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

1 Terminology and Notation **2**

2 Regular Expressions **2**

3 λ_{RS} **2**

3.1 Head and Tail Operations . . . 2
3.2 Replacement . . . 3
3.3 Small Step Semantics of λ_{RS} . . . 3
3.3.1 The Security Theorem . . . 6

4 Proofs of Lemmas and Theorems About λ_P **6**

5 Proofs and Lemmas and Theorems About Translation **9**

List of Figures

1 Regular expressions over the alphabet Σ. . . . 13
2 Syntax of λ_{RS}. . . . 13
3 Syntax for the target language, λ_P, containing strings and statically constructed regular expressions. . . . 13
4 Typing rules for λ_{RS}. The typing context Ψ is standard. . . . 13
5 Call-by-name small step Semantics for λ and its reflexive, transitive closure. . . . 14
6 Small step semantics for λ_{RS}. Extends 5. . . . 14
7 Typing rules for λ_P. The typing context Θ is standard. . . . 15
8 Small step semantics for λ_P (extends L-E rules) . . . 16
9 Translation from source terms (e) to target terms (ι). . . . 17

1 Terminology and Notation

Theorems and lemmas appearing in [1] 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 [1].

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 $\texttt{lreplace}(r, r_1, r_2)$.

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

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

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

3.3 Small Step Semantics of λ_{RS}

Figure 6 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.

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 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 $\emptyset \vdash \mathsf{rstrcase}(e_1'; e_2; x, y.e_3) : \sigma$.

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

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 $\emptyset \vdash e : \mathsf{stringin}[\mathtt{lreplace}(r, r_1, r_2)]$ 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 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, $\mathtt{lreplace}(r, s_1, s_2) \in \mathcal{L}\{\mathtt{lreplace}(r, r_1, r_2)\}$ 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 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 $\emptyset \vdash \mathsf{rcoerce}[r](s_1) : \mathsf{stringin}[r]$. By Canonical Forms, $s \in \mathcal{L}\{r\}$. Therefore, $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$.

Case S-E-Check-Ok. Suppose $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto [\mathsf{rstr}[s]/x]e_1$, $s \in \mathcal{L}\{r\}$, 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 S-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 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 sfaetly, $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$. Therefore, $s \in \mathcal{L}\{r\}$. □

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 5 (Canonical Forms for Target Language)**.**

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

Theorem 6 (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 K (Substitution Lemma)**.** *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 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:

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

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:

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

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

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:

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

By induction, $\emptyset \vdash \iota_2' : \mathsf{string}$. 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:

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

By induction, $\emptyset \vdash \iota_3' : \mathsf{string}$. 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 $\emptyset \vdash \iota_3 : \tau$.

□

5 Proofs and Lemmas and Theorems About Translation

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

Proof. By induction on the typing relation.

Case S-T-Abs Suppose $\Psi \vdash \lambda x.e' : \sigma_1 \to \sigma_2$. By induction, $[\![\Psi x : \sigma_1]\!] \vdash [\![e']\!] : [\![\sigma_1]\!]$. By P-T-Abs, $[\![\Psi]\!] \vdash [\![\lambda x.e']\!] : [\![\sigma_1 \to \sigma_2]\!]$.

Case S-T-App Suppose $\Psi \vdash e_1(e_2) : \sigma$. By induction, $[\![\Psi]\!] \vdash [\![e_1]\!] : [\![\sigma_2 \to \sigma]\!]$ and $[\![\Psi]\!] \vdash [\![e_2]\!] : [\![\sigma_2]\!]$. By the definition of type translation, $[\![\sigma_2 \to \sigma]\!] = [\![\sigma_2]\!] \to [\![\sigma]\!]$. Therefore, $[\![\Psi]\!] \vdash [\![e_1(e_2)]\!] : [\![\sigma]\!]$.

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

Case S-T-Concat. Suppose $\Psi \vdash \mathsf{rconcat}(e_1; e_2) : \mathsf{stringin}[r_1 \cdot r_2]$. By induction, $[\![\Psi]\!] \vdash [\![e_1]\!] : [\![\mathsf{stringin}[r_1]]\!]$ and $[\![\Psi]\!] \vdash [\![e_2]\!] : [\![\mathsf{stringin}[r_2]]\!]$. The result follows by P-T-Concat and the definition of term and type translation.

Case S-T-Case. Suppose $\Psi \vdash \mathsf{rstrcase}(e_1; e_2; x, y.e_3) : \sigma$. By induction, $\llbracket\Psi\rrbracket \vdash \llbracket e_1\rrbracket : \llbracket\mathsf{stringin}[r]\rrbracket$, $\llbracket\Psi\rrbracket \vdash \llbracket e_2\rrbracket : \llbracket\sigma\rrbracket$, and $\llbracket\Psi, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)]\rrbracket \vdash \llbracket e_3\rrbracket : \llbracket\sigma\rrbracket$. By the definition of context and type translation, $\llbracket\Psi, x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)]\rrbracket = \llbracket\Psi\rrbracket, x : \mathsf{string}, y :$ string. The result follows by P-T-Case.

Case S-T-Replace. Suppoes $\Psi \vdash \mathsf{rreplace}[r](e_1; e_2) : \mathsf{stringin}[\mathtt{lreplace}(r, r_1, r_2)]$. Note that $\llbracket\mathsf{rreplace}[r](e_1; e_2)\rrbracket = \mathsf{replace}(\mathsf{rx}[r]; \llbracket e_1\rrbracket; \llbracket e_2\rrbracket)$. Let $\Theta = \llbracket\Psi\rrbracket$. By induction, $\Theta \vdash \llbracket e_1\rrbracket : \llbracket\mathsf{stringin}[r_1]\rrbracket$ and $\Theta \vdash \llbracket e_2\rrbracket : \llbracket\mathsf{stringin}[r_2]\rrbracket$ from which it follows by the definition of type translation that $\Theta \vdash \llbracket e_1\rrbracket :$ string and $\Theta \vdash \llbracket e_2\rrbracket :$ string. So P-T-Replace, $\Theta \vdash \mathsf{replace}(\mathsf{rx}[r]; \llbracket e_1\rrbracket; \llbracket e_2\rrbracket) : \mathsf{string}$

Case S-T-SafeCoerce. Suppose $\Psi \vdash \mathsf{rcoerce}[r](e) : \mathsf{stringin}[r]$ where $\Psi \vdash e : \mathsf{stringin}[r']$ and $\mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}$. Let $\llbracket\Psi\rrbracket = \Theta$. Note that $\llbracket\mathsf{rcoerce}[r](e)\rrbracket = \llbracket e\rrbracket$. By induction, $\Theta \vdash \llbracket e\rrbracket : \llbracket\mathsf{stringin}[r']\rrbracket$.

Case S-T-Check. Suppose $\Psi \vdash \mathsf{rcheck}[r](e_0; x.e_1; e_2) : \sigma$ where $\Psi \vdash e_0 : \mathsf{stringin}[r]$, $\Psi, x : \mathsf{stringin}[r] \vdash e_1 : \sigma$, and $\Psi \vdash e_2 : \sigma$.

Note that $\llbracket\ \mathsf{rcheck}[r](e_0; x.e_1; e_2)\rrbracket = \mathsf{check}(\mathsf{rx}[r]\llbracket e_0\rrbracket\lambda x.\llbracket e_1\rrbracket(\llbracket e_0\rrbracket)\llbracket e_2\rrbracket;;\ .; \mathrm{L})$et $\llbracket\Psi\rrbracket = \Theta$. Note that $\llbracket r\rrbracket = \mathsf{rx}[r]$ and by canonical forms $\Theta \vdash \mathsf{rx}[r] : \mathsf{regex}$. By induction, $\Theta \vdash \llbracket e_0\rrbracket : \llbracket\mathsf{stringin}[r]\rrbracket$ and so $\Theta \vdash \llbracket e_0\rrbracket : \mathsf{string}$. By context translation, $\llbracket\Psi, x : \mathsf{stringin}[r]\rrbracket = \Theta, x : \mathsf{string}$ and so by induction $\Theta, x : \mathsf{string} \vdash \llbracket e_1\rrbracket : \llbracket\sigma\rrbracket$. Therefore, $\Theta \vdash \lambda x.\llbracket e_1\rrbracket(\llbracket e_0\rrbracket) : \llbracket\sigma\rrbracket$. By induction, $\Theta \vdash \llbracket e_0\rrbracket : \llbracket\sigma\rrbracket$. From these three facts, it follows that $\Theta \vdash \mathsf{check}(\mathsf{rx}[r]\llbracket e_0\rrbracket\lambda x.\llbracket e_1\rrbracket(\llbracket e_0\rrbracket)\llbracket e_2\rrbracket; :; \llbracket; \rrbracket\sigma)$.

□

Theorem 9 (Translation Correctness). *If $\Psi \vdash e : \sigma$ and $e \mapsto e'$ then $\llbracket e\rrbracket \mapsto^* \llbracket e'\rrbracket$.*

Proof. By induction on the typing and evaluation derivations. □

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

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

This proof needs to be changed to use only the small-step semantics.

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 ($\llbracket\cdot\rrbracket$) the following statements hold:

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

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

$$\llbracket\emptyset\rrbracket = \emptyset \tag{14}$$

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 $\llbracket\Theta\rrbracket$ is either a weakening of $\emptyset$ or $\emptyset$ itself, $\llbracket\Theta\rrbracket \vdash \mathsf{str}[s] : \mathsf{string}$ by weakening.

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

It remains to be shown that there exist $v, \dot{v}$ such that $\mathsf{rstr}[s] \mapsto^* v$, $\mathsf{str}[s] \mapsto^* \dot{v}$, and $[\![v]\!] = \dot{v}$. But this is immediate because each term is already a value and $s = s$.

Case $e = \mathsf{rconcat}(e_1; e_2)$. The applicable typing rule is S-T-Concat, so $\Psi \vdash \mathsf{rconcat}(e_1; e_2) : \mathsf{stringin}[r_1 \cdot r_2]$ where $\Psi \vdash e_1 : \mathsf{stringin}[r_1]$ and $\Psi \vdash e_2 : \mathsf{stringin}[r_2]$.

By induction, $e_1 \rightsquigarrow \iota_1$ and $e_2 \rightsquigarrow \iota_2$. Therefore, $[\![\Psi]\!] \vdash \mathsf{concat}(\iota_1; \iota_2)$ by P-T-Concat.

By canonical forms, $e_1 \mapsto^* \mathsf{rstr}[s_1]$ where by induction $\iota_1 \mapsto^* \mathsf{str}[s_1]$. Similarly, $e_2 \mapsto^* \mathsf{rstr}[s_2]$ and $\iota_2 \mapsto^* \mathsf{str}[s_2]$. Therefore, $e \mapsto^* \mathsf{rstr}[s_1 s_2]$ by S-E-Concat at last, and $\mathsf{concat}(\iota_1; \iota_2) \mapsto^* \mathsf{str}[s_1 s_2]$ by P-E-Concat at last. Note that $[\![\mathsf{rstr}[s_1 s_2]]\!] = \mathsf{str}[s_1 s_2]$.

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 $e_1 \mapsto \iota_1$.

By canonical forms, $e_1 \mapsto^* \mathsf{rstr}[s]$. Therefore, $\iota_1 \mapsto *\mathsf{str}[s]$ because $e_1 \rightsquigarrow \iota_1$.

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

To prove the evaluation correspondence, we consider two cases for the value of s.

Suppose $s = \epsilon$. Then $e \mapsto^* v$ where $e_2 \mapsto^* v$, from which it follows that $\iota \mapsto^* \dot{v}$ where $\iota_2 \mapsto^* \dot{v}$. 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 $e \mapsto^* v$ where $[a, t/x, y]e_3 \mapsto^* v$. Similarly, $\iota \mapsto^* \dot{v}$ where $[a, t/x, y]\iota_3 \mapsto^* \dot{v}$

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 $\Theta = [\![\Psi]\!]$. Note that $[\![\mathsf{rreplace}[r](e_1; e_2)]\!] = \mathsf{replace}(\mathsf{rx}[r]; \iota_1; \iota_2)$ where by induction $[\![e_1]\!] = \iota_1$ and $[\![e_2]\!] = \iota_2$ such that $\Theta \vdash \iota_1$ and $\Theta \vdash \iota_2$. It follows by P-T-Replace that $\Theta \vdash \mathsf{replace}(\mathsf{rx}[r]; \iota_1; \iota_2) : \mathsf{string}$. 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 $\mathsf{replace}(\mathsf{rx}[r]; \iota_1; \iota_2) \mapsto^* \mathsf{rstr}[r]s_1 s_2$. Note that $\mathtt{lreplace}(r, e_1, e_2) \mapsto^* \mathsf{rstr}[\mathtt{lreplace}(r, s_1, s_2)]$ where $e_1 \mapsto^* \mathsf{rstr}[s_1]$, $e_2 \mapsto^* \mathsf{rstr}[s_2]$, $r \mapsto^* r$. By induction, $\iota_1 \mapsto^* \mathsf{rstr}[s_1]$, $\iota_2 \mapsto^* \mathsf{rstr}[s_2]$, and $\mathsf{rx}[r] \mapsto^* \mathsf{rx}[r]$. 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 $e \mapsto^* v$ where $e' \mapsto^* v$. 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.

□

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 4, $e \mapsto^* \mathsf{rstr}[s]$ 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 10, $\iota \mapsto^* \mathsf{str}[s]$. $\square$

References

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

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

Figure 1: Regular expressions over the alphabet Σ.

σ	$::=$	$\sigma \to \sigma \mid \mathsf{stringin}[r]$	source types
e	$::=$	$x \mid v$	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	$::=$	$\lambda x.e \mid \mathsf{rstr}[s]$	source values

Figure 2: Syntax of λ_{RS}.

τ	$::=$	$\tau \to \tau \mid \mathsf{string} \mid \mathsf{regex}$	target types
ι	$::=$	$x \mid \dot{v}$	target terms
	$\mid$	$\mathsf{concat}(\iota; \iota) \mid \mathsf{strcase}(\iota; \iota; x, y.\iota)$	
	$\mid$	$\mathsf{rx}[r] \mid \mathsf{replace}(\iota; \iota; \iota) \mid \mathsf{check}(\iota; \iota; \iota; \iota)$	
$\dot{v}$	$::=$	$\lambda x.\iota \mid \mathsf{str}[s] \mid \mathsf{rx}[r]$	target values

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

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

$$\text{S-T-VAR}\ \frac{x : \sigma \in \Psi}{\Psi \vdash x : \sigma} \qquad \text{S-T-ABS}\ \frac{\Psi, x : \sigma_1 \vdash e : \sigma_2}{\Psi \vdash \lambda x.e : \sigma_1 \to \sigma_2} \qquad \text{S-T-APP}\ \frac{\Psi \vdash e_1 : \sigma_2 \to \sigma \qquad \Psi \vdash e_2 : \sigma_2}{\Psi \vdash e_1(e_2) : \sigma} \qquad \text{S-T-STRINGIN-I}\ \frac{s \in \mathcal{L}\{r\}}{\Psi \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]}$$

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

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

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

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

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

$\boxed{e \mapsto e}$

$$\frac{e_1 \mapsto e_1'}{e_1(e_2) \mapsto e_1'(e_2)}\ \textsc{SS-E-AppLeft} \qquad \frac{e_2 \mapsto e_2'}{v_1 \mapsto v_1}\ \textsc{SS-E-AppRight} \qquad \frac{}{(\lambda x : \tau_{11}.t_{12})v_2 \mapsto [v_2/x]t_{12}}\ \textsc{SS-E-AppAbs}$$

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

$$\frac{}{e \mapsto^* e}\ \textsc{RT-Refl} \qquad \frac{e \mapsto^* e' \qquad e' \mapsto e''}{e \mapsto^* e''}\ \textsc{RT-Trans}$$

Figure 5: Call-by-name small step Semantics for λ and its reflexive, transitive closure.

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

$$\frac{e_1 \mapsto e_1'}{\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)}\ \textsc{SS-E-Concat-Left} \qquad \frac{e_2 \mapsto e_2'}{\mathsf{rconcat}(v_1; e_2) \mapsto \mathsf{rconcat}(v_1; e_2')}\ \textsc{SS-E-Concat-Right}$$

$$\frac{}{\mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]}\ \textsc{SS-E-Concat} \qquad \frac{e_1 \mapsto e_1'}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \mapsto \mathsf{rstrcase}(e_1'; e_2; x, y.e_3)}\ \textsc{SS-E-Case-Left}$$

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

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

$$\frac{}{\mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[\mathsf{replace}(r; s_1; s_2)]}\ \textsc{SS-E-Replace} \qquad \frac{e \mapsto e'}{\mathsf{rcoerce}[r](e) \mapsto \mathsf{rcoerce}[r](e')}\ \textsc{SS-E-SafeCoerce-Step}$$

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

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

Figure 6: Small step semantics for λ_{RS}. Extends 5.

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

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

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

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

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

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

Figure 7: Typing rules for λ_P. The typing context Θ is standard.

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

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

$$\textsc{PS-E-ConcatRight}\quad \frac{\iota_2 \mapsto \iota_2'}{\mathsf{concat}(\iota_1; \iota_2) \mapsto \mathsf{concat}(\iota_1; \iota_2')}$$

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

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

$$\textsc{PS-E-Case-Epsilon}\quad \frac{}{\mathsf{strcase}(\epsilon; \iota_2; x, y.\iota_3) \mapsto \iota_2}$$

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

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

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

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

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

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

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

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

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

Figure 8: Small step semantics for λ_P (extends L-E rules)

.

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

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

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

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

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

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

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

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

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

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

Figure 9: Translation from source terms (e) to target terms (ι).