

# **Statically Typed String Sanitation Inside a Python (Technical Report)**

**Nathan Fulton      Cyrus Omar      Jonathan Aldrich**

December 2014  
CMU-ISR-14-112

School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

## **Abstract**

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

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

## **Contents**

## **List of Figures**

# 1 Terminology and Notation

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

## 2 Regular Expressions

The syntax of regular expressions over some alphabet  $\Sigma$  is shown in Figure ??.

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

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

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

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

## 3 $\lambda_{RS}$

The syntax of  $\lambda_{RS}$  is specified in Figure ?. The static semantics is specified in Figure ?.

### 3.1 Head and Tail Operations

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

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

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

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  $\text{lhead}(r)$ ). 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:

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

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

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

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

### 3.2 Replacement

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

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

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

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

### 3.3 Small Step Semantics of $\lambda_{RS}$

Figure ?? specifies a small-step operational semantics for  $\lambda_{RS}$ .

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

1. If  $\sigma = \text{stringin}[r]$  then  $v = \text{rstr}[s]$  and  $s \in \mathcal{L}\{r\}$ .
2. If  $\sigma = \sigma_1 \rightarrow \sigma_2$  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  $\emptyset \vdash e : \sigma$ .

**$\lambda$  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  $\emptyset \vdash \text{rstr}[s] : \text{stringin}[s]$ . Then  $e = \text{rstr}[s]$ .

**S-T-Concat.** Suppose  $\emptyset \vdash \text{rconcat}(e_1; e_2) : \text{stringin}[r_1 \cdot r_2]$  and  $\emptyset \vdash e_1 : \text{stringin}[r_1]$  and  $\emptyset \vdash e_2 : \text{stringin}[r_2]$ . 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  $\text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e'_1; e_2)$ . 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  $e_1 = \text{rstr}[s_1]$  and  $e_2 = \text{rstr}[s_2]$ . Finally, by SS-E-Concat,  $\text{rconcat}(\text{rstr}[s_1]; \text{rstr}[s_2]) \mapsto \text{rstr}[s_1 s_2]$ .

**S-T-Case.** Suppose  $e = \text{rstrcase}(e_1; e_2; x, y.e_3)$  and  $\emptyset \vdash e_1 : \text{stringin}[r]$ . By induction and Canonical Forms it follows that  $e_1 \mapsto e'_1$  or  $e_1 = \text{rstr}[s]$ . 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- $\epsilon$ -Val, and if  $s = at$  then  $e$  steps by S-E-Case-Concat.

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

- (1)  $\emptyset \vdash e_1 : \text{stringin}[r_1]$
- (2)  $\emptyset \vdash e_2 : \text{stringin}[r_2]$

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,  $e_1 = \text{rstr}[s_1]$  and  $e_2 = \text{rstr}[s_2]$ . Therefore,  $e \mapsto \text{rstr}[\text{replace}(r; s_1; s_2)]$  by SS-E-Replace.

**S-T-SafeCoerce.** Suppose that  $\emptyset \vdash \text{rcoerce}[r](e_1) : \text{stringin}[r]$ . and  $\emptyset \vdash e_1 : \text{stringin}[r']$  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  $e_1 = \text{rstr}[s]$ . In this case,  $e = \text{rcoerce}[r](\text{rstr}[s]) \mapsto \text{rstr}[s]$  by SS-E-SafeCoerce.

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

- (3)  $\emptyset \vdash e_0 : \text{stringin}[r_0]$
- (4)  $\emptyset, x : \text{stringin}[r] \vdash e_1 : \sigma$
- (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,  $e_0 = \text{rstr}[s]$  by Canonical Forms. By Lemma ?? part 2, either  $s \in \mathcal{L}\{r_0\}$  or  $s \notin \mathcal{L}\{r_0\}$ . In the former case  $e$  takes a step by SS-E-Check-Ok. In the latter case  $e$  takes a step by SS-E-Check-NotOk.

□

**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  $\emptyset \vdash e : \sigma$ .

**$\lambda$  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  $e = \text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e'_1; e_2)$  and  $e_1 \mapsto e'_1$ . The only rule that applies is S-T-Concat, so  $\emptyset \vdash e_1 : \text{stringin}[r_1]$  and  $\emptyset \vdash e_2 : \text{stringin}[r_2]$ . By induction,  $\emptyset \vdash e'_1 : \text{stringin}[r_1]$ . Therefore, by S-T-Concat,  $\emptyset \vdash \text{rconcat}(e'_1; e_2) : \text{stringin}[r_1 r_2]$ .

**S-E-Concat-Right.** Suppose  $e = \text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e_1; e'_2)$  and  $e_2 \mapsto e'_2$ . The only rule that applies is S-T-Concat, so  $\emptyset \vdash e_1 : \text{stringin}[r_1]$  and  $\emptyset \vdash e_2 : \text{stringin}[r_2]$ . By induction,  $\emptyset \vdash e'_2 : \text{stringin}[r_2]$ . Therefore, by S-T-Concat,  $\emptyset \vdash \text{rconcat}(e_1; e'_2) : \text{stringin}[r_1 r_2]$ .

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

**S-E-Case-Left.** Suppose  $e \mapsto \text{rstrcase}(e'_1; e_2; x, y.e_3)$  and  $\emptyset \vdash e : \sigma$  and  $e_1 \mapsto e'_1$ . The only rule that applies is S-T-Case, so:

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

By (??) and the assumption that  $e_1 \mapsto e'_1$ , it follows by induction that  $\emptyset \vdash e'_1 : \text{stringin}[r]$ . This fact together with (??) and (??) implies by S-T-Case that  $\emptyset \vdash \text{rstrcase}(e'_1; e_2; x, y.e_3) : \sigma$ .

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

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

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

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

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

**Case S-E-Replace-Left.** Suppose that  $e = \text{rreplace}[r](e_1; e_2) \mapsto \text{rreplace}[r](e'_1; e_2)$  when  $e_1 \mapsto e'_1$ . The only rule that applies is S-T-Replace, so  $\emptyset \vdash e : \text{stringin}[\text{lreplace}(r, r_1, r_2)]$  where:

$$\begin{aligned} \emptyset \vdash e_1 &: \text{stringin}[r_1] \\ \emptyset \vdash e_2 &: \text{stringin}[r_2] \end{aligned}$$

By induction,  $\emptyset \vdash e'_1 : \text{stringin}[r_1]$ . Therefore,  $\emptyset \vdash \text{rreplace}[r](e'_1; e_2) : \text{stringin}[\text{lreplace}(r, r_1, r_2)]$  by S-T-Replace.

Cyrus  
stopped  
here

**Case S-E-Replace-Right.** Suppose that  $e = \text{rreplace}[r](e_1; e_2) \mapsto \text{rreplace}[r](e'_1; e_2)$  when  $e_1 \mapsto e'_1$ . The only rule that applies is S-T-Replace, so  $\emptyset \vdash e : \text{stringin}[\text{lreplace}(r, r_1, r_2)]$  where:

$$\begin{aligned}\emptyset \vdash e_1 &: \text{stringin}[r_1] \\ \emptyset \vdash e_2 &: \text{stringin}[r_2]\end{aligned}$$

By induction,  $\emptyset \vdash e'_1 : \text{stringin}[r_1]$ . Therefore,  $\emptyset \vdash \text{rreplace}[r](e'_1; e_2) : \text{stringin}[\text{lreplace}(r, r_1, r_2)]$  by S-T-Replace.

**Case S-E-Replace.**

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

$$\begin{aligned}\emptyset \vdash \text{rstr}[s_1] &: \text{stringin}[r_1] \\ \emptyset \vdash \text{rstr}[s_2] &: \text{stringin}[r_2]\end{aligned}$$

By conanical forms,  $s_1 \in \mathcal{L}\{r_1\}$  and  $s_2 \in \mathcal{L}\{r_2\}$ . Therefore,  $\text{lreplace}(r, s_1, s_2) \in \mathcal{L}\{\text{lreplace}(r, r_1, r_2)\}$  by Theorem ?? . It is finally derivable by S-T-Rstr that:

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

**Case S-E-SafeCoerce.** Suppose that  $\text{rcoerce}[r](\text{rstr}[s_1]) \mapsto \text{rstr}[s_1]$ . The only applicable rule is S-T-SafeCoerce, so  $\emptyset \vdash \text{rcoerce}[r](s_1) : \text{stringin}[r]$ . By Canonical Forms,  $s \in \mathcal{L}\{r\}$ . Therefore,  $\emptyset \vdash \text{rstr}[s] : \text{stringin}[r]$ .

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

**Case S-E-Check-NotOk.** Suppose  $\text{rcheck}[r](\text{rstr}[s]; x.e_1; e_2) \mapsto e_2$ ,  $s \notin \mathcal{L}\{r\}$ , and  $\emptyset \vdash \text{rcheck}[r](\text{rstr}[s]; x.e_1; e_2) : \sigma$ . 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  $e \text{ val}$  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  $\lambda_{RS}$ ). *If  $\emptyset \vdash e : \text{stringin}[r]$  and  $e \mapsto^* \text{rstr}[s]$  then  $s \in \mathcal{L}\{r\}$ .*

*Proof.* By type safety,  $\emptyset \vdash \text{rstr}[s] : \text{stringin}[r]$ . Therefore,  $s \in \mathcal{L}\{r\}$ . □

## 4 Proofs of Lemmas and Theorems About $\lambda_P$

This section follows the same structure as the safety proof for  $\lambda_{RS}$  – 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  $\emptyset \vdash \iota : \text{regex}$  then  $\iota \mapsto^* \text{rx}[r]$  such that  $r$  is a well-formed regular expression.
- If  $\emptyset \vdash \iota : \text{string}$  then  $\iota \mapsto^* \text{str}[s]$ .

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

*Proof.* The proof proceeds by induction on the typing assumption. Consider only the string and regex (non- $\lambda$ ) fragments of  $\lambda_P$ .

**P-T-Case.** Suppose  $\emptyset \vdash \text{strcase}(\iota_1; \iota_2; x, y, \iota_3)$ . By inversion,  $\iota_1 : \text{string}$  and so either  $\iota_1 \mapsto \iota'_1$  or by canonical forms,  $\iota_1 = \text{str}[s_1]$ . Similarly,  $\iota_2 \mapsto \iota'_2$  or else  $\iota_2 = \text{str}[s_2]$ . 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  $\emptyset \vdash \text{replace}(\iota_1; \iota_2; \iota_3)$ . By inversion,  $\iota_1 : \text{regex}$  and so by canonical forms  $\iota_1 = \text{rx}[r]$ . By inversion,  $\iota_2 : \text{string}$  and so by induction either  $\iota_2 \mapsto \iota'_2$  or else  $\iota_2 = \text{str}[s_2]$  for some string  $s_2$ . Similarly, either  $\iota_3$  steps or else  $\iota_3 = \text{str}[s_3]$ . 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  $\iota_x$  and  $\iota_1$  implies that the term at hand equals  $\text{rx}[r]\text{str}[s]\iota_2\iota_3$ , which evaluates to either  $\iota_2$  or  $\iota_3$ .

□

**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:

$$\begin{aligned} \iota &= \text{rconcat}(\iota_1; \iota_2) \mapsto \text{rconcat}(\iota'_1; \iota_2) \\ \emptyset \vdash \iota : \text{string} \\ \iota &\mapsto \iota' \end{aligned}$$

The only applicable typing rule is P-T-Concat, so  $\emptyset \vdash \iota_1 : \text{string}$  and  $\emptyset \vdash \iota_2 : \text{string}$ . By induction,  $\emptyset \vdash \iota'_1 : \text{string}$ , so  $\emptyset \vdash \text{rconcat}(\iota'_1; \iota_2) : \text{string}$ .

**Case PS-E-ConcatRight**

$$\begin{aligned} e &= \text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e_1; e'_2) \\ \emptyset \vdash e : \text{string} \\ \iota &\mapsto \iota' \end{aligned}$$

The only applicable typing rule is P-T-Concat, so  $\emptyset \vdash \iota_1 : \text{string}$  and  $\emptyset \vdash \iota_2 : \text{string}$ . By induction,  $\emptyset \vdash \iota'_1 : \text{string}$ , so  $\emptyset \vdash \text{rconcat}(\iota_1; \iota'_2) : \text{string}$ .

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

**Case PS-E-CaseLeft** Let  $\iota = \text{rstrcase}(\iota_1; \iota_2; x, y, \iota_3) \mapsto \text{rstrcase}(\iota'_1; \iota_2; x, y, \iota_3)$  when  $\iota_1 \mapsto \iota'_1$ . The only rule that applies is P-T-Case, so  $\emptyset \vdash \iota : \tau$  where:

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

By induction,  $\emptyset \vdash \iota'_1 : \text{string}$ . By P-T-Case,  $\emptyset \vdash \text{rstrcase}(\iota_1; \iota_2; x, y, \iota_3) : \tau$ .

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

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

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

The result follows by the substitution lemma.

**Case PS-E-ReplaceLeft** Let  $\iota = \text{rreplace}[\iota_1](\iota_2; \iota_3) \mapsto \text{rreplace}[\iota'_1](\iota_2; \iota_3)$  where  $\iota_1 \mapsto \iota'_1$ . The applicable typing rule is P-T-Replace, so  $\emptyset \vdash \iota : \text{string}$  where:

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

By induction,  $\emptyset \vdash \iota'_1 : \text{regex}$ . Therefore,  $\emptyset \vdash \text{rreplace}[\iota'_1](\iota_2; \iota_3)$ .

**Case PS-E-ReplaceMid** Let  $\iota = \text{rreplace}[\iota_1](\iota_2; \iota_3) \mapsto \text{rreplace}[\iota_1](\iota'_2; \iota_3)$  where  $\iota_2 \mapsto \iota'_2$ . The applicable typing rule is P-T-Replace, so  $\emptyset \vdash \iota : \text{string}$  where:

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

By induction,  $\emptyset \vdash \iota'_2 : \text{string}$ . Therefore,  $\emptyset \vdash \text{rreplace}[\iota_1](\iota'_2; \iota_3)$ .

**Case PS-E-ReplaceRight** Let  $\iota = \text{rreplace}[\iota_1](\iota_2; \iota_3) \mapsto \text{rreplace}[\iota_1](\iota_2; \iota'_3)$  where  $\iota_3 \mapsto \iota'_3$ . The applicable typing rule is P-T-Replace, so  $\emptyset \vdash \iota : \text{string}$  where:

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

By induction,  $\emptyset \vdash \iota'_3 : \text{string}$ . Therefore,  $\emptyset \vdash \text{rreplace}[\iota_1](\iota_2; \iota'_3)$ .

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

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

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

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

**Case PS-E-CheckRight** Let  $\iota = \text{rcheck}[\iota_x](\iota_1; \iota_2; \iota_3) \mapsto \text{rcheck}[\iota_x](\iota'_1; \iota_2; \iota_3)$  where  $\iota_1 \mapsto \iota'_1$ . The applicable typing rule is P-T-Check, so  $\emptyset \vdash \iota : \tau$  where:

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

By induction,  $\iota'_1 : \text{string}$ . Therefore,  $\emptyset \text{rcheck}[\iota_x](\iota'_1; \iota_2; \iota_3) : \tau$ .

**Case PS-E-Check-Ok** Let  $\iota = \text{rcheck}[\text{rx}[r]](\text{rstr}[s]; \iota_2; \iota_3) \mapsto \iota_2$  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  $\iota = \text{rcheck}[\text{rx}[r]](\text{rstr}[s]; \iota_2; \iota_3) \mapsto \iota_3$  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  $\llbracket \Psi \rrbracket \vdash \iota : \llbracket \sigma \rrbracket$*

*Proof.* By induction on the typing relation.

**Case S-T-Abs** Suppose  $\Psi \vdash \lambda x. e' : \sigma_1 \rightarrow \sigma_2$ . By induction,  $\llbracket \Psi x : \sigma_1 \rrbracket \vdash \llbracket e' \rrbracket : \llbracket \sigma_1 \rrbracket$ . By P-T-Abs,  $\llbracket \Psi \rrbracket \vdash \llbracket \lambda x. e' \rrbracket : \llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket$ .

**Case S-T-App** Suppose  $\Psi \vdash e_1(e_2) : \sigma$ . By induction,  $\llbracket \Psi \rrbracket \vdash \llbracket e_1 \rrbracket : \llbracket \sigma_2 \rightarrow \sigma \rrbracket$  and  $\llbracket \Psi \rrbracket \vdash \llbracket e_2 \rrbracket : \llbracket \sigma_2 \rrbracket$ . By the definition of type translation,  $\llbracket \sigma_2 \rightarrow \sigma \rrbracket = \llbracket \sigma_2 \rrbracket \rightarrow \llbracket \sigma \rrbracket$ . Therefore,  $\llbracket \Psi \rrbracket \vdash \llbracket e_1(e_2) \rrbracket : \llbracket \sigma \rrbracket$ .

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

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

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

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

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

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

Note that  $\llbracket \text{rcheck}[r](e_0; x, e_1; e_2) \rrbracket = \text{check}(\text{rx}[r]\llbracket e_0 \rrbracket \lambda x. \llbracket e_1 \rrbracket(\llbracket e_0 \rrbracket)\llbracket e_2 \rrbracket;; \cdot; \cdot) \text{et } \llbracket \Psi \rrbracket = \Theta$ . Note that  $\llbracket r \rrbracket = \text{rx}[r]$  and by canonical forms  $\Theta \vdash \text{rx}[r] : \text{regex}$ . By induction,  $\Theta \vdash \llbracket e_0 \rrbracket : \llbracket \text{stringin}[r] \rrbracket$  and so  $\Theta \vdash \llbracket e_0 \rrbracket : \text{string}$ . By context translation,  $\llbracket \Psi, x : \text{stringin}[r] \rrbracket = \Theta, x : \text{string}$  and so by induction  $\Theta, x : \text{string} \vdash \llbracket e_1 \rrbracket : \llbracket \sigma \rrbracket$ . 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  $\Theta \vdash \text{check}(\text{rx}[r]\llbracket e_0 \rrbracket \lambda x. \llbracket e_1 \rrbracket(\llbracket e_0 \rrbracket)\llbracket e_2 \rrbracket;; \cdot; \cdot) : \llbracket \sigma \rrbracket$ .

□

**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  $\iota$  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.

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

By examination the syntactic structure of conclusions in the relation S-T, we know this is true just in case  $\sigma = \text{stringin}[r]$  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  $\text{rstr}[s]$ , so we might as well proceed from the fact that  $\emptyset \vdash \text{rstr}[s] : \text{stringin}[r]$ .

By definition of the translation ( $\llbracket \cdot \rrbracket$ ) the following statements hold:

$$(12) \quad \llbracket \text{rstr}[s] \rrbracket = \text{str}[s]$$

$$(13) \quad \llbracket \text{stringin}[r] \rrbracket = \text{string}$$

$$(14) \quad \llbracket \emptyset \rrbracket = \emptyset$$

Note that  $\emptyset \vdash \text{str}[s] : \text{string}$  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,  $\llbracket \Theta \rrbracket \vdash \text{str}[s] : \text{string}$  by weakening.

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

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

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

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

By induction,  $e_1 \rightsquigarrow \iota_1$  and  $e_2 \rightsquigarrow \iota_2$ . Therefore,  $\llbracket \Psi \rrbracket \vdash \text{concat}(\iota_1; \iota_2)$  by P-T-Concat.

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

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

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

By induction, there exists an  $\iota_1$  such that  $e_1 \mapsto \iota_1$ .

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

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

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  $e_2 \rightsquigarrow \iota_2$  and so  $\llbracket v \rrbracket = \dot{v}$ .

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

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

**Case**  $e = \text{rcoerce}[r](e')$ . The only applicable typing rule is S-T-SafeCoerce, so suppose  $\Psi \vdash \text{rcoerce}[r](e') : \text{stringin}[r]$  where  $\Psi \vdash e' : \text{stringin}[r']$  and  $\mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}$ . By induction,  $e' \rightsquigarrow \iota$  for some  $\iota$ . Therefore,  $\llbracket \text{rcoerce}[r](e') \rrbracket = \iota$  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**  $e = \text{rcheck}[r](e_1; x.e_2; e_3)$ . The applicable typing rule is S-T-Check, so  $\Psi \vdash e : \sigma$  where  $\Psi \vdash e_1 : \text{stringin}[r]$ ,  $\Psi, x : \text{stringin}[r] \vdash e_2 : \sigma$ , 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  $\Psi, s : \text{stringin}[r]$ , and  $e_3 \rightsquigarrow \iota_3$ . Choose  $\iota = \text{check}(\text{rx}[r]; \iota_1; \lambda x.\iota_2; \iota_3)$ . The result follows by induction.

□

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

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

## References

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

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

**Figure 1:** Regular expressions over the alphabet  $\Sigma$ .

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

**Figure 2:** Syntax of  $\lambda_{RS}$ .

$$\begin{aligned} \tau &::= \tau \rightarrow \tau \mid \text{string} \mid \text{regex} && \text{target types} \\ \iota &::= x \mid \dot{v} && \text{target terms} \\ &\quad \mid \text{concat}(\iota; \iota) \mid \text{strcase}(\iota; \iota; x, y.\iota) \\ &\quad \mid \text{rx}[r] \mid \text{replace}(\iota; \iota; \iota) \mid \text{check}(\iota; \iota; \iota; \iota) \\ \dot{v} &::= \lambda x.\iota \mid \text{str}[s] \mid \text{rx}[r] && \text{target values} \end{aligned}$$

**Figure 3:** Syntax for the target language,  $\lambda_P$ , containing strings and statically constructed regular expressions.

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

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

$$\begin{array}{c} \text{S-T-CONCAT} \\ \frac{\Psi \vdash e_1 : \text{stringin}[r_1] \quad \Psi \vdash e_2 : \text{stringin}[r_2]}{\Psi \vdash \text{rconcat}(e_1; e_2) : \text{stringin}[r_1 \cdot r_2]} \end{array}$$

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

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

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

**Figure 4:** Typing rules for  $\lambda_{RS}$ . The typing context  $\Psi$  is standard.

$e \mapsto e$			
	$\frac{\text{SS-E-APPLEFT} \quad e_1 \mapsto e'_1}{e_1(e_2) \mapsto e'_1(e_2)}$	$\frac{\text{SS-E-APPRIGHT} \quad e_2 \mapsto e'_2}{v_1 \mapsto v_1}$	$\frac{\text{SS-E-APPABS}}{(\lambda x : \tau_{11}.t_{12})v_2 \mapsto [v_2/x]t_{12}}$
$e \mapsto^* e$			
	$\frac{\text{RT-REFL}}{e \mapsto^* e}$	$\frac{\text{RT-TRANS} \quad e \mapsto^* e' \quad e' \mapsto e''}{e \mapsto^* e''}$	

**Figure 5:** Call-by-name small step Semantics for  $\lambda$  and its reflexive, transitive closure.

$e \mapsto e$	(Continues figure 6)		
	$\frac{\text{SS-E-CONCAT-LEFT} \quad e_1 \mapsto e'_1}{\text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e'_1; e_2)}$	$\frac{\text{SS-E-CONCAT-RIGHT} \quad e_2 \mapsto e'_2}{\text{rconcat}(v_1; e_2) \mapsto \text{rconcat}(v_1; e'_2)}$	
SS-E-CONCAT	$\frac{}{\text{rconcat}(\text{rstr}[s_1]; \text{rstr}[s_2]) \mapsto \text{rstr}[s_1 s_2]}$	SS-E-CASE-LEFT	$\frac{e_1 \mapsto e'_1}{\text{rstrcase}(e_1; e_2; x, y.e_3) \mapsto \text{rstrcase}(e'_1; e_2; x, y.e_3)}$
SS-E-CASE- $\epsilon$ -VAL	$\frac{}{\text{rstrcase}(\text{rstr}[\epsilon]; e_2; x, y.e_3) \mapsto e_2}$	SS-E-CASE-CONCAT	$\frac{}{\text{rstrcase}(\text{rstr}[as]; e_2; x, y.e_3) \mapsto [\text{rstr}[a], \text{rstr}[s]/x, y]e_3}$
SS-E-REPLACE-LEFT	$\frac{e_1 \mapsto e'_1}{\text{rreplace}[r](v_1; e_2) \mapsto \text{rreplace}[r](v'_1; e_2)}$	SS-E-REPLACE-RIGHT	$\frac{e_2 \mapsto e'_2}{\text{rreplace}[r](e_1; e_2) \mapsto \text{rreplace}[r](e_1; e'_2)}$
SS-E-REPLACE	$\frac{}{\text{rreplace}[r](\text{rstr}[s_1]; \text{rstr}[s_2]) \mapsto \text{rstr}[\text{replace}(r; s_1; s_2)]}$	SS-E-SAFE-COERCE-STEP	$\frac{e \mapsto e'}{\text{rcoerce}[r](e) \mapsto \text{rcoerce}[r](e')}$
SS-E-SAFE-COERCE	$\frac{}{\text{rcoerce}[r](\text{rstr}[s]) \mapsto \text{rstr}[s]}$	SS-E-CHECK-STEPLEFT	$\frac{e \mapsto e'}{\text{rcheck}[r](e; x.e_1; e_2) \mapsto \text{rcheck}[r](e'; x.e_1; e_2)}$
SS-E-CHECK-OK	$\frac{s \in \mathcal{L}\{r\}}{\text{rcheck}[r](\text{rstr}[s]; x.e_1; e_2) \mapsto [\text{rstr}[s]/x]e_1}$	SS-E-CHECK-NOTOK	$\frac{s \notin \mathcal{L}\{r\}}{\text{rcheck}[r](\text{rstr}[s]; x.e_1; e_2) \mapsto e_2}$

**Figure 6:** Small step semantics for  $\lambda_{RS}$ . Extends ??.



$\boxed{\Theta \vdash \iota : \tau} \quad \Theta ::= \emptyset \mid \Theta, x : \tau$			
$\frac{\text{P-T-VAR} \quad x : \tau \in \Theta}{\Theta \vdash x : \tau}$	$\frac{\text{P-T-ABS} \quad \Theta, x : \tau_1 \vdash \iota_2 : \tau_2}{\Theta \vdash \lambda x. \iota_2 : \tau_1 \rightarrow \tau_2}$	$\frac{\text{P-T-APP} \quad \Theta \vdash \iota_1 : \tau_2 \rightarrow \tau \quad \Theta \vdash \iota_2 : \tau_2}{\Theta \vdash \iota_1(\iota_2) : \tau}$	$\frac{\text{P-T-STRING}}{\Theta \vdash \text{str}[s] : \text{string}}$
	$\frac{\text{P-T-REGEX}}{\Theta \vdash \text{rx}[r] : \text{regex}}$	$\frac{\text{P-T-CONCAT} \quad \Theta \vdash \iota_1 : \text{string} \quad \Theta \vdash \iota_2 : \text{string}}{\Theta \vdash \text{concat}(\iota_1; \iota_2) : \text{string}}$	
	$\frac{\text{P-T-CASE} \quad \Theta \vdash \iota_1 : \text{string} \quad \Theta \vdash \iota_2 : \tau \quad \Theta, x : \text{string}, y : \text{string} \vdash \iota_3 : \tau}{\Theta \vdash \text{strcase}(\iota_1; \iota_2; x, y. \iota_3) : \tau}$		
	$\frac{\text{P-T-REPLACE} \quad \Theta \vdash \iota_1 : \text{regex} \quad \Theta \vdash \iota_2 : \text{string} \quad \Theta \vdash \iota_3 : \text{string}}{\Theta \vdash \text{replace}(\iota_1; \iota_2; \iota_3) : \text{string}}$		
	$\frac{\text{P-T-CHECK} \quad \Theta \vdash \iota_x : \text{regex} \quad \Theta \vdash \iota_1 : \text{string} \quad \Theta \vdash \iota_2 : \tau \quad \Theta \vdash \iota_3 : \tau}{\Theta \vdash \text{check}(\iota_x; \iota_1; \iota_2; \iota_3) : \tau}$		

**Figure 7:** Typing rules for  $\lambda_P$ . The typing context  $\Theta$  is standard.

$$\boxed{\ell \mapsto \ell}$$

$\frac{\text{PS-E-CONCATLEFT} \quad \ell_1 \mapsto \ell'_1}{\text{concat}(\ell_1; \ell_2) \mapsto \text{concat}(\ell'_1; \ell_2)}$	$\frac{\text{PS-E-CONCATRIGHT} \quad \ell_2 \mapsto \ell'_2}{\text{concat}(\ell_1; \ell_2) \mapsto \text{concat}(\ell_1; \ell'_2)}$	$\frac{\text{PS-E-CONCAT}}{\text{concat}(\text{str}[s_1]; \text{str}[s_2]) \mapsto \text{str}[s_1 s_2]}$
$\frac{\text{PS-E-CASELEFT} \quad \ell_1 \mapsto \ell'_1}{\text{strcase}(\ell_1; \ell_2; x, y. \ell_3) \mapsto \text{strcase}(\ell'_1; \ell_2; x, y. \ell_3)}$		$\frac{\text{PS-E-CASE-EPSILON}}{\text{strcase}(\epsilon; \ell_2; x, y. \ell_3) \mapsto \ell_2}$
$\frac{\text{PS-E-CASE}}{\text{strcase}(\text{str}[as]; \ell_2; x, y. \ell_3) \mapsto \text{str}[as]}$		$\frac{\text{PS-E-REPLACELLEFT} \quad \ell_1 \mapsto \ell'_1}{\text{replace}(\ell_1; \ell_2; \ell_3) \mapsto \text{replace}(\ell'_1; \ell_2; \ell_3)}$
$\frac{\text{PS-E-REPLACEMID} \quad \ell_2 \mapsto \ell'_2}{\text{replace}(\text{rx}[r]; \ell_2; \ell_3) \mapsto \text{replace}(\text{rx}[r]; \ell'_2; \ell_3)}$	$\frac{\text{PS-E-REPLACERIGHT} \quad \ell_3 \mapsto \ell'_3}{\text{replace}(\text{rx}[r]; \text{str}[s_2]; \ell_3) \mapsto \text{replace}(\text{rx}[r]; \text{str}[s_2]; \ell'_3)}$	
$\frac{\text{PS-E-REPLACE}}{\text{replace}(\text{rx}[r]; \text{str}[s_2]; \text{str}[s_3]) \mapsto \text{str}[\text{replace}(r; s_2; s_3)]}$		$\frac{\text{PS-E-CHECKLEFT} \quad \ell_x \mapsto \ell'_x}{\text{rcheck}[\ell_x](\ell; \ell_1; \ell_2) \mapsto \text{rcheck}[\ell'_x](\ell; \ell_1; \ell_2)}$
$\frac{\text{PS-E-CHECKRIGHT} \quad \ell \mapsto \ell'}{\text{rcheck}[\text{rx}[r]](\ell; \ell_1; \ell_2) \mapsto \text{rcheck}[\text{rx}[r]](\ell'; \ell_1; \ell_2)}$		$\frac{\text{PS-E-CHECK-OK} \quad s \in \mathcal{L}\{r\}}{\text{rcheck}[\text{rx}[r]](\text{str}[s]; \ell_1; \ell_2) \mapsto \ell_1}$
	$\frac{\text{PS-E-CHECK-NOTOK} \quad s \notin \mathcal{L}\{r\}}{\text{rcheck}[\text{rx}[r]](\text{str}[s]; \ell_1; \ell_2) \mapsto \ell_2}$	

**Figure 8:** Small step semantics for  $\lambda_P$  (extends L-E rules)

$$\boxed{\llbracket \sigma \rrbracket = \tau}$$

$$\frac{\text{TR-T-STRING}}{\llbracket \text{stringin}[r] \rrbracket = \text{string}}$$

$$\frac{\text{TR-T-ARROW} \quad \llbracket \sigma_1 \rrbracket = \tau_1 \quad \llbracket \sigma_2 \rrbracket = \tau_2}{\llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket = \tau_1 \rightarrow \tau_2}$$

$$\boxed{\llbracket \Psi \rrbracket = \Theta}$$

$$\frac{\text{TR-T-CONTEXT-EMP}}{\llbracket \emptyset \rrbracket = \emptyset}$$

$$\frac{\text{TR-T-CONTEXT-EXT} \quad \llbracket \Psi \rrbracket = \Theta \quad \llbracket \sigma \rrbracket = \tau}{\llbracket \Psi, x : \sigma \rrbracket = \Theta, x : \tau}$$

$$\boxed{\llbracket e \rrbracket = \iota}$$

$$\frac{\text{TR-VAR}}{\llbracket x \rrbracket = x}$$

$$\frac{\text{TR-ABS} \quad \llbracket e \rrbracket = \iota}{\llbracket \lambda x. e \rrbracket = \lambda x. \iota}$$

$$\frac{\text{TR-APP} \quad \llbracket e_1 \rrbracket = \iota_1 \quad \llbracket e_2 \rrbracket = \iota_2}{\llbracket e_1(e_2) \rrbracket = \iota_1(\iota_2)}$$

$$\frac{\text{TR-CASE} \quad \llbracket e_1 \rrbracket = \iota_1 \quad \llbracket e_2 \rrbracket = \iota_2 \quad \llbracket e_3 \rrbracket = \iota_3}{\llbracket \text{rstrcase}(e_1; e_2; x, y. e_3) \rrbracket = \text{strcase}(\iota_1; \iota_2; x, y. \iota_3)}$$

$$\frac{\text{TR-STRING}}{\llbracket \text{rstr}[s] \rrbracket = \text{str}[s]}$$

$$\frac{\text{TR-CONCAT} \quad \llbracket e_1 \rrbracket = \iota_1 \quad \llbracket e_2 \rrbracket = \iota_2}{\llbracket \text{rconcat}(e_1; e_2) \rrbracket = \text{concat}(\iota_1; \iota_2)}$$

$$\frac{\text{TR-SUBST} \quad \llbracket e_1 \rrbracket = \iota_1 \quad \llbracket e_2 \rrbracket = \iota_2}{\llbracket \text{rreplace}[r](e_1; e_2) \rrbracket = \text{replace}(\text{rx}[r]; \iota_1; \iota_2)}$$

$$\frac{\text{TR-SAFECOERCE} \quad \llbracket e \rrbracket = \iota}{\llbracket \text{rcoerce}[r'](e) \rrbracket = \iota}$$

$$\frac{\text{TR-CHECK} \quad \llbracket e \rrbracket = \iota \quad \llbracket e_1 \rrbracket = \iota_1 \quad \llbracket e_2 \rrbracket = \iota_2}{\llbracket \text{rcheck}[r](e; x. e_1; e_2) \rrbracket = \text{check}(\text{rx}[r]; \iota; (\lambda x. \iota_1)(\iota); \iota_2)}$$

**Figure 9:** Translation from source terms ( $e$ ) to target terms ( $\iota$ ).