathsf{rx}[r]; \dot{v}; \iota_2')$.
10. $\mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s_1]; \mathsf{str}[s_2]) \mapsto \mathsf{str}[\mathsf{lreplace}(r, s_1, s_2)]$.
11. *If* $\iota \mapsto^* \iota'$ *then* $\mathsf{pcheck}(\mathsf{rx}[r]; \iota; (\lambda x.[\![\iota_1]\!])(\iota); [\![\iota_2]\!]) \mapsto^* \mathsf{pcheck}(\mathsf{rx}[r]; \iota'; (\lambda x.[\![\iota_1]\!])(\iota'); [\![\iota_2]\!])$
12. *If* $s \in \mathcal{L}\{r\}$ *then* $\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; (\lambda x.\iota_1)(\mathsf{str}[s]); \iota_2) \mapsto^* [\mathsf{str}[s]/x]\iota_1$.
13. *If* $s \notin \mathcal{L}\{r\}$ *then* $\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; (\lambda x.\iota_1)(\mathsf{str}[s]); \iota_2) \mapsto^* \iota_2$.

Theorem 9 (Translation Correctness). *If* $\emptyset \vdash e : \sigma$ *and* $e \mapsto e'$ *then* $[\![e]\!] \mapsto^* [\![e']\!]$.

Proof. By induction on evaluation and typing.

Case SS-E-AppLeft. Suppose $e_1(e_2) \mapsto e_1'(e_2)$ and $e_1 \mapsto e_1'$. We have by definition that

$$[\![e_1(e_2)]\!] = [\![e_1]\!]([\![e_2]\!])$$
$$[\![e_1'(e_2)]\!] = [\![e_1']\!]([\![e_2]\!])$$

The only typing rule that applies is S-T–App, so $\emptyset \vdash e_1 : \sigma_2 \to \sigma$.
Inductively, we have that $[\![e_1]\!] \mapsto^* [\![e_1']\!]$.
By Assumption **??**.1, we have that $[\![e_1]\!]([\![e_2]\!]) \mapsto^* [\![e_1']\!]([\![e_2]\!])$.

Case SS-E-AppRight. Suppose $v_1(e_2) \mapsto v_1(e_2')$ and $e_2 \mapsto e_2'$. We know by definition that

$$\llbracket v_1(e_2) \rrbracket = \llbracket v_1 \rrbracket (\llbracket) \rightsquigarrow e_2 \rrbracket$$
$$\llbracket v_1(e_2') \rrbracket = \llbracket v_1 \rrbracket (\llbracket e_2' \rrbracket)$$

The only typing rule that applies is S-T-App, so $\emptyset \vdash v_1 : \sigma_2 \to \sigma$.
Inductively, we have that $\llbracket e_2 \rrbracket \mapsto^* \llbracket e_2' \rrbracket$.
By assumption **??**.2, we have that $\llbracket v_1 \rrbracket (\llbracket e_2 \rrbracket) \mapsto^* \llbracket v_1 \rrbracket (\llbracket e_2' \rrbracket)$.

Case SS-E-AppAbs. Suppose $(\lambda x.e)(v_2) \mapsto [v_2/x]e$. We know by definition that

$$\llbracket (\lambda.e)(v_2) \rrbracket = \llbracket \lambda x.e \rrbracket \llbracket v_2 \rrbracket \qquad = (\lambda x.\llbracket e \rrbracket)(\llbracket v_2 \rrbracket)$$

By Assumption **??**, $(\lambda x.\llbracket e \rrbracket)(\llbracket v_2 \rrbracket) = [\llbracket v_2 \rrbracket / x]\llbracket e \rrbracket$.

Case SS-E-Concat-Left Suppose $\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)$ We have by definition that

$$\llbracket \mathsf{rconcat}(e_1; e_2) \rrbracket = \mathsf{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket)$$
$$\llbracket \mathsf{rconcat}(e_1'; e_2) \rrbracket = \mathsf{pconcat}(\llbracket e_1' \rrbracket; \llbracket e_2 \rrbracket)$$

The only typing rule that applies is S-T-Concat, so $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$ and $\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$.
Inductively, we have that $\llbracket e_1 \rrbracket \mapsto^* \llbracket e_1' \rrbracket$.
By assupmtion **??**, we have that $\mathsf{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \mapsto^* \mathsf{pconcat}(\llbracket e_1' \rrbracket; \llbracket e_2 \rrbracket)$.

Case SS-E-Concat-Right Suppose $\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1; e_2')$ We have by definition that

$$\llbracket \mathsf{rconcat}(e_1; e_2) \rrbracket = \mathsf{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket)$$
$$\llbracket \mathsf{rconcat}(e_1; e_2') \rrbracket = \mathsf{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2' \rrbracket)$$

The only typing rule that applies is S-T-Concat, so $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$ and $\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$.
Inductively, we have that $\llbracket e_2 \rrbracket \mapsto^* \llbracket e_2' \rrbracket$.
By assupmtion **??**, we have that $\mathsf{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \mapsto^* \mathsf{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2' \rrbracket)$.

Case SS-E-Concat Suppose $\mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]$. We have by definition that

$$\llbracket \mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \rrbracket = \mathsf{pconcat}(\mathsf{str}[s_1]; \mathsf{str}[s_2])$$
$$\llbracket \mathsf{rstr}[s_1 s_2] \rrbracket = \mathsf{str}[s_1 s_2]$$

By Assumption **??**, $\mathsf{pconcat}(\mathsf{str}[s_1]; \mathsf{str}[s_2]) \mapsto^* \mathsf{str}[s_1 s_2]$.

Case SS-E-Case-Left Suppose $\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \mapsto \mathsf{rstrcase}(e_1'; e_2; x, y.e_3)$. The only typing rule that applies is S-T-Case, so $\emptyset \vdash e_1 : \mathsf{stringin}[r]$. We have by definition that:

$$\llbracket \mathsf{rstrcase}(e_1; e_2; x, y.e_3) \rrbracket = \mathsf{pstrcase}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket; \lambda x.\lambda y.\llbracket e_3 \rrbracket)$$
$$\llbracket \mathsf{rstrcase}(e_1'; e_2; x, y.e_3) \rrbracket = \mathsf{pstrcase}(\llbracket e_1' \rrbracket; \llbracket e_2 \rrbracket; \lambda x.\lambda y.\llbracket e_3 \rrbracket)$$

Inductively, $\llbracket e_1 \rrbracket \mapsto^* \llbracket e_1' \rrbracket$.
By Assumption **??**, we have that $\mathsf{pstrcase}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket; \lambda x.\lambda y.\llbracket e_3 \rrbracket) \mapsto^* \mathsf{pstrcase}(e_1'; e_2; \lambda x.\lambda y.\llbracket e_3 \rrbracket)$.

Case SS-E-Case-ϵ-Val Suppose $\mathsf{rstrcase}(\mathsf{rstr}[\epsilon]; e_2; x, y.e_3) \mapsto e_2$. The only typing rule that applies is S-T-Case, so $\emptyset \vdash e_2 : \sigma$. We have by definition that:

$$\llbracket \mathsf{rstrcase}(\mathsf{rstr}[\epsilon]; v_2; x, y.e_3) \rrbracket = \mathsf{pstrcase}(\mathsf{str}[\epsilon]; \llbracket v_2 \rrbracket; \lambda x.\lambda y.\llbracket e_3 \rrbracket)$$
$$\llbracket v_2 \rrbracket = \llbracket v_2 \rrbracket$$

Note that $\llbracket v_2 \rrbracket \mapsto^* \llbracket v_2 \rrbracket$.
By Assumption **??**, we have that $\mathsf{pstrcase}(\mathsf{str}[\epsilon]; \llbracket v_2 \rrbracket; \lambda x.\lambda y.\llbracket e_3 \rrbracket) \mapsto^* \llbracket v_2 \rrbracket$.

Case SS-E-Case-Concat Suppose $\mathsf{rstrcase}(\mathsf{rstr}[as]; e_2; x, y.e_3) \mapsto [a, s/x, y]e_3$. By definition,

$$\llbracket e \rrbracket = \mathsf{pstrcase}(\mathsf{str}[as]; \llbracket e_2 \rrbracket; \lambda x.\lambda y.\llbracket e_3 \rrbracket)$$
$$\llbracket [a, s/x, y]e_3 \rrbracket = [a, s/x, y]\llbracket e_3 \rrbracket$$

By Assumption **??**, $\mathsf{pstrcase}(\mathsf{str}[as]; \llbracket e_2 \rrbracket; \lambda x.\lambda y.\llbracket e_3 \rrbracket) \mapsto^* [a, s/x, y]\llbracket e_3 \rrbracket$.

Case SS-E-Replace-Left Suppose $\mathsf{rreplace}[r](e_1; e_2) \mapsto \mathsf{rreplace}[r](e_1'; e_2)$ and $e_1 \mapsto e_1'$. The only typing rule that applies is S-T-Replace, so $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$. By definition,

$$\llbracket \mathsf{rreplace}[r](e_1; e_2) \rrbracket = \mathsf{preplace}(\mathsf{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket)$$
$$\llbracket \mathsf{rreplace}[r](e_1'; e_2) \rrbracket = \mathsf{preplace}(\mathsf{rx}[r]; \llbracket e_1' \rrbracket; \llbracket e_2 \rrbracket)$$

Inductively, we have that $\llbracket e_1 \rrbracket \mapsto \llbracket e_1' \rrbracket$.
By Assumption **??**, $\mathsf{preplace}(\mathsf{rx}[r]; e_1; e_2) \mapsto^* \mathsf{preplace}(\mathsf{rx}[r]; e_1'; e_2)$.

Case SS-E-Replace-Right Suppose $\mathsf{rreplace}[r](e_1; e_2) \mapsto \mathsf{rreplace}[r](e_1; e_2 2)$ and $e_e \mapsto e_e'$. The only typing rule that applies is S-T-Replace, so $\emptyset \vdash e_2 : \mathsf{stringin}[r_2]$. By definition,

$$\llbracket \mathsf{rreplace}[r](e_1; e_2) \rrbracket = \mathsf{preplace}(\mathsf{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket)$$
$$\llbracket \mathsf{rreplace}[r](e_1; e_2') \rrbracket = \mathsf{preplace}(\mathsf{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e_2' \rrbracket)$$

Inductively, we have that $\llbracket e_2 \rrbracket \mapsto \llbracket e_2' \rrbracket$.
By Assumption **??**, $\mathsf{preplace}(\mathsf{rx}[r]; e_1; e_2) \mapsto^* \mathsf{preplace}(\mathsf{rx}[r]; e_1; e_2')$

Case SS-E-Replace Suppose $\mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[\mathsf{lreplace}(r, s_1, s_2)]$. By definition,

$$\llbracket \mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \rrbracket = \mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s_1]; \mathsf{str}[s_2])$$
$$\llbracket \mathsf{rstr}[\mathsf{lreplace}(r, s_1, s_2)] \rrbracket = \mathsf{str}[\mathsf{lreplace}(r, s_1, s_2)]$$

By assumption **??**, $\mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s_1]; \mathsf{str}[s_2]) \mapsto \mathsf{str}[\mathsf{lreplace}(r, s_1, s_2)]$.

Case SS-E-SafeCoerce-Step Supose $\mathsf{rcoerce}[r](e) \mapsto \mathsf{rcoerce}[r](e')$ and $e \mapsto^* r'$. By definition,

$$\llbracket \mathsf{rcoerce}[r](e) \rrbracket = \llbracket e \rrbracket$$
$$\llbracket \mathsf{rcoerce}[r](e') \rrbracket = \llbracket e' \rrbracket$$

The only rule that applies is S-T-SafeCoerce, so $\emptyset \vdash e' : \mathsf{stringin}[r]$.
Inductively, $\llbracket e \rrbracket \mapsto^* \llbracket e' \rrbracket$.

Case SS-E-SafeCoerce Suppose $\mathsf{rcoerce}[r](\mathsf{rstr}[s]) \mapsto \mathsf{rstr}[s]$. By definition,

$$\begin{aligned} [\![\mathsf{rcoerce}[r](\mathsf{rstr}[s])]\!] &= \mathsf{str}[s] \\ \mathsf{rstr}[s] &= \mathsf{str}[s] \end{aligned}$$

Note that $\mapsto^*$ is reflexive.

Case SS-E-Check-StepLeft Suppose $\mathsf{rcheck}[r](e; x.e_1; e_2) \mapsto \mathsf{rcheck}[r](e'; x.e_1; e_2)$ and $e \mapsto e'$. By definition,

$$\begin{aligned} [\![\mathsf{rcheck}[r](e; x.e_1; e_2)]\!] &= \mathsf{pcheck}(\mathsf{rx}[r]; [\![e]\!]; (\lambda x.[\![e_1]\!])([\![e]\!]); [\![e_2]\!]) \\ [\![\mathsf{rcheck}[r](e'; x.e_1; e_2)]\!] &= \mathsf{pcheck}(\mathsf{rx}[r]; [\![e']\!]; (\lambda x.[\![e_1]\!])([\![e']\!]); [\![e_2]\!]) \end{aligned}$$

Inductively, $e \mapsto^* e'$. By Assumption **??**, $\mathsf{pcheck}(\mathsf{rx}[r]; [\![e]\!]; (\lambda x.[\![e_1]\!])([\![e]\!]); [\![e_2]\!]) \mapsto^* \mathsf{pcheck}(\mathsf{rx}[r]; [\![e']\!]; (\lambda x.[\![e_1]\!])([\![e]\!]);$

Case SS-E-Check-Ok Suppose $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto [\mathsf{rstr}[s]/x]e_1$ and $s \in \mathcal{L}\{r\}$. By definition,

$$\begin{aligned} [\![\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2)]\!] &= \mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; (\lambda x.[\![e_1]\!])(\mathsf{str}[s]); [\![e_2]\!]) \\ [\![[\mathsf{rstr}[s]/x]e_1]\!] &= [\mathsf{str}[s]/x][\![e_1]\!] \end{aligned}$$

By Assumption **??**, $\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; (\lambda x.[\![e_1]\!])(\mathsf{str}[s]); [\![e_2]\!]) \mapsto^* [\mathsf{str}[s]/x][\![e_1]\!]$.

Case SS-E-Check-NotOk Suppose $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto e_2$ and $s \notin \mathcal{L}\{r\}$. By definition,

$$[\![\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2)]\!] = \mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; (\lambda x.[\![e_1]\!])(\mathsf{str}[s]); [\![e_2]\!])$$

By Assumption **??**, $\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; (\lambda x.[\![e_1]\!])(\mathsf{str}[s]); [\![e_2]\!]) \mapsto^* [\![e_2]\!]$.

□

Theorem 10 (Translation Correctness)**.** *If $\Psi \vdash e : \sigma$ then there exists an ι such that $[\![e]\!] = \iota$ and $[\![\Psi]\!] \vdash \iota : [\![\sigma]\!]$. Furthermore, if $e \mapsto^* v$ then $\iota \mapsto^* \dot{v}$ such that $[\![v]\!] = \dot{v}$.*

Proof. By the preceeding two theorems and induction on the length of the reduction of ι. □

Theorem 11 (Correctness of Input Sanitation for Translated Terms)**.** *If $[\![e]\!] = \iota$ and $\emptyset \vdash e : \mathsf{stringin}[r]$ then $\iota \mapsto^* \mathsf{str}[s]$ for $s \in \mathcal{L}\{r\}$.*

Proof. By **??**, $e \mapsto^* \mathsf{rstr}[s]$ where $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$. Therefore, $s \in \mathcal{L}\{r\}$. Note that $[\![\cdot]\!]$ is a function and $[\![\mathsf{rstr}[s]]\!] = \mathsf{str}[s]$; therefore, by theorem **??**, $\iota \mapsto^* \mathsf{str}[s]$. □

References

[1] N. Fulton, C. Omar, and J. Aldrich. Statically typed string sanitation inside a python. SPLASH '14. ACM, 2014.

$$r \quad ::= \quad \epsilon \mid a \mid r \cdot r \mid r + r \mid r* \qquad\qquad a \in \Sigma$$

Figure 1: Syntax of regular expressions over the alphabet Σ.

$$\begin{array}{lll l}
\sigma & ::= & \sigma \rightarrow \sigma \mid \mathsf{stringin}[r] & \text{source types} \\
e & ::= & x \mid v \mid e(e) & \text{source terms} \\
& \mid & \mathsf{rconcat}(e; e) \mid \mathsf{rstrcase}(e; e; x, y.e) & s \in \Sigma^* \\
& \mid & \mathsf{rreplace}[r](e; e) \mid \mathsf{rcoerce}[r](e) \mid \mathsf{rcheck}[r](e; x.e; e) & \\
v & ::= & \lambda x.e \mid \mathsf{rstr}[s] & \text{source values}
\end{array}$$

Figure 2: Syntax of λ_{RS}

$$\begin{array}{lll l}
\tau & ::= & \tau \rightarrow \tau \mid \mathsf{string} \mid \mathsf{regex} & \text{target types} \\
\iota & ::= & x \mid \dot{v} \mid \iota(\iota) & \text{target terms} \\
& \mid & \mathsf{pconcat}(\iota; \iota) \mid \mathsf{pstrcase}(\iota; \iota; x, y.\iota) & \\
& \mid & \mathsf{preplace}(\iota; \iota; \iota) \mid \mathsf{pcheck}(\iota; \iota; \iota; \iota) & \\
\dot{v} & ::= & \lambda x.\iota \mid \mathsf{str}[s] \mid \mathsf{rx}[r] & \text{target values}
\end{array}$$

Figure 3: Syntax of λ_P

$\boxed{\Psi \vdash e : \sigma}$ $\quad \Psi ::= \emptyset \mid \Psi, x : \sigma$

$$\textsc{S-T-Var}\ \frac{x : \sigma \in \Psi}{\Psi \vdash x : \sigma} \qquad \textsc{S-T-Abs}\ \frac{\Psi, x : \sigma_1 \vdash e : \sigma_2}{\Psi \vdash \lambda x.e : \sigma_1 \rightarrow \sigma_2} \qquad \textsc{S-T-App}\ \frac{\Psi \vdash e_1 : \sigma_2 \rightarrow \sigma \qquad \Psi \vdash e_2 : \sigma_2}{\Psi \vdash e_1(e_2) : \sigma} \qquad \textsc{S-T-Stringin-I}\ \frac{s \in \mathcal{L}\{r\}}{\Psi \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]}$$

$$\textsc{S-T-Concat}\ \frac{\Psi \vdash e_1 : \mathsf{stringin}[r_1] \qquad \Psi \vdash e_2 : \mathsf{stringin}[r_2]}{\Psi \vdash \mathsf{rconcat}(e_1; e_2) : \mathsf{stringin}[r_1 \cdot r_2]}$$

$$\textsc{S-T-Case}\ \frac{\Psi \vdash e_1 : \mathsf{stringin}[r] \qquad \Psi \vdash e_2 : \sigma \qquad \Psi, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma}{\Psi \vdash \mathsf{rstrcase}(e_1; e_2; x, y.e_3) : \sigma}$$

$$\textsc{S-T-Replace}\ \frac{\Psi \vdash e_1 : \mathsf{stringin}[r_1] \qquad \Psi \vdash e_2 : \mathsf{stringin}[r_2]}{\Psi \vdash \mathsf{rreplace}[r](e_1; e_2) : \mathsf{stringin}[\mathsf{lreplace}(r, r_1, r_2)]} \qquad \textsc{S-T-SafeCoerce}\ \frac{\Psi \vdash e : \mathsf{stringin}[r'] \qquad \mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}}{\Psi \vdash \mathsf{rcoerce}[r](e) : \mathsf{stringin}[r]}$$

$$\textsc{S-T-Check}\ \frac{\Psi \vdash e_0 : \mathsf{stringin}[r] \qquad \Psi, x : \mathsf{stringin}[r] \vdash e_1 : \sigma \qquad \Psi \vdash e_2 : \sigma}{\Psi \vdash \mathsf{rcheck}[r](e_0; x.e_1; e_2) : \sigma}$$

Figure 4: Typing rules for λ_{RS}. The typing context Ψ is standard.

$\boxed{e \mapsto e}$

SS-E-APPLEFT
$$\frac{e_1 \mapsto e_1'}{e_1(e_2) \mapsto e_1'(e_2)}$$

SS-E-APPRIGHT
$$\frac{e_2 \mapsto e_2'}{v_1(e_2) \mapsto v_1(e_2')}$$

SS-E-APPABS
$$\frac{}{(\lambda x : \sigma.e)v_2 \mapsto [v_2/x]e}$$

SS-E-CONCAT-LEFT
$$\frac{e_1 \mapsto e_1'}{\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)}$$

SS-E-CONCAT-RIGHT
$$\frac{e_2 \mapsto e_2'}{\mathsf{rconcat}(v_1; e_2) \mapsto \mathsf{rconcat}(v_1; e_2')}$$

SS-E-CONCAT
$$\frac{}{\mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]}$$

SS-E-CASE-LEFT
$$\frac{e_1 \mapsto e_1'}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \mapsto \mathsf{rstrcase}(e_1'; e_2; x, y.e_3)}$$

SS-E-CASE-ϵ-VAL
$$\frac{}{\mathsf{rstrcase}(\mathsf{rstr}[\epsilon]; e_2; x.y.e_3) \mapsto e_2}$$

SS-E-CASE-CONCAT
$$\frac{}{\mathsf{rstrcase}(\mathsf{rstr}[as]; e_2; x, y.e_3) \mapsto [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3}$$

SS-E-REPLACE-LEFT
$$\frac{e_1 \mapsto e_1'}{\mathsf{rreplace}[r](v_1; e_2) \mapsto \mathsf{rreplace}[r](v_1'; e_2)}$$

SS-E-REPLACE-RIGHT
$$\frac{e_2 \mapsto e_2'}{\mathsf{rreplace}[r](e_1; e_2) \mapsto \mathsf{rreplace}[r](e_1; e_2')}$$

SS-E-REPLACE
$$\frac{}{\mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]}$$

SS-E-SAFECOERCE-STEP
$$\frac{e \mapsto e'}{\mathsf{rcoerce}[r](e) \mapsto \mathsf{rcoerce}[r](e')}$$

SS-E-SAFECOERCE
$$\frac{}{\mathsf{rcoerce}[r](\mathsf{rstr}[s]) \mapsto \mathsf{rstr}[s]}$$

SS-E-CHECK-STEPLEFT
$$\frac{e \mapsto e'}{\mathsf{rcheck}[r](e; x.e_1; e_2) \mapsto \mathsf{rcheck}[r](e'; x.e_1; e_2)}$$

SS-E-CHECK-OK
$$\frac{s \in \mathcal{L}\{r\}}{\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto [\mathsf{rstr}[s]/x]e_1}$$

SS-E-CHECK-NOTOK
$$\frac{s \notin \mathcal{L}\{r\}}{\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto e_2}$$

Figure 5: Small step semantics for λ_{RS}.

$\boxed{\Theta \vdash \iota : \tau}$ $\quad \Theta ::= \emptyset \mid \Theta, x : \tau$

$$\text{P-T-VAR}\quad \frac{x : \tau \in \Theta}{\Theta \vdash x : \tau}$$

$$\text{P-T-ABS}\quad \frac{\Theta, x : \tau_1 \vdash \iota_2 : \tau_2}{\Theta \vdash \lambda x.\iota_2 : \tau_1 \to \tau_2}$$

$$\text{P-T-APP}\quad \frac{\Theta \vdash \iota_1 : \tau_2 \to \tau \qquad \Theta \vdash \iota_2 : \tau_2}{\Theta \vdash \iota_1(\iota_2) : \iota}$$

$$\text{P-T-STRING}\quad \frac{}{\Theta \vdash \mathsf{str}[s] : \mathsf{string}}$$

$$\text{P-T-REGEX}\quad \frac{}{\Theta \vdash \mathsf{rx}[r] : \mathsf{regex}}$$

$$\text{P-T-CONCAT}\quad \frac{\Theta \vdash \iota_1 : \mathsf{string} \qquad \Theta \vdash \iota_2 : \mathsf{string}}{\Theta \vdash \mathsf{pconcat}(\iota_1; \iota_2) : \mathsf{string}}$$

$$\text{P-T-CASE}\quad \frac{\Theta \vdash \iota_1 : \mathsf{string} \qquad \Theta \vdash \iota_2 : \tau \qquad \Theta, x : \mathsf{string}, y : \mathsf{string} \vdash \iota_3 : \tau}{\Theta \vdash \mathsf{pstrcase}(\iota_1; \iota_2; x, y.\iota_3) : \tau}$$

$$\text{P-T-REPLACE}\quad \frac{\Theta \vdash \iota_1 : \mathsf{regex} \qquad \Theta \vdash \iota_2 : \mathsf{string} \qquad \Theta \vdash \iota_3 : \mathsf{string}}{\Theta \vdash \mathsf{preplace}(\iota_1; \iota_2; \iota_3) : \mathsf{string}}$$

$$\text{P-T-CHECK}\quad \frac{\Theta \vdash \iota_1 : \mathsf{regex} \qquad \Theta \vdash \iota_2 : \mathsf{string} \qquad \Theta \vdash \iota_3 : \tau \qquad \Theta \vdash \iota_4 : \tau}{\Theta \vdash \mathsf{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4) : \tau}$$

Figure 6: Typing rules for λ_P. The typing context Θ is standard.

$\boxed{\iota \mapsto \iota}$

$$\text{PS-E-AppLeft}\quad \frac{\iota_1 \mapsto \iota_1'}{\iota_1(\iota_2) \mapsto \iota_1'(\iota_2)}$$

$$\text{PS-E-AppRight}\quad \frac{\iota_2 \mapsto \iota_2'}{\dot{v}_1(\iota_2) \mapsto \dot{v}_1(\iota_2')}$$

$$\text{PS-E-AppAbs}\quad \frac{}{(\lambda x : \tau.\iota)\dot{v}_2 \mapsto [\dot{v}_2/x]\iota}$$

$$\text{PS-E-ConcatLeft}\quad \frac{\iota_1 \mapsto \iota_1'}{\mathsf{pconcat}(\iota_1; \iota_2) \mapsto \mathsf{pconcat}(\iota_1'; \iota_2)}$$

$$\text{PS-E-ConcatRight}\quad \frac{\iota_2 \mapsto \iota_2'}{\mathsf{pconcat}(\mathsf{str}[s_1]; \iota_2) \mapsto \mathsf{pconcat}(\mathsf{str}[s_1]; \iota_2')}$$

$$\text{PS-E-Concat}\quad \frac{}{\mathsf{pconcat}(\mathsf{str}[s_1]; \mathsf{str}[s_2]) \mapsto \mathsf{str}[s_1 s_2]}$$

$$\text{PS-E-CaseLeft}\quad \frac{\iota_1 \mapsto \iota_1'}{\mathsf{pstrcase}(\iota_1; \iota_2; x, y.\iota_3) \mapsto \mathsf{pstrcase}(\iota_1'; \iota_2; x, y.\iota_3)}$$

$$\text{PS-E-Case-Epsilon}\quad \frac{}{\mathsf{pstrcase}(\mathsf{str}[\epsilon]; \iota_2; x, y.\iota_3) \mapsto \iota_2}$$

$$\text{PS-E-Case-Cons}\quad \frac{}{\mathsf{pstrcase}(\mathsf{str}[as]; \iota_2; x, y.\iota_3) \mapsto [\mathsf{str}[a], \mathsf{str}[s]/x, y]\iota_3}$$

$$\text{PS-E-ReplaceLeft}\quad \frac{\iota_1 \mapsto \iota_1'}{\mathsf{preplace}(\iota_1; \iota_2; \iota_3) \mapsto \mathsf{preplace}(\iota_1'; \iota_2; \iota_3)}$$

$$\text{PS-E-ReplaceMid}\quad \frac{\iota_2 \mapsto \iota_2'}{\mathsf{preplace}(\mathsf{rx}[r]; \iota_2; \iota_3) \mapsto \mathsf{preplace}(\mathsf{rx}[r]; \iota_2'; \iota_3)}$$

$$\text{PS-E-ReplaceRight}\quad \frac{\iota_3 \mapsto \iota_3'}{\mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s_2]; \iota_3) \mapsto \mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s_2]; \iota_3')}$$

$$\text{PS-E-Replace}\quad \frac{}{\mathsf{preplace}(\mathsf{rx}[r]; \mathsf{str}[s_2]; \mathsf{str}[s_3]) \mapsto \mathsf{str}[\mathsf{replace}(r; s_2; s_3)]}$$

$$\text{PS-E-CheckLeft}\quad \frac{\iota_1 \mapsto \iota_1'}{\mathsf{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4) \mapsto \mathsf{pcheck}(\iota_1'; \iota_2; \iota_3; \iota_4)}$$

$$\text{PS-E-CheckRight}\quad \frac{\iota_2 \mapsto \iota_2'}{\mathsf{pcheck}(\mathsf{rx}[r]; \iota_2; \iota_3; \iota_4) \mapsto \mathsf{pcheck}(\mathsf{rx}[r]; \iota_2'; \iota_3; \iota_4)}$$

$$\text{PS-E-Check-OK}\quad \frac{s \in \mathcal{L}\{r\}}{\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; \iota_3; \iota_4) \mapsto \iota_3}$$

$$\text{PS-E-Check-NotOK}\quad \frac{s \notin \mathcal{L}\{r\}}{\mathsf{pcheck}(\mathsf{rx}[r]; \mathsf{str}[s]; \iota_3; \iota_4) \mapsto \iota_4}$$

Figure 7: Small step semantics for λ_P

.

$\boxed{[\![\sigma]\!] = \tau}$

$$
\begin{aligned}
[\![\mathsf{stringin}[r]]\!] &= \mathsf{string} \\
[\![\sigma_1 \to \sigma_2]\!] &= [\![\sigma_1]\!] \to [\![\sigma_2]\!]
\end{aligned}
$$

$\boxed{[\![\Psi]\!] = \Theta}$

$$
\begin{aligned}
[\![\emptyset]\!] &= \emptyset \\
[\![\Psi, x : \sigma]\!] &= [\![\Psi]\!], x : [\![\sigma]\!]
\end{aligned}
$$

$\boxed{[\![e]\!] = \iota}$

$$
\begin{aligned}
[\![x]\!] &= x \\
[\![\lambda x : \sigma.e]\!] &= \lambda x : [\![\sigma]\!].[\![e]\!] \\
[\![e_1(e_2)]\!] &= [\![e_1]\!]([\![e_2]\!]) \\
[\![\mathsf{rstr}[s]]\!] &= \mathsf{str}[s] \\
[\![\mathsf{rstrcase}(e_1; e_2; x, y.e_3)]\!] &= \mathsf{pstrcase}([\![e_1]\!]; [\![e_2]\!]; x, y.[\![e_3]\!]) \\
[\![\mathsf{rconcat}(e_1; e_2)]\!] &= \mathsf{pconcat}([\![e_1]\!]; [\![e_2]\!]) \\
[\![\mathsf{rreplace}[r](e_1; e_2)]\!] &= \mathsf{preplace}(\mathsf{rx}[r]; [\![e_1]\!]; [\![e_2]\!]) \\
[\![\mathsf{rcoerce}[r](e)]\!] &= [\![e]\!] \\
[\![\mathsf{rcheck}[r](e; x.e_1; e_2)]\!] &= \mathsf{pcheck}(\mathsf{rx}[r]; [\![e]\!]; (\lambda x : \mathsf{string}.[\![e_1]\!])([\![e]\!]); [\![e_2]\!])
\end{aligned}
$$

Figure 8: Translation from λ_{RS} to λ_P