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" [1], 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 [1] are numbered correspondingly, while supporting facts appearing only in the Technical Report are lettered.

2 Regular Expressions

The syntax of regular expressions over some alphabet Σ is shown in Figure 1.

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 2. The static semantics is specified in Figure 4.

3.1 Head and Tail Operations

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 $\mathsf{ltail}(r) = \delta_{a_1}(r) + \delta_{a_2}(r) + ... + \delta_{a_i}(r)$.

3.2 Replacement

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

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

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

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

3.3 Small Step Semantics of λ_{RS}

Figure 7 specifies a small-step operational semantics for λ_{RS}.

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. □

Lemma G (Progress)**.** *If $\emptyset \vdash e : \sigma$ either $e = v$ for some v or $e \mapsto e'$ for some 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 $e = \mathsf{rreplace}[r](e_1; e_2)$, $\emptyset \vdash e : \mathsf{stringin}[\mathtt{lreplace}(r, r_1, r_2)]$ and:

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

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

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:

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

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

(5) $$\emptyset \vdash e_2 : \sigma$$

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 B 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.

□

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.

SS-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, $\emptyset \vdash \mathsf{rconcat}(e_1'; e_2) : \mathsf{stringin}[r_1 r_2]$.

SS-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, $\emptyset \vdash \mathsf{rconcat}(e_1; e_2') : \mathsf{stringin}[r_1 r_2]$.

SS-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 B 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 (6) 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 (7) and (8) implies by S-T-Case that $\emptyset \vdash \mathsf{rstrcase}(e_1'; e_2; x, y.e_3) : \sigma$.

SS-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 $\emptyset \vdash e_2 : \sigma$.

SS-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 (9) Therefore, $a \in \mathcal{L}\{\mathsf{lhead}(r)\}$ by Condition C and $s \in \mathcal{L}\{\mathsf{ltail}(r)\}$ by Condition D.

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 H that $\emptyset \vdash [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 : \sigma$. Cyrus stopped here

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

SS-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 $\emptyset \vdash e : \mathsf{stringin}[\mathtt{lreplace}(r, r_1, r_2)]$ 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, $\emptyset \vdash \mathsf{rreplace}[r](r_1'; r_2) : \mathsf{stringin}[\mathtt{lreplace}(r, r_1, r_2)]$ by S-T-Replace.

Case SS-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, $\mathtt{lreplace}(r, s_1, s_2) \in \mathcal{L}\{\mathtt{lreplace}(r, r_1, r_2)\}$ by Theorem E. It is finally derivable by S-T-Rstr that:

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

Case SS-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 $\emptyset \vdash \mathsf{rcoerce}[r](s_1) : \mathsf{stringin}[r]$. By Canonical Forms, $s \in \mathcal{L}\{r\}$. Therefore, $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$.

Case SS-E-Check-Ok. Suppose $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto [\mathsf{rstr}[s]/x]e_1$, $s \in \mathcal{L}\{r\}$, and $\emptyset \vdash \mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) : \sigma$. By inversion of S-T-Check, $x : \mathsf{stringin}[r] \vdash e_1 : \sigma$. Note that $s \in \mathcal{L}\{r\}$ implies that $s : \mathsf{stringin}[r]$ by S-T-RStr. Therefore, $\emptyset \vdash [\mathsf{rstr}[s]/x]e_1 : \sigma$.

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

3.3.1 Semantic Correspondence between Big and Small Step Semantics for λ_{RS}

Before extending the previous theorem to the big step semantics, we first establish a correspondence between the big step semantics in Figure 7 and the small step semantics in Figure 5.

Lemma K. *If* $e \Downarrow v$ *and* $e \mapsto e'$ *then* $e' \Downarrow v$.

Proof. By induction on the structure of e.

Case $e = e_1(e_2)$. The only applicable rule is S-E-App, so $e_1 \Downarrow \lambda x.e_3$ and $e_2 \Downarrow v_2$ such that $[v_2/x]e_3 \Downarrow v$.

The term $e = e_1(e_2)$ may step by three rules.

First, if e steps by L-E-AppLeft then $e_1(e_2) \mapsto e_1'(e_2)$. By induction, $e_1' \Downarrow \lambda x.e_3$. By S-E-App, $e_1'(e_2) \Downarrow \lambda x.e_3$.

Second, if e steps by L-E-AppRight then $e_1(e_2) \mapsto e_1(e_2')$. By induction, $e_2' \Downarrow v_2$. By S-E-App, $e_1(e_2') \Downarrow \lambda x.e_3$.

Third, if e steps by L-E-AppAbs then $e = (\lambda x.e_3)(v_2) \mapsto [v_2/x]e_3$. By induction, $e = (\lambda x.e')v_2 \Downarrow v$.

Other lambda cases

$s = \mathsf{concat}(e_1; e_2)$. The only big-step rule that applies is S-E-Concat, so $e \Downarrow v$ where $v = \mathsf{rstr}[s_1 s_2]$, $e_1 \Downarrow \mathsf{rstr}[s_1]$, and $e_2 \Downarrow \mathsf{rstr}[s_2]$.

If $e_1 = \mathsf{rstr}[e_1]$ and $e_2 = \mathsf{rstr}[e_2]$ then $e \mapsto v$. Otherwise either $e_1 \mapsto e_1'$ or else $e_2 \mapsto e_2'$.

In the former case, $e \mapsto \mathsf{concat}(e_1'; e_2)$ by S-E-Concat-Left. By induction, $e_1' \Downarrow \mathsf{rstr}[s_1]$ and so $\mathsf{concat}(e_1'; e_2) \Downarrow v$.

In the latter case, $e \mapsto \mathsf{concat}(e_1; e_2')$ by S-E-Concat-Right. By induction, $e_2 \Downarrow e_2'$ form which it follows by S-E-Concat that $\mathsf{concat}(e_1; e_2') \Downarrow v$.

$s = \mathsf{rstrcase}(e_1; e_2; x, y.e_3)$. finish...

$\square$

Theorem L (Semantic Correspondence for λ_{RS} (Part I))**.** *If $e \Downarrow v$ then $e \mapsto^* v$.*

Proof. We proceed by structural induction on e.

Case $e = \lambda x.e_1$. The only applicable rule is S-E-Abs, so $v = \lambda x.e_1$. Note that $\lambda x.e_1 \mapsto^* \lambda x.e_1$ by RT-Refl.

Case $e = e_1(e_2)$. The only applicable rule is S-E-App. By inversion:

$$\begin{aligned} e_1 &\Downarrow \lambda x.e_1' \\ e_2 &\Downarrow v_2 \\ [v_2/x]e_1' &\Downarrow v \end{aligned}$$

From which it follows by induction that:

$$\begin{aligned} e_1 &\mapsto^* \lambda x.e_1' \\ e_2 &\mapsto^* v_2 \\ [v_2/x]e_1' &\mapsto^* v \end{aligned}$$

If $e_1 = \lambda x.e_1'$ and $e_2 = v_2$ (henceforth the reflexive case) then $e \mapsto [v_2/x]e_1'$ and the conclusion follows by RT-Trans.

If $e_1 \mapsto \lambda x.e_1'$ then $e_1(e_2) \mapsto (\lambda x.e_1')(e_2) \mapsto [e_2/x]e_1'$ and the conclusion follows by two applications of RT-Trans.

If $e_1 \mapsto^k \lambda x.e'_2$ then $e_1 \mapsto e'$ so $e_1(e_2) \mapsto e'(e_2)$ and by Lemma K $e'(e_2) \mapsto^* v$. So $e_1(e_2) \mapsto e'(e_2) \mapsto^* v$; it follows by RT-Trans that $e_1(e_2) \mapsto v$.

not modified below this line...

Note that the following rule is derivable by repeating applications of the left and right compatibility rules for application:

$$\textsc{L*-App}\quad \frac{e_1 \mapsto^* e'_1 \qquad e_2 \mapsto^* e'_2}{e_1(e_2) \mapsto^* e'_1(e'_2)}$$

From these facts and L-AppAbs, we may establish that $e_1(e_2) \mapsto^* (\lambda x.e_2)(v_2) \mapsto [v_2/x]e_2$. Note that $[v_2/x]e_2 \mapsto^* v$, so by RT-Trans it follows that $e = e_1(e_2) \mapsto^* v$.

Case $e = \mathsf{rstr}[s]$. The only applicable rule is S-E-RStr, so $v = \mathsf{rstr}[s]$. By RT-Refl, $\mathsf{rstr}[s] \mapsto^* \mathsf{rstr}[s]$.

Case $e = \mathsf{rconcat}(e_1; e_2)$. The only applicable rule is S-E-Concat, so $v = \mathsf{rstr}[s_1 s_2]$. By inversion, $e_1 \Downarrow \mathsf{rstr}[s_1]$ and $e_2 \Downarrow \mathsf{rstr}[s_2]$. By induction, $e_1 \mapsto^* \mathsf{rstr}[s_1]$ and $e_2 \mapsto^* \mathsf{rstr}[s_2]$. Note that the rule following is derivable:

$$\textsc{SS-E-Concat-LR*}\quad \frac{e_1 \mapsto^* e'_1 \qquad e_2 \mapsto^* e'_2}{\mathsf{rconcat}(e_1; e_2) \mapsto^* \mathsf{rconcat}(e'_1; e'_2)}$$

From these facts, it follows that $\mathsf{rconcat}(e_1; e_2) \mapsto^* \mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2])$. Finally, $\mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]$ by SS-E-Concat. By RT-Step, it follows that $\mathsf{rconcat}(e_1; e_2) \mapsto^* \mathsf{rstr}[s_1 s_2]$.

Case $e = \mathsf{rstrcase}(e_1; e_2; x, y.e_3)$.

There are two subcases. For the first, suppose $\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \Downarrow v$ was finally derived by S-E-Case-ϵ. By inversion:

$$e_1 \Downarrow \mathsf{rstr}[\epsilon]$$
$$e_2 \Downarrow v$$

from which it follows by induction that:

$$e_1 \mapsto^* \mathsf{rstr}[\epsilon]$$
$$e_2 \mapsto^* v$$

Note that the following rule is derivable:

$$\textsc{SS-E-Case-LR*}\quad \frac{e_1 \mapsto^* e'_1 \qquad e_2 \mapsto^* e'_2}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \mapsto^* \mathsf{rstrcase}(e'_1; e'_2; x, y.e_3)}$$

From these facts is follows that $e \mapsto^* \mathsf{rstrcase}(\mathsf{rstr}[\epsilon]; v; x, y.e_3)$. By S-E-Case-$\epsilon$-Val and RT-Step it follows that $e \mapsto^* v$.

Now consider the other case where $\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \Downarrow v$ was finally derived by S-E-Case-Concat. By inversion, $e_1 \Downarrow \mathsf{rstr}[as]$ and $[\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \Downarrow v$. From these facts it follows by induction that $e_1 \mapsto^* \mathsf{rstr}[as]$ and $[\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \mapsto^* v$.

By the first of these facts, it is derivable via SS-E-Case-LR* that $e \mapsto^* \mathsf{rstrcase}(e_1'; \mathsf{rstr}[as]; x, y.e_3)$. SE-E-Case-Concat applies to this form, so by RT-Step we know $e \mapsto^* [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3$. Recall that $[\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \mapsto^* v$, so by RT-Trans we finally derive $e \mapsto^* v$.

Case $e = \mathsf{rreplace}[r](e_1; e_2)$. There is only one applicable rule, so $v = \mathsf{rstr}[s]$ and by inversion it follows that:

$$e_1 \Downarrow \mathsf{rstr}[s_1]$$
$$e_2 \Downarrow \mathsf{rstr}[s_2]$$

From which it follows by induction that:

$$e_1 \mapsto^* \mathsf{rstr}[s_1]$$
$$e_2 \mapsto^* \mathsf{rstr}[s_2]$$

Furthermore, $\mathsf{replace}(r; s_1; s_2) = s$ by induction. Note that the following rule is derivable:

$$\textsc{SS-E-Replace-LR*}\quad \frac{e_1 \mapsto^* e_1' \qquad e_2 \mapsto^* e_2'}{\mathsf{rreplace}[r](e_1; e_2) \mapsto^* \mathsf{rreplace}[r](e_1'; e_2')}$$

From these facts, $\mathsf{rreplace}[r](e_1; e_2) \mapsto^* \mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2])$.

Finally, $\mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{replace}(r; s_1; s_2)$.

From these two facts we know via RT-Step that $\mathsf{rreplace}[r](e_1; e_2) \mapsto^* \mathsf{rreplace}[r](e_1; e_2)$. Recall that $\mathsf{replace}(r; s_1; s_2) = s$, from which the conclusion follows.

Case $e = \mathsf{rcoerce}[r](e_1)$. In this case $e \Downarrow v$ is only finally derivable via S-E-SafeCoerce. Therefore, $v = \mathsf{rstr}[s]$ and by inversion $e_1 \Downarrow \mathsf{rstr}[s]$. By induction, $e_1 \mapsto^* \mathsf{rstr}[s]$.

The following rule is derivable:

$$\textsc{SS-E-SafeCoerce-Step}\quad \frac{e \mapsto^* e'}{\mathsf{rcoerce}[r](e) \mapsto^* \mathsf{rcoerce}[r](e')}$$

Applying this rule at $e_1 \mapsto^* \mathsf{rstr}[s]$ derives $\mathsf{rcoerce}[r](e_1) \mapsto^* \mathsf{rcoerce}[r](\mathsf{rstr}[s])$. In the final step, $\mathsf{rcoerce}[r](\mathsf{rstr}[s]) \mapsto \mathsf{rstr}[s]$ by SS-E-SafeCoerce. From this fact, we may derive via RT-Trans that $e \mapsto^* \mathsf{rstr}[s]$ as required.

Case $e = \mathsf{rcheck}[r](e_1; x.e_2; e_3)$.

Note that the rule following is derivable:

$$\frac{\text{SS-E-CHECK-STEP} \quad e_1 \mapsto^* e_1' \qquad e_3 \mapsto^* e_3'}{\mathsf{rcheck}[r](e_1; x.e_2; e_3) \mapsto^* \mathsf{rcheck}[r](e_1'; x.e_2; e_3')}$$

There are two ways to finally derive $e \Downarrow v$. In both cases, $e_1 \Downarrow \mathsf{rstr}[s]$ by inversion. Therefore, in both cases, $e_1 \mapsto^* \mathsf{rstr}[s]$ by induction and so $e \mapsto^* \mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_2; e_3)$ by SS-E-Check-Step.

Suppose $e \Downarrow v$ is finally derived via SS-E-Check-Ok. By the facts mentioned above and SS-E-Check-Step, $e \mapsto^* \mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_2; e_2)$. Note that by inversion $s \in \mathcal{L}\{r\}$. Therefore, SS-E-Check-Ok applies and so by RT-Trans $e \mapsto^* [\mathsf{rstr}[s]/x]e_1$. By inversion, $[\mathsf{rstr}[s]/x]e_1 \Downarrow v$. Therefore, by induction and RT-Step $e \mapsto^* v$ as required.

Suppose that $e \Downarrow v$ is instead finally derived via SS-E-Check-NotOk. By inversion, $e_3 \Downarrow v$ and by induction $e_3 \mapsto^* v$. From these facts at SS-E-Check-Step, it is derivable that $e \mapsto^* \mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_2; v)$.

Also by inversion, $s \notin \mathcal{L}\{r\}$ and so SS-E-Check-NotOk applies. Therefore, $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_2; v) \mapsto v$.

The conclusion $e \mapsto^* v$ follows from these facts by RT-Step.

$\square$

Theorem M (Semantic Correspondence for λ_{RS} (Part II))**.** *If* $\emptyset \vdash e : \sigma$*,* $e \mapsto^* v$ *and* v val *then* $e \Downarrow v$*.*

Proof. The proof proceeds by structural induction on e.

Case $e = \mathsf{concat}(e_1; e_2)$. By inversion, $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$. By Type Safety, Canonical Forms and Termination it follows that $e_1 \mapsto^* \mathsf{rstr}[s_1]$ for some s_1. By induction, $e_1 \Downarrow \mathsf{rstr}[s_1]$.

Similarly, $e_2 \mapsto^* \mathsf{rstr}[s_2]$ and $e_2 \Downarrow \mathsf{rstr}[s_2]$.

Note that $\mathsf{concat}(e_1; e_2) \mapsto^* \mathsf{concat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]$ by SS-E-Concat-LR* and S-E-Concat. Therefore, $e \mapsto^* \mathsf{rstr}[s_1 s_2]$ by RT-Step. So it suffices to show that $e \Downarrow \mathsf{rstr}[s_1 s_2]$.

Finally, $e \Downarrow \mathsf{rstr}[s_1 s_2]$ follows via S-E-Concat from the facts that $e_1 \Downarrow \mathsf{rstr}[s_1]$ and $e_2 \Downarrow \mathsf{rstr}[s_2]$. This completes the case.

Case $e = \mathsf{rreplace}[r](e_1; e_2)$. By inversion of S-T-Replace, $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$ for some r_1. It follows by Type Safety, Termination and Canonical Forms that $e_1 \mapsto^* \mathsf{rstr}[s_1]$. By induction, $e_1 \Downarrow \mathsf{rstr}[s_1]$.

Similarly, $e_2 \mapsto^* \mathsf{rstr}[s_2]$ and $e_2 \Downarrow \mathsf{rstr}[s_2]$.

Note that $e \mapsto^* \mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]$ by SS-Replace-LR* and SS-E-Replace. Therefore $e \mapsto^* \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]$ by RT-Step.

It suffices to show $e \Downarrow \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]$, which follows by S-E-Replace from the facts that $e_1 \Downarrow \mathsf{rstr}[s_1]$ and $e_2 \Downarrow \mathsf{rstr}[s_2]$.

Case $e = \mathsf{rstrcase}(e_1; e_2; x.y.e_3)$. By inversion, $\emptyset \vdash e_1 : \mathsf{stringin}[r]$ and $e_2 : \sigma$. By Type Safety, Canonical Forms and Termination $e_1 \mapsto^* \mathsf{stringin}[s_1]$ and by induction $e_1 \Downarrow \mathsf{stringin}[s_1]$. Similarly, $e_2 \mapsto^* v_2$ and $\emptyset \vdash e_2 \Downarrow v_2$.

By SS-E-Case-LR*, $\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \mapsto^* \mathsf{rstrcase}(v_1; v_2; x, y.e_3)$.

Note that either $s_1 = \epsilon$ or $s_1 = as$ because we define strings as either empty or finite sequences of characters. We proceed by cases.

If $s_1 = \epsilon$ then $\mathsf{rstrcase}(v; v_2; x, y.e_3) \mapsto v_2$ by SS-E-Case-ϵ. Therefore, by RT-Step, $e \mapsto^* v_2$. Recall $e_1 \Downarrow \mathsf{rstr}[\epsilon]$ and $e_2 \Downarrow v_2$, which is enough to establish by S-E-Case-ϵ that $e \Downarrow v_2$.

If $s_1 = as$ instead, then $\mathsf{rstrcase}(\mathsf{rstr}[s_1]; v_2; x, y.e_3) \mapsto [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3$ by SS-E-Case-Concat. Inversion of the typing relation satisfies the assumptions necessary to appeal to termination. Therefore,

$$[\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \mapsto^* v \text{ for } v \text{ val.}$$

It follows by RT-Step that $e \mapsto^* v$.

Note that the substitution does not change the structure of e_3. So by induction, $[\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \Downarrow v$. Recall that $e_1 \Downarrow \mathsf{rstr}[s_1]$ and so by S-E-Case it follows that $e \Downarrow [a, s/x, y]e_3 \Downarrow v$.

□

The cases for coercion and checking are straightforward.

3.4 Extension of Safety for Small Step Semantics

Theorem 4 (Type Safety)**.** *If $\emptyset \vdash e : \sigma$ and $e \Downarrow e'$ then $\emptyset \vdash e' : \sigma$.*

Proof. If $\emptyset \vdash e : \sigma$ then $e \mapsto^* e'$. Therefore, $e \Downarrow e'$ by part 2 of the semantic correspondence theorem. Since $\emptyset \vdash e : \sigma$ and $e \mapsto^* e'$, it follows that $\emptyset \vdash e' : \sigma$ by type safety for the small step semantics. □

3.4.1 The Security Theorem

Theorem 5 (Correctness of Input Sanitation for λ_{RS})**.** *If $\emptyset \vdash e : \mathsf{stringin}[r]$ and $e \Downarrow \mathsf{rstr}[s]$ then $s \in \mathcal{L}\{r\}$.*

Proof. If $\emptyset \vdash e : \mathsf{stringin}[r]$ and $e \Downarrow \mathsf{rstr}[s]$ then $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$ by Type Safety. By Caonical Forms, $s \in \mathcal{L}\{r\}$. □

4 Proofs of Lemmas and Theorems About λ_P

This section follows the same structure as the safety proof for λ_{RS} – we prove type safety for a small-step semantics, prove a semantic correspondence, and then transfer the safety result to the big-step semantics in the paper.

Lemma 6 (Canonical Forms for Target Language)**.**

- *If $\emptyset \vdash \iota : \mathsf{regex}$ then $\iota \Downarrow \mathsf{rx}[r]$ such that r is a well-formed regular expression.*
- *If $\emptyset \vdash \iota : \mathsf{string}$ then $\iota \Downarrow \mathsf{str}[s]$.*

Theorem 7 (Progress)**.** *If $\emptyset \vdash \iota : \tau$ either $\iota = \dot{v}$ or $\iota \mapsto \iota'$ for some ι'.*

Proof. The proof proceeds by induction on the typing assumption. Consider only the string and regex (non-λ) fragments of λ_P.

P-T-Case. Suppose $\emptyset \vdash \mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3)$. By inversion, $\iota_1 : \mathsf{string}$ and so either $\iota_1 \mapsto \iota_1'$ or by canonical forms, $\iota_1 = \mathsf{str}[s_1]$. Similarly, $\iota_2 \mapsto \iota_2'$ or else $\iota_2 = \mathsf{str}[s_2]$. In the former cases, progress occurs via the compatibility rules. in the case where both are string values, progress occurs via the case concatenation rule.

P-T-Replace. Suppose $\emptyset \vdash \mathsf{replace}(\iota_1; \iota_2; \iota_3)$. By inversion, $\iota_1 : \mathsf{regex}$ and so by canonical forms $\iota_1 = \mathsf{rx}[r]$. By inversion, $\iota_2 : \mathsf{string}$ and so by induction either $\iota_2 \mapsto \iota_2'$ or else $\iota_2 = \mathsf{str}[s_2]$ for some string s_2. Similarly, either ι_3 steps or else $\iota_3 = \mathsf{str}[s_3]$. In case any steps occur, progress occurs. In the remaining case, PP-E-Replace applies and so progress occurs.

P-T-Check. Finally, suppose $\emptyset \vdash \check{\iota_x}\iota_1\iota_2\iota_3$. In case any of these step, then progress occurs. In the remaining cases, applications of inversion and canonical forms for each ι_x and ι_1 implies that the term at hand equals $\mathsf{rx}\check{[r]}\mathsf{str}[s]\iota_2\iota_3$, which evaluates to either ι_2 or ι_3.

□

Lemma N (Substitution Lemma)**.** *If $\theta, x : \tau \vdash \iota : \tau'$ and $\theta \vdash \iota' : \tau$ then $\theta \vdash [\iota'/x]\iota : \tau'$.*

Theorem 8 (Preservation)**.** *If $\emptyset \vdash \iota : \tau$ and $\iota \mapsto \iota'$ then $\emptyset \vdash \iota' : \tau$.*

Proof. The proof proceeds by induction of the derivations of $\emptyset \vdash \iota : \tau$ and $\iota \mapsto \iota'$.

We treat only the non-lambda fragment.

Case PS-E-ConcatLeft. Suppose:

$$\begin{aligned}&\iota = \mathsf{rconcat}(\iota_1; \iota_2) \mapsto \mathsf{rconcat}(\iota_1'; \iota_2)\\&\emptyset \vdash \iota : \mathsf{string}\\&\iota \mapsto \iota'\end{aligned}$$

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 $\emptyset \vdash \mathsf{rconcat}(\iota_1'; \iota_2) : \mathsf{string}$.

Case PS-E-ConcatRight

$$\begin{aligned}&e = \mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1; e_2')\\&\emptyset \vdash e : \mathsf{string}\\&\iota \mapsto \iota'\end{aligned}$$

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 $\emptyset \vdash \mathsf{rconcat}(\iota_1; \iota_2') : \mathsf{string}$.

Case PS-E-Concat Let $e = \mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]$. The only rule that applies is P-T-Concat, so $\emptyset \vdash e : \mathsf{string}$. By canonical forms, $\emptyset \mathsf{rstr}[s_1 s_2] : \mathsf{string}$.

Case PS-E-CaseLeft Let $\iota = \mathsf{rstrcase}(\iota_1; \iota_2; x, y.\iota_3) \mapsto \mathsf{rstrcase}(\iota_1'; \iota_2; x, y.\iota_3)$ when $\iota_1 \mapsto \iota_1'$. The only rule that applies is P-T-Case, so $\emptyset \vdash \iota : \tau$ where:

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

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

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

Case PS-E-Case Let $\iota = \mathsf{rstrcase}(\mathsf{rstr}[as]; \iota_2; x, y.\iota_3) \mapsto [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]\iota_3$ The only rule that applies is P-T-Case, so $\emptyset \vdash \iota : \tau$ where:

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

The result follows by the substitution lemma.

Case PS-E-ReplaceLeft Let $\iota = \mathsf{rreplace}[\iota_1](\iota_2; \iota_3) \mapsto \mathsf{rreplace}[\iota_1'](\iota_2; \iota_3)$ where $\iota_1 \mapsto \iota_1'$. The applicable typing rule is P-T-Replace, so $\emptyset \vdash \iota : \mathsf{string}$ where:

$$\emptyset \vdash \iota_1 : \mathsf{regex}$$
$$\emptyset \vdash \iota_2 : \mathsf{string}$$
$$\emptyset \vdash \iota_3 : \mathsf{string}$$

By induction, $\emptyset \vdash \iota_1' : \mathsf{regex}$. Therefore, $\emptyset \vdash \mathsf{rreplace}[\iota_1'](\iota_2; \iota_3)$.

Case PS-E-ReplaceMid Let $\iota = \mathsf{rreplace}[\iota_1](\iota_2; \iota_3) \mapsto \mathsf{rreplace}[\iota_1](\iota_2'; \iota_3)$ where $\iota_2 \mapsto \iota_2'$. The applicable typing rule is P-T-Replace, so $\emptyset \vdash \iota : \mathsf{string}$ where:

$$\emptyset \vdash \iota_1 : \mathsf{regex}$$
$$\emptyset \vdash \iota_2 : \mathsf{string}$$
$$\emptyset \vdash \iota_3 : \mathsf{string}$$

By induction, $\emptyset \vdash \iota_2' : \mathsf{string}$. Therefore, $\emptyset \vdash \mathsf{rreplace}[\iota_1](\iota_2'; \iota_3)$.

Case PS-E-ReplaceRight Let $\iota = \mathsf{rreplace}[\iota_1](\iota_2; \iota_3) \mapsto \mathsf{rreplace}[\iota_1](\iota_2; \iota_3')$ where $\iota_3 \mapsto \iota_3'$. The applicable typing rule is P-T-Replace, so $\emptyset \vdash \iota : \mathsf{string}$ where:

$$\emptyset \vdash \iota_1 : \mathsf{regex}$$
$$\emptyset \vdash \iota_2 : \mathsf{string}$$
$$\emptyset \vdash \iota_3 : \mathsf{string}$$

By induction, $\emptyset \vdash \iota_3' : \mathsf{string}$. Therefore, $\emptyset \vdash \mathsf{rreplace}[\iota_1](\iota_2; \iota_3')$.

Case PS-E-Replace Let $\iota = \mathsf{rreplace}[\mathsf{rx}[r]](\mathsf{rstr}[s_2]; \mathsf{rstr}[s_3]) \mapsto \mathsf{rstr}[\mathtt{lreplace}(r, s_2, s_3)]$. The applicable typing rule is P-T-Replace, so $\emptyset \vdash \iota : \mathsf{string}$. The result follows by canonical forms.

Case PS-E-CheckLeft Let $\iota = \mathsf{rcheck}[\iota_x](\iota_1; \iota_2; \iota_3) \mapsto \mathsf{rcheck}[\iota_x'](\iota_1; \iota_2; \iota_3)$ where $\iota_x \mapsto \iota_x'$. The applicable typing rule is P-T-Check, so $\emptyset \vdash \iota : \tau$ where:

$$\emptyset \vdash \iota_x : \mathsf{regex}$$
$$\emptyset \vdash \iota_1 : \mathsf{string}$$
$$\emptyset \vdash \iota_2 : \tau$$
$$\emptyset \vdash \iota_3 : \tau$$

By induction, $\iota_x : \mathsf{regex}$. Therefore, $\emptyset \mathsf{rcheck}[\iota_x'](\iota_1; \iota_2; \iota_3) : \tau$.

Case PS-E-CheckRight Let $\iota = \mathsf{rcheck}[\iota_x](\iota_1; \iota_2; \iota_3) \mapsto \mathsf{rcheck}[\iota_x](\iota_1'; \iota_2; \iota_3)$ where $\iota_1 \mapsto \iota_1'$. The applicable typing rule is P-T-Check, so $\emptyset \vdash \iota : \tau$ where:

$$\emptyset \vdash \iota_x : \mathsf{regex}$$
$$\emptyset \vdash \iota_1 : \mathsf{string}$$
$$\emptyset \vdash \iota_2 : \tau$$
$$\emptyset \vdash \iota_3 : \tau$$

By induction, $\iota_1' : \mathsf{string}$. Therefore, $\emptyset \mathsf{rcheck}[\iota_x](\iota_1'; \iota_2; \iota_3) : \tau$.

Case PS-E-Check-Ok Let $\iota = \mathsf{rcheck}[\mathsf{rx}[r]](\mathsf{rstr}[s]; \iota_2; \iota_3) \mapsto \iota_2$ and $s \in \mathcal{L}\{r\}$. The applicable typing rule is P-T-Check, so $\emptyset \vdash \iota : \tau$ where $\emptyset \vdash \iota_2 : \tau$.

Case PS-E-Check-NotOk Let $\iota = \mathsf{rcheck}[\mathsf{rx}[r]](\mathsf{rstr}[s]; \iota_2; \iota_3) \mapsto \iota_3$ where $s \notin \mathcal{L}\{r\}$. The applicable typing rule is P-T-Check, so $\emptyset \vdash \iota : \tau$ where $\emptyset \vdash \iota_3 : \tau$.

□

Theorem 9 (Safety for Small-Step Semantics). *If $\iota \Downarrow \dot{v}$ and $\iota \mapsto \iota'$ then $\iota' \Downarrow \dot{v}$.*

Proof. A direct result from progress and preservation. □

We only prove semantic correspondence in one direction; again, whereas λ_{RS} proofs were detailed, here we provide a less verbosy proof.

Theorem 10 (Semantic Correspondence). *If $\iota \mapsto^* \iota'$ then $\iota \Downarrow \iota'$.*

Proof. By induction on the structure of ι. The proof is similar to the proof for λ_{RS}. □

if there's time add the proof...

Theorem 11 (Safety for λ_P). *If $\emptyset \vdash \iota : \tau$ then $\iota \Downarrow \dot{v}$ and $\emptyset \vdash \dot{v} : \tau$.*

Proof. If $\emptyset \vdash \iota : \tau$ then $\iota \mapsto^* \iota'$. Therefore, $\iota \Downarrow \iota'$ by part 2 of the semantic correspondence theorem. Since $\emptyset \vdash \iota : \tau$ and $\iota \mapsto^* \iota'$, it follows that $\emptyset \vdash \iota' : \tau$ by type safety for the small step semantics.

□

5 Proofs and Lemmas and Theorems About Translation

Theorem 12 (Translation Correctness). *If $\Psi \vdash e : \sigma$ then there exists an ι such that $[\![e]\!] = \iota$ and $[\![\Psi]\!] \vdash \iota : [\![\sigma]\!]$. Furthermore, $e \Downarrow v$ and $\iota \Downarrow \dot{v}$ such that $[\![v]\!] = \dot{v}$.*

Proof. We present a proof by induction on the structure of e. We write $e \rightsquigarrow \iota$ as shorthand for the final property.

Case $e = \mathsf{rstr}[s]$. Suppose $\Theta \vdash \mathsf{rstr}[s] : \sigma$.

By examination the syntactic structure of conclusions in the relation S-T, we know this is true just in case $\sigma = \mathsf{stringin}[r]$ for some r such that $s \in \mathcal{L}\{r\}$; and of course, there is always such an r.

There are no free variables in $\mathsf{rstr}[s]$, so we might as well proceed from the fact that $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$.

By definition of the translation ($[\![\cdot]\!]$) the following statements hold:

$$[\![\mathsf{rstr}[s]]\!] = \mathsf{str}[s] \tag{12}$$

$$[\![\mathsf{stringin}[r]]\!] = \mathsf{string} \tag{13}$$

$$[\![\emptyset]\!] = \emptyset \tag{14}$$

Note that $\emptyset \vdash \mathsf{str}[s] : \mathsf{string}$ by P-T-Str. Recall that contexts are standard and, in particular, can be weakened. So since $[\![\Theta]\!]$ is either a weakening of $\emptyset$ or $\emptyset$ itself, $[\![\Theta]\!] \vdash \mathsf{str}[s] : \mathsf{string}$ by weakening.

Summarily, $\mathsf{str}[s]$ is a term of λ_P such that $[\![\Theta]\!] \vdash \mathsf{str}[s] : [\![\sigma]\!]$

It remains to be shown that there exist $v, \dot{v}$ such that $\mathsf{rstr}[s] \Downarrow v$, $\mathsf{string}s \Downarrow \dot{v}$, and $[\![v]\!] = \dot{v}$. But this is immediate because each term evaluates to itself and we have already established the equality.

Case $e = \mathsf{rconcat}(e_1; e_2)$. By induction.

hand wave lots and lots of symbol pushing.

Case $e = \mathsf{rstrcase}(e_1; e_2; x, y.e_3)$. This case relies on our definition of context translation.

Suppose $\Psi \vdash \mathsf{rstrcase}(e_1; e_2; x, y.e_3) : \sigma$. By inversion of the typing relation it follows that $\Psi \vdash e_1 : \mathsf{stringin}[r]$, $\Psi \vdash e_2 : \sigma$ and $\Psi, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma$.

By induction, there exists an ι_1 such that $[\![e_1]\!] = \iota_1$, $[\![\Psi]\!] \vdash \iota_1 : [\![\sigma]\!]$, and $e_1 \rightsquigarrow \iota_1$. Similarly for e_2 and some ι_2.

By canonical forms, $e_1 \Downarrow \mathsf{rstr}[s]$ and so $\iota_1 \Downarrow \mathsf{str}[s]$ by $\rightsquigarrow$.

Choose $\iota = \mathsf{concat}(\iota_1; \iota_2)x, y.\iota_3$ and note that by the properties established via induction, $[\![e]\!] = \iota$ and $[\![\Psi]\!] \vdash \iota : [\![\sigma]\!]$.

Suppose $s = \epsilon$. Then $e \Downarrow v$ where $e_2 \Downarrow v$ and $\iota \Downarrow \dot{v}$ where $\iota_2 \Downarrow \dot{v}$. But recall that $e_2 \rightsquigarrow v_2$ and so $[\![v]\!] = \dot{v}$.

Suppose otherwise that $s = at$ for some character a and string t. Then $e \Downarrow v$ where $[a, t/x, y]e_3 \Downarrow v$. Similarly, $\iota \Downarrow \dot{v}$ where $[a, t/x, y]\iota_3 \Downarrow \dot{v}$

Case $e = \mathsf{rreplace}[r](e_1; e_2)$. There is only one applicable typing rule, so suppose $\Psi \vdash \mathsf{rreplace}[r](e_1; e_2) : \mathsf{stringin}[\mathtt{lreplace}(r, e_1, e_2)]$. Let $\psi = [\![\Psi]\!]$. Note that $[\![\mathsf{rreplace}[r](e_1; e_2)]\!] = \mathsf{replace}(\mathsf{rx}[r]; \iota_1; \iota_2)$ when by induction $[\![e_1]\!] = \iota_1$ and $[\![e_2]\!] = \iota_2$ such that $\psi \vdash \iota_1$ and $\psi \vdash \iota_2$. It follows by P-T-Replace that $\psi \vdash \mathsf{replace}(\mathsf{rx}[r]; \iota_1; \iota_2) : \mathsf{string}$. Finally, note that $[\![\mathsf{stringin}[\mathtt{lreplace}(r, e_1, e_2)]]\!] = \mathsf{string}$.

For evaluation correspondence, note that $[\![\mathsf{rstr}[\mathtt{lreplace}(r, s_1, s_2)]]\!] = \mathsf{rstr}[\mathtt{lreplace}(r, s_1, s_2)]$ and so it suffices to show that $\mathsf{replace}(\mathsf{rx}[r]; \iota_1; \iota_2) \Downarrow \mathsf{rstr}[r]s_1 s_2$. Note that $\mathtt{lreplace}(r, e_1, e_2) \Downarrow \mathsf{rstr}[\mathtt{lreplace}(r, s_1, s_2)]$ where $e_1 \Downarrow \mathsf{rstr}[s_1]$, $e_2 \Downarrow \mathsf{rstr}[s_2]$, $r \Downarrow r$. By induction, $\iota_1 \Downarrow \mathsf{rstr}[s_1], \iota_2 \Downarrow \mathsf{rstr}[s_2]$, and $\mathsf{rx}[r] \Downarrow \mathsf{rx}[r]$. So by S-E-Replace, the sufficient condition holds.

Case $e = \mathsf{rcoerce}[r](e')$. The only applicable tpying rule is S-T-SafeCoerce, so suppose $\Psi \vdash \mathsf{rcoerce}[r](e') : \mathsf{stringin}[r]$ where $\Psi \vdash e' : \mathsf{stringin}[r']$ and $\mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}$. By induction, $e' \rightsquigarrow \iota$ for some ι. Therefore, $[\![\mathsf{rcoerce}[r](e')]\!] = \iota$ by Tr-SafeCoerce.

For evaluation correspondence, note that $e \Downarrow v$ where $e' \Downarrow v$. The result follows by induction because $e' \rightsquigarrow \iota$.

Case $e = \mathsf{rcheck}[r](e_1; x.e_2; e_3)$. The applicable typing rule is S-T-Check, so $\psi \vdash e : \sigma$ where $\psi \vdash e_1 : \mathsf{stringin}[r]$, $\psi, x : \mathsf{stringin}[r] \vdash e_2 : \sigma$, and $\psi \vdash e_3 : \sigma$. By induction and a corresponding substitution principle there exists $\iota_1, \iota_2, \iota_3$ such that $e_1 \rightsquigarrow \iota_1$, $e_2 \rightsquigarrow \iota_2$ in context $\psi, s : \mathsf{stringin}[r]$, and $e_3 \rightsquigarrow \iota_3$. Choose $\iota = \mathsf{check}(\mathsf{rx}[r]; \iota_1; \lambda x.\iota_2; \iota_3)$. The result follows by induction. hand wave

$\square$

Theorem 13 (Correctness of Input Sanitation for Translated Terms)**.** *If* $[\![e]\!] = \iota$ *and* $\emptyset \vdash e : \mathsf{stringin}[r]$ *then* $\iota \Downarrow \mathsf{str}[s]$ *for* $s \in \mathcal{L}\{r\}$.

Proof. By Theorem 12 and the rules given, $\iota \Downarrow \mathsf{str}[s]$ implies that $e \Downarrow \mathsf{rstr}[s]$. Theorem 5 together with the assumption that e is well-typed implies that $s \in \mathcal{L}\{r\}$. $\square$

References

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

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

Figure 1: Regular expressions over the alphabet Σ.

$$\sigma \quad ::= \quad \sigma \to \sigma \mid \mathsf{stringin}[r] \qquad \text{source types}$$

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

$$v \quad ::= \quad \lambda x.e \mid \mathsf{rstr}[s] \qquad \text{source values}$$

Figure 2: Syntax of λ_{RS}.

$$\tau \quad ::= \quad \tau \to \tau \mid \mathsf{string} \mid \mathsf{regex} \qquad \text{target types}$$

$$\begin{array}{lll} \iota & ::= & x \mid \dot{v} \qquad \text{target terms} \\ & \mid & \mathsf{concat}(\iota; \iota) \mid \mathsf{strcase}(\iota; \iota; x, y.\iota) \\ & \mid & \mathsf{rx}[r] \mid \mathsf{replace}(\iota; \iota; \iota) \mid \mathsf{check}(\iota; \iota; \iota; \iota) \end{array}$$

$$\dot{v} \quad ::= \quad \lambda x.\iota \mid \mathsf{str}[s] \mid \mathsf{rx}[r] \qquad \text{target values}$$

Figure 3: Syntax for the target language, λ_P, containing strings and statically constructed regular expressions.

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

S-T-VAR
$$\frac{x : \sigma \in \Psi}{\Psi \vdash x : \sigma}$$

S-T-ABS
$$\frac{\Psi, x : \sigma_1 \vdash e : \sigma_2}{\Psi \vdash \lambda x.e : \sigma_1 \to \sigma_2}$$

S-T-APP
$$\frac{\Psi \vdash e_1 : \sigma_2 \to \sigma \qquad \Psi \vdash e_2 : \sigma_2}{\Psi \vdash e_1(e_2) : \sigma}$$

S-T-STRINGIN-I
$$\frac{s \in \mathcal{L}\{r\}}{\Psi \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]}$$

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

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

S-T-REPLACE
$$\frac{\Psi \vdash e_1 : \mathsf{stringin}[r_1] \qquad \Psi \vdash e_2 : \mathsf{stringin}[r_2]}{\Psi \vdash \mathsf{rreplace}[r](e_1; e_2) : \mathsf{stringin}[\mathtt{lreplace}(r, r_1, r_2)]}$$

S-T-SAFECOERCE
$$\frac{\Psi \vdash e : \mathsf{stringin}[r'] \qquad \mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}}{\Psi \vdash \mathsf{rcoerce}[r](e) : \mathsf{stringin}[r]}$$

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.

$\boxed{e \Downarrow v}$

$$\text{S-E-ABS}\quad \frac{}{\lambda x.e \Downarrow \lambda x.e}$$

$$\text{S-E-APP}\quad \frac{e_1 \Downarrow \lambda x.e_3 \qquad e_2 \Downarrow v_2 \qquad [v_2/x]e_3 \Downarrow v}{e_1(e_2) \Downarrow v}$$

$$\text{S-E-RSTR}\quad \frac{}{\mathsf{rstr}[s] \Downarrow \mathsf{rstr}[s]}$$

$$\text{S-E-CONCAT}\quad \frac{e_1 \Downarrow \mathsf{rstr}[s_1] \qquad e_2 \Downarrow \mathsf{rstr}[s_2]}{\mathsf{rconcat}(e_1; e_2) \Downarrow \mathsf{rstr}[s_1 s_2]}$$

$$\text{S-E-CASE-}\epsilon\quad \frac{e_1 \Downarrow \mathsf{rstr}[\epsilon] \qquad e_2 \Downarrow v_2}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \Downarrow v_2}$$

$$\text{S-E-CASE-CONCAT}\quad \frac{e_1 \Downarrow \mathsf{rstr}[as] \qquad [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \Downarrow v_3}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \Downarrow v_3}$$

$$\text{S-E-REPLACE}\quad \frac{e_1 \Downarrow \mathsf{rstr}[s_1] \qquad e_2 \Downarrow \mathsf{rstr}[s_2]}{\mathsf{rreplace}[r](e_1; e_2) \Downarrow \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]}$$

$$\text{S-E-SAFECOERCE}\quad \frac{e \Downarrow \mathsf{rstr}[s]}{\mathsf{rcoerce}[r](e) \Downarrow \mathsf{rstr}[s]}$$

$$\text{S-E-CHECK-OK}\quad \frac{e \Downarrow \mathsf{rstr}[s] \qquad s \in \mathcal{L}\{r\} \qquad [\mathsf{rstr}[s]/x]e_1 \Downarrow v}{\mathsf{rcheck}[r](e; x.e_1; e_2) \Downarrow v}$$

$$\text{S-E-CHECK-NOTOK}\quad \frac{e \Downarrow \mathsf{rstr}[s] \qquad s \notin \mathcal{L}\{r\} \qquad e_2 \Downarrow v}{\mathsf{rcheck}[r](e; x.e_1; e_2) \Downarrow v}$$

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

$\boxed{e \mapsto e}$

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

$$\text{SS-E-APPRIGHT}\quad \frac{e_2 \mapsto e_2'}{v_1 \mapsto v_1}$$

$$\text{SS-E-APPABS}\quad \frac{}{(\lambda x : \tau_{11}.t_{12})v_2 \mapsto [v_2/x]t_{12}}$$

$\boxed{e \mapsto^* e}$

$$\text{RT-REFL}\quad \frac{}{e \mapsto^* e}$$

$$\text{RT-TRANS}\quad \frac{e \mapsto^* e' \qquad e' \mapsto e''}{e \mapsto^* e''}$$

Figure 6: Call-by-name small step Semantics for λ and its reflexive, transitive closure.

$\boxed{e \mapsto e}$ (Contiues figure 6)

$$\frac{e_1 \mapsto e_1'}{\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)}\ \textsc{SS-E-Concat-Left}$$

$$\frac{e_2 \mapsto e_2'}{\mathsf{rconcat}(v_1; e_2) \mapsto \mathsf{rconcat}(v_1; e_2')}\ \textsc{SS-E-Concat-Right}$$

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

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

$$\frac{}{\mathsf{rstrcase}(\mathsf{rstr}[\epsilon]; e_2; x.y.e_3) \mapsto e_2}\ \textsc{SS-E-Case-}\epsilon\textsc{-val}$$

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

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

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

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

$$\frac{e \mapsto e'}{\mathsf{rcoerce}[r](e) \mapsto \mathsf{rcoerce}[r](e')}\ \textsc{SS-E-SafeCoerce-Step}$$

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

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

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

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

Figure 7: Small step semantics for λ_{RS}. Extends 6.

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

$$\textsc{P-T-Var}\ \frac{x : \tau \in \Theta}{\Theta \vdash x : \tau} \qquad \textsc{P-T-Abs}\ \frac{\Theta, x : \tau_1 \vdash \iota_2 : \tau_2}{\Theta \vdash \lambda x.\iota_2 : \tau_1 \to \tau_2} \qquad \textsc{P-T-App}\ \frac{\Theta \vdash \iota_1 : \tau_2 \to \tau \qquad \Theta \vdash \iota_2 : \tau_2}{\Theta \vdash \iota_1(\iota_2) : \iota} \qquad \textsc{P-T-String}\ \frac{}{\Theta \vdash \mathsf{str}[s] : \mathsf{string}}$$

$$\textsc{P-T-Regex}\ \frac{}{\Theta \vdash \mathsf{rx}[r] : \mathsf{regex}} \qquad \textsc{P-T-Concat}\ \frac{\Theta \vdash \iota_1 : \mathsf{string} \qquad \Theta \vdash \iota_2 : \mathsf{string}}{\Theta \vdash \mathsf{concat}(\iota_1; \iota_2) : \mathsf{string}}$$

$$\textsc{P-T-Case}\ \frac{\Theta \vdash \iota_1 : \mathsf{string} \qquad \Theta \vdash \iota_2 : \tau \qquad \Theta, x : \mathsf{string}, y : \mathsf{string} \vdash \iota_3 : \tau}{\Theta \vdash \mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3) : \tau}$$

$$\textsc{P-T-Replace}\ \frac{\Theta \vdash \iota_1 : \mathsf{regex} \qquad \Theta \vdash \iota_2 : \mathsf{string} \qquad \Theta \vdash \iota_3 : \mathsf{string}}{\Theta \vdash \mathsf{replace}(\iota_1; \iota_2; \iota_3) : \mathsf{string}}$$

$$\textsc{P-T-Check}\ \frac{\Theta \vdash \iota_x : \mathsf{regex} \qquad \Theta \vdash \iota_1 : \mathsf{string} \qquad \Theta \vdash \iota_2 : \tau \qquad \Theta \vdash \iota_3 : \tau}{\Theta \vdash \mathsf{check}(\iota_x; \iota_1; \iota_2; \iota_3) : \tau}$$

Figure 8: Typing rules for λ_P. The typing context Θ is standard.

$\boxed{\iota \Downarrow \dot{v}}$

$$\textsc{P-E-Abs}\ \frac{}{\lambda x.e \Downarrow \lambda x.e} \qquad \textsc{P-E-App}\ \frac{\iota_1 \Downarrow \lambda x.\iota_3 \qquad \iota_2 \Downarrow \dot{v}_2 \qquad [\dot{v}_2/x]\iota_3 \Downarrow \dot{v}_3}{\iota_1(\iota_2) \Downarrow \dot{v}_3} \qquad \textsc{P-E-Str}\ \frac{}{\mathsf{str}[s] \Downarrow \mathsf{str}[s]} \qquad \textsc{P-E-Rx}\ \frac{}{\mathsf{rx}[r] \Downarrow \mathsf{rx}[r]}$$

$$\textsc{P-E-Concat}\ \frac{\iota_1 \Downarrow \mathsf{str}[s_1] \qquad \iota_2 \Downarrow \mathsf{str}[s_2]}{\mathsf{concat}(\iota_1; \iota_2) \Downarrow \mathsf{str}[s_1 s_2]} \qquad \textsc{P-E-Case-}\epsilon\ \frac{\iota_1 \Downarrow \mathsf{str}[\epsilon] \qquad \iota_2 \Downarrow \dot{v_2}}{\mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3) \Downarrow \dot{v_2}} \qquad \textsc{P-E-Case-Concat}\ \frac{\iota_1 \Downarrow \mathsf{str}[as] \qquad [\mathsf{str}[a], \mathsf{str}[s]/x, y]\iota_3 \Downarrow \dot{v}}{\mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3) \Downarrow \dot{v}}$$

$$\textsc{P-E-Replace}\ \frac{\iota_1 \Downarrow \mathsf{rx}[r] \qquad \iota_2 \Downarrow \mathsf{str}[s_2] \qquad \iota_3 \Downarrow \mathsf{str}[s_3]}{\mathsf{replace}(\iota_1; \iota_2; \iota_3) \Downarrow \mathsf{str}[\mathsf{replace}(r; s_2; s_3)]} \qquad \textsc{P-E-Check-OK}\ \frac{\iota_x \Downarrow \mathsf{rx}[r] \qquad \iota \Downarrow \mathsf{str}[s] \qquad s \in \mathcal{L}\{r\} \qquad \iota_1 \Downarrow \dot{v_1}}{\mathsf{check}(\iota_x; \iota; \iota_1; \iota_2) \Downarrow \dot{v_1}}$$

$$\textsc{P-E-Check-NotOK}\ \frac{\iota_x \Downarrow \mathsf{rx}[r] \qquad \iota \Downarrow \mathsf{str}[s] \qquad s \notin \mathcal{L}\{r\} \qquad \iota_2 \Downarrow \dot{v_2}}{\mathsf{check}(\iota_x; \iota; \iota_1; \iota_2) \Downarrow \dot{v_2}}$$

Figure 9: Big step semantics for λ_P

.

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

PS-E-CONCATLEFT
$$\frac{\iota_1 \mapsto \iota_1'}{\mathsf{concat}(\iota_1; \iota_2) \Downarrow \mathsf{concat}(\iota_1'; \iota_2)}$$

PS-E-CONCATRIGHT
$$\frac{\iota_2 \mapsto \iota_2'}{\mathsf{concat}(\iota_1; \iota_2) \Downarrow \mathsf{concat}(\iota_1; \iota_2')}$$

PS-E-CONCAT
$$\frac{}{\mathsf{concat}(\mathsf{str}[s_1]; \mathsf{str}[s_2]) \Downarrow \mathsf{str}[s_1 s_2]}$$

PS-E-CASELEFT
$$\frac{\iota_1 \mapsto \iota_1'}{\mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3) \mapsto \mathsf{strcase}(\iota_1'; \iota_2; x, y.\iota_3)}$$

PS-E-CASE-EPSILON
$$\frac{}{\mathsf{strcase}(\epsilon; \iota_2; x, y.\iota_3) \mapsto \iota_2}$$

PS-E-CASE
$$\frac{}{\mathsf{strcase}(\mathsf{str}[as]; \iota_2; x, y.\iota_3) \mapsto \mathsf{str}[as]}$$

PS-E-REPLACELEFT
$$\frac{\iota_1 \mapsto \iota_1'}{\mathsf{replace}(\iota_1; \iota_2; \iota_3) \mapsto \mathsf{replace}(\iota_1'; \iota_2; \iota_3)}$$

PS-E-REPLACEMID
$$\frac{\iota_2 \mapsto \iota_2'}{\mathsf{replace}(\mathsf{rx}[r]; \iota_2; \iota_3) \mapsto \mathsf{replace}(\mathsf{rx}[r]; \iota_2'; \iota_3)}$$

PS-E-REPLACERIGHT
$$\frac{\iota_3 \mapsto \iota_3'}{\mathsf{replace}(\mathsf{rx}[r]; \mathsf{str}[s_2]; \iota_3) \mapsto \mathsf{replace}(\mathsf{rx}[r]; \mathsf{str}[s_2]; \iota_3')}$$

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

PS-E-CHECKLEFT
$$\frac{\iota_x \mapsto \iota_x'}{\mathsf{rcheck}[\iota_x](\iota; \iota_1; \iota_2) \mapsto \mathsf{rcheck}[\iota_x](\iota; \iota_1; \iota_2)}$$

PS-E-CHECKRIGHT
$$\frac{\iota \mapsto \iota'}{\mathsf{rcheck}[\mathsf{rx}[r]](\iota; \iota_1; \iota_2) \mapsto \mathsf{rcheck}[\mathsf{rx}[r]](\iota'; \iota_1; \iota_2)}$$

PS-E-CHECK-OK
$$\frac{s \in \mathcal{L}\{r\}}{\mathsf{rcheck}[\mathsf{rx}[r]](\mathsf{str}[s]; \iota_1; \iota_2) \mapsto \iota_1}$$

PS-E-CHECK-NOTOK
$$\frac{s \notin \mathcal{L}\{r\}}{\mathsf{rcheck}[\mathsf{rx}[r]](\mathsf{str}[s]; \iota_1; \iota_2) \mapsto \iota_2}$$

Figure 10: Small step semantics for λ_P (extends L-E rules)

.

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

$$\textsc{Tr-T-String}\quad \frac{}{[\![\mathsf{stringin}[r]]\!] = \mathsf{string}} \qquad \textsc{Tr-T-Arrow}\quad \frac{[\![\sigma_1]\!] = \tau_1 \qquad [\![\sigma_2]\!] = \tau_2}{[\![\sigma_1 \to \sigma_2]\!] = \tau_1 \to \tau_2}$$

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

$$\textsc{Tr-T-Context-Emp}\quad \frac{}{[\![\emptyset]\!] = \emptyset} \qquad \textsc{Tr-T-Context-Ext}\quad \frac{[\![\Psi]\!] = \Theta \qquad [\![\sigma]\!] = \tau}{[\![\Psi, x : \sigma]\!] = \Theta, x : \tau}$$

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

$$\textsc{Tr-Var}\quad \frac{}{[\![x]\!] = x} \qquad \textsc{Tr-Abs}\quad \frac{[\![e]\!] = \iota}{[\![\lambda x.e]\!] = \lambda x.\iota} \qquad \textsc{Tr-App}\quad \frac{[\![e_1]\!] = \iota_1 \qquad [\![e_2]\!] = \iota_2}{[\![e_1(e_2)]\!] = \iota_1(\iota_2)}$$

$$\textsc{Tr-Case}\quad \frac{[\![e_1]\!] = \iota_1 \qquad [\![e_2]\!] = \iota_2 \qquad [\![e_3]\!] = \iota_3}{[\![\mathsf{rstrcase}(e_1; e_2; x, y.e_3)]\!] = \mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3)}$$

$$\textsc{Tr-string}\quad \frac{}{[\![\mathsf{rstr}[s]]\!] = \mathsf{str}[s]} \qquad \textsc{Tr-Concat}\quad \frac{[\![e_1]\!] = \iota_1 \qquad [\![e_2]\!] = \iota_2}{[\![\mathsf{rconcat}(e_1; e_2)]\!] = \mathsf{concat}(\iota_1; \iota_2)}$$

$$\textsc{Tr-Subst}\quad \frac{[\![e_1]\!] = \iota_1 \qquad [\![e_2]\!] = \iota_2}{[\![\mathsf{rreplace}[r](e_1; e_2)]\!] = \mathsf{replace}(\mathsf{rx}[r]; \iota_1; \iota_2)}$$

$$\textsc{Tr-SafeCoerce}\quad \frac{[\![e]\!] = \iota}{[\![\mathsf{rcoerce}[r'](e)]\!] = \iota} \qquad \textsc{Tr-Check}\quad \frac{[\![e]\!] = \iota \qquad [\![e_1]\!] = \iota_1 \qquad [\![e_2]\!] = \iota_2}{[\![\mathsf{rcheck}[r](e; x.e_1; e_2)]\!] = \mathsf{check}(\mathsf{rx}[r]; \iota; (\lambda x.\iota_1)(\iota); \iota_2)}$$

Figure 11: Translation from source terms (e) to target terms (ι).