

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

fix this

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	Static Semantics	2
3.1.1	Case Analysis	2
3.1.2	Replacement	3
3.2	Dynamic Semantics	3
3.2.1	Canonical Forms	3
3.2.2	Type Safety	3
3.2.3	The Security Theorem	6
4	λ_P	7
4.1	Static Semantics	7
4.2	Dynamic Semantics	7
4.2.1	Canonical Forms	7
4.2.2	Type Safety	7
5	Translation from λ_{RS} to λ_P	10

List of Figures

1	Syntax of Regular Expressions	16
2	Syntax of λ_{RS}	16
3	Syntax of λ_P	16
4	Static Semantics of λ_{RS}	16
5	Dynamic Semantics of λ_{RS}	17
6	Static Semantics of λ_P	18
7	Dynamic Semantics of λ_P	19
8	Translation from λ_{RS} to λ_P	20

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.

3.1 Static Semantics

The static semantics of λ_{RS} is specified in Figure 4. The typing context obeys the standard structural properties of weakening, exchange and contraction.

3.1.1 Case Analysis

The following correctness conditions must hold for any definition of $\text{lhead}(r)$ 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 $\text{ltail}(r) = \delta_{a_1}(r) + \delta_{a_2}(r) + \dots + \delta_{a_i}(r)$.

3.1.2 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.

3.2 Dynamic Semantics

Figure 5 specifies a small-step operational semantics for λ_{RS} .

3.2.1 Canonical Forms

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

1. If $\sigma = \text{stringin}[r]$ 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. □

3.2.2 Type Safety

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

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

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

S-T-Stringin-I. Suppose $\emptyset \vdash \text{rstr}[s] : \text{stringin}[s]$. 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- ϵ -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 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 $e = \text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e'_1; e_2)$ 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 B 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 (6) 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 (7) and (8) 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 (9) Therefore, $a \in \mathcal{L}\{\text{lhead}(r)\}$ by Condition C and $s \in \mathcal{L}\{\text{ltail}(r)\}$ by Condition D.

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 H 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.

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, $\emptyset \vdash \text{rreplace}[r](r'_1; r_2) : \text{stringin}[\text{lreplace}(r, r_1, r_2)]$ 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 conanonical forms, $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$. Therefore,

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

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

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

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 $\emptyset \vdash \text{rcoerce}[r](s_1) : \text{stringin}[r]$ and $\emptyset \vdash \text{rstr}[s_1] : \text{stringin}[r']$ and $\mathcal{L}\{r'\} \subset \mathcal{L}\{r\}$. By Canonical Forms, $s' \in \mathcal{L}\{r'\}$. By the definition of subset, $s' \in \mathcal{L}\{r\}$. Therefore, by S-T-Rstr, we have that $\emptyset \vdash \text{rstr}[s'] : \text{stringin}[r]$.

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

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

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

3.2.3 The Security Theorem

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

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

4 λ_P

We will define a translation to a language with only standard strings and regular expressions. The syntax of λ_P is shown in Figure 3.

4.1 Static Semantics

The static semantics of λ_P is shown in Figure 6. The typing context of λ_P obeys the standard structural properties of weakening, exchange and contraction.

4.2 Dynamic Semantics

The dynamic semantics of λ_P is shown in Figure 7.

4.2.1 Canonical Forms

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

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

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

4.2.2 Type Safety

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

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

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

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

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

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

P-T-Case. Suppose $\emptyset \vdash \text{pstrcase}(\iota_1; \iota_2; x, y, \iota_3) : \tau$ and $\emptyset \vdash \iota_1 : \text{string}$. By induction and canonical forms, either $\iota_1 \mapsto \iota'_1$ or $\iota_1 = \text{str}[s_1]$. If ι_1 steps then we can make progress by PS-E-CaseLeft. If it is a value, then by the definition of strings, either $s_1 = \epsilon$ or $s_1 = as$ for some string s . If s_1 is empty, then we can make progress by PS-E-Case-Epsilon. Otherwise, we can make progress by PS-E-Case-Cons.

P-T-Replace. Suppose $\emptyset \vdash \text{preplace}(\iota_1; \iota_2; \iota_3) : \text{string}$ and $\emptyset \vdash \iota_1 : \text{regex}$ and $\emptyset \vdash \iota_2 : \text{string}$ and $\emptyset \vdash \iota_3 : \text{string}$. By induction and canonical forms, either $\iota_1 \mapsto \iota'_1$ or $\iota_1 = \text{rx}[r]$. Similarly, $\iota_2 \mapsto \iota'_2$ or $\iota_2 = \text{str}[s_2]$, and $\iota_3 \mapsto \iota'_3$ or $\iota_3 = \text{str}[s_3]$. If ι_1 steps, then we can make progress by PS-E-ReplaceLeft. If ι_2 steps then we can make progress by PS-E-ReplaceMid. If ι_3 steps, then we can make progress by PS-E-ReplaceRight. If all three are values, we can make progress by PS-E-Replace.

P-T-Check. Suppose $\emptyset \vdash \text{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4)$ and $\emptyset \vdash \iota_1 : \text{regex}$ and $\emptyset \vdash \iota_2 : \text{string}$. By induction and canonical forms, either $\iota_1 \mapsto \iota'_1$ or $\iota_1 = \text{rx}[r]$. Similarly, $\iota_2 \mapsto \iota'_2$ or $\iota_2 = \text{str}[s]$. If ι_1 steps, then we can make progress by PS-E-CheckLeft. If ι_2 steps, then we can make progress by PS-E-CheckRight. If both are values, then by Assumption B.2, either $s \in \mathcal{L}\{r\}$ or $s \notin \mathcal{L}\{r\}$. In the former case, we can make progress by PS-E-Check-OK. In the latter case, we can make progress by PS-E-Check-NotOK.

□

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

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

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

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

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

Case PS-E-ConcatRight. Suppose $\text{pconcat}(\text{str}[s_1]; \iota_2) \mapsto \text{pconcat}(\text{str}[s_1]; \iota'_2)$ and $\iota_2 \mapsto \iota'_2$. The only applicable typing rule is P-T-Concat, so $\emptyset \vdash \text{str}[s_1] : \text{string}$ and $\emptyset \vdash \iota_2 : \text{string}$. By induction, $\emptyset \vdash \iota'_2 : \text{string}$, so $\emptyset \vdash \text{rconcat}(\text{str}[s_1]; \iota'_2) : \text{string}$ by P-T-Concat.

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

Case PS-E-CaseLeft. Suppose $\text{pstrcase}(\iota_1; \iota_2; x, y. \iota_3) \mapsto \text{rstrcase}(\iota'_1; \iota_2; x, y. \iota_3)$ and $\iota_1 \mapsto \iota'_1$. The only rule that applies is P-T-Case, so:

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

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

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

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

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

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

Case PS-E-ReplaceLeft. Suppose $\text{preplace}(\iota_1; \iota_2; \iota_3) \mapsto \text{preplace}(\iota'_1; \iota_2; \iota_3)$ and $\iota_1 \mapsto \iota'_1$. The only rule that applies is P-T-Replace, so $\tau = \text{string}$ and:

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

By induction, $\emptyset \vdash \iota'_1 : \text{regex}$. Therefore, by P-T-Replace $\emptyset \vdash \text{preplace}(\iota'_1; \iota_2; \iota_3)$.

Case PS-E-ReplaceMid. Suppose $\text{preplace}(\text{rx}[r]; \iota_2; \iota_3) \mapsto \text{preplace}(\text{rx}[r]; \iota'_2; \iota_3)$ and $\iota_2 \mapsto \iota'_2$. The only rule that applies is P-T-Replace, so $\tau = \text{string}$ and:

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

By induction, $\emptyset \vdash \iota'_2 : \text{string}$. Therefore, by P-T-Replace $\emptyset \vdash \text{preplace}(\text{rx}[r]; \iota'_2; \iota_3)$.

Case PS-E-ReplaceRight. Suppose $\text{preplace}(\text{rx}[r]; \text{str}[s]; \iota_3) \mapsto \text{preplace}(\text{rx}[r]; \text{str}[s]; \iota'_3)$ and $\iota_3 \mapsto \iota'_3$. The only rule that applies is P-T-Replace, so $\tau = \text{string}$ and:

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

By induction, $\emptyset \vdash \iota'_3 : \text{string}$. Therefore, by P-T-Replace $\emptyset \vdash \text{preplace}(\text{rx}[r]; \text{str}[s]; \iota'_3)$.

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

Case PS-E-CheckLeft. Suppose $\text{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4) \mapsto \text{pcheck}(\iota'_1; \iota_2; \iota_3; \iota_4)$ and $\iota_1 \mapsto \iota'_1$. The only applicable typing rule is P-T-Check, so:

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

By induction, $\emptyset \vdash \iota'_1 : \text{regex}$. Therefore, by P-T-Check $\emptyset \vdash \text{pcheck}(\iota'_1; \iota_2; \iota_3; \iota_4) : \tau$.

Case PS-E-CheckRight. Suppose $\text{pcheck}(\text{rx}[r]; \iota_2; \iota_3; \iota_4) \mapsto \text{pcheck}(\text{rx}[r]; \iota'_2; \iota_3; \iota_4)$ and $\iota_2 \mapsto \iota'_2$. The only applicable typing rule is P-T-Check, so:

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

By induction, $\emptyset \vdash \iota'_2 : \text{string}$. Therefore, by P-T-Check $\emptyset \vdash \text{pcheck}(\text{rx}[r]; \iota'_2; \iota_3; \iota_4) : \tau$.

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

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

□

5 Translation from λ_{RS} to λ_P

The translation from λ_{RS} to λ_P is defined in Figure 8.

Theorem 8 (Type-Preserving Translation). *If $\Psi \vdash e : \sigma$ then $\llbracket \Psi \rrbracket \vdash \llbracket e \rrbracket : \llbracket \sigma \rrbracket$*

Proof. By induction on the typing relation.

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

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

$$\begin{aligned} \llbracket \lambda x : \sigma_1. e' \rrbracket &= \lambda x : \llbracket \sigma_1 \rrbracket. \llbracket e' \rrbracket \\ \llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket &= \llbracket \sigma_1 \rrbracket \rightarrow \llbracket \sigma_2 \rrbracket \\ \llbracket \Psi, x : \sigma_1 \rrbracket &= \llbracket \Psi \rrbracket, x : \llbracket \sigma_1 \rrbracket \end{aligned}$$

By induction, we have that $\llbracket \Psi \rrbracket, x : \llbracket \sigma_1 \rrbracket \vdash \llbracket e' \rrbracket : \llbracket \sigma_2 \rrbracket$.

By P-T-Abs, we have that $\llbracket \Psi \rrbracket \vdash \lambda x : \llbracket \sigma_1 \rrbracket. \llbracket e' \rrbracket : \llbracket \sigma_1 \rrbracket \rightarrow \llbracket \sigma_2 \rrbracket$.

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

$$\begin{aligned} \llbracket e_1(e_2) \rrbracket &= \llbracket e_1 \rrbracket(\llbracket e_2 \rrbracket) \\ \llbracket \sigma_2 \rightarrow \sigma \rrbracket &= \llbracket \sigma_2 \rrbracket \rightarrow \llbracket \sigma \rrbracket \end{aligned}$$

By induction, $\llbracket \Psi \rrbracket \vdash \llbracket e_1 \rrbracket : \llbracket \sigma_2 \rrbracket \rightarrow \llbracket \sigma \rrbracket$ and $\llbracket \Psi \rrbracket \vdash \llbracket e_2 \rrbracket : \llbracket \sigma_2 \rrbracket$. Therefore, $\llbracket \Psi \rrbracket \vdash \llbracket e_1 \rrbracket(\llbracket e_2 \rrbracket) : \llbracket \sigma \rrbracket$ by P-T-App.

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

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

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

By induction, $\llbracket \Psi \rrbracket \vdash \llbracket e_1 \rrbracket : \text{string}$ and $\llbracket \Psi \rrbracket \vdash \llbracket e_2 \rrbracket : \text{string}$. Thus, $\llbracket \Psi \rrbracket \vdash \text{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) : \text{string}$ by P-T-Concat.

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

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

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

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

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

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

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

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

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

By induction, we have that $\llbracket \Psi \rrbracket \vdash \llbracket e_0 \rrbracket : \text{string}$ and $\llbracket \Psi \rrbracket, x : \text{string} \vdash \llbracket e_1 \rrbracket : \llbracket \sigma \rrbracket$ and $\llbracket \Psi \rrbracket \vdash \llbracket e_2 \rrbracket : \llbracket \sigma \rrbracket$.

By P-T-Regex, we have that $\llbracket \Psi \rrbracket \vdash \text{rx}[r] : \text{regex}$.

By P-T-Abs and P-T-App, we have that $\llbracket \Psi \rrbracket \vdash (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\llbracket e_0 \rrbracket) : \llbracket \sigma \rrbracket$.

By P-T-Check, we have that $\llbracket \Psi \rrbracket \vdash \text{pcheck}(\text{rx}[r]; \llbracket e_0 \rrbracket; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\llbracket e_0 \rrbracket); \llbracket e_2 \rrbracket) : \llbracket \sigma \rrbracket$.

□

Assumption L (Substitution Translation). $\llbracket [v/x]e \rrbracket = \llbracket [v]/x \rrbracket \llbracket e \rrbracket$.

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

1. If $\iota_1 \mapsto^* \iota'_1$ then $\iota_1(\iota_2) \mapsto^* \iota'_1(\iota_2)$.
2. If $\iota_2 \mapsto^* \iota'_2$ then $\dot{v}_1(\iota_2) \mapsto^* \dot{v}_1(\iota'_2)$.
3. If $\iota_1 \mapsto^* \iota'_1$ then $\text{pconcat}(\iota_1; \iota_2) \mapsto^* \text{pconcat}(\iota'_1; \iota_2)$.
4. If $\iota_2 \mapsto^* \iota'_2$ then $\text{pconcat}(\text{str}[s_1]; \iota_2) \mapsto^* \text{pconcat}(\text{str}[s_1]; \iota'_2)$.
5. If $\iota_1 \mapsto^* \iota'_1$ then $\text{pstrcase}(\iota_1; \iota_2; x, y, \iota_3) \mapsto^* \text{pstrcase}(\iota'_1; \iota_2; x, y, \iota_3)$.
6. If $\iota_1 \mapsto^* \iota'_1$ then $\text{preplace}(\iota_1; \iota_2; \iota_3) \mapsto^* \text{preplace}(\iota'_1; \iota_2; \iota_3)$.
7. If $\iota_2 \mapsto^* \iota'_2$ then $\text{preplace}(\text{rx}[r]; \iota_2; \iota_3) \mapsto^* \text{preplace}(\text{rx}[r]; \iota'_2; \iota_3)$.
8. If $\iota_3 \mapsto^* \iota'_3$ then $\text{preplace}(\text{rx}[r]; \text{str}[s]; \iota_3) \mapsto^* \text{preplace}(\text{rx}[r]; \text{str}[s]; \iota'_3)$.
9. If $\iota_1 \mapsto^* \iota'_1$ then $\text{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4) \mapsto^* \text{pcheck}(\iota'_1; \iota_2; \iota_3; \iota_4)$.
10. If $\iota_2 \mapsto^* \iota'_2$ then $\text{pcheck}(\text{rx}[r]; \iota_2; \iota_3; \iota_4) \mapsto^* \text{pcheck}(\text{rx}[r]; \iota'_2; \iota_3; \iota_4)$.

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

Proof. By induction on evaluation and typing.

Case SS-E-AppLeft. Suppose $e_1(e_2) \mapsto e'_1(e_2)$ and $e_1 \mapsto e'_1$. We have by definition that

$$\begin{aligned}\llbracket e_1(e_2) \rrbracket &= \llbracket e_1 \rrbracket(\llbracket e_2 \rrbracket) \\ \llbracket e'_1(e_2) \rrbracket &= \llbracket e'_1 \rrbracket(\llbracket e_2 \rrbracket)\end{aligned}$$

The only typing rule that applies is S-T-App, so $\emptyset \vdash e_1 : \sigma_2 \rightarrow \sigma$.

Inductively, we have that $\llbracket e_1 \rrbracket \mapsto^* \llbracket e'_1 \rrbracket$.

By Assumption M.1, we have that $\llbracket e_1 \rrbracket(\llbracket e_2 \rrbracket) \mapsto^* \llbracket e'_1 \rrbracket(\llbracket e_2 \rrbracket)$.

Case SS-E-AppRight. Suppose $v_1(e_2) \mapsto v_1(e'_2)$ and $e_2 \mapsto e'_2$. We have by definition that

$$\begin{aligned}\llbracket v_1(e_2) \rrbracket &= \llbracket v_1 \rrbracket(\llbracket e_2 \rrbracket) \\ \llbracket v_1(e'_2) \rrbracket &= \llbracket v_1 \rrbracket(\llbracket e'_2 \rrbracket)\end{aligned}$$

The only typing rule that applies is S-T-App, so $\emptyset \vdash e_2 : \sigma_2$.

Inductively, we have that $\llbracket e_2 \rrbracket \mapsto^* \llbracket e'_2 \rrbracket$.

By Assumption M.2, we have that $\llbracket v_1 \rrbracket(\llbracket e_2 \rrbracket) \mapsto^* \llbracket v_1 \rrbracket(\llbracket e'_2 \rrbracket)$.

Case SS-E-AppAbs. Suppose $(\lambda x : \sigma_2.e')(v_2) \mapsto [v_2/x]e'$. We have by definition and Assumption L that

$$\begin{aligned} \llbracket (\lambda x : \sigma_2.e')(v_2) \rrbracket &= (\lambda x : \llbracket \sigma_2 \rrbracket. \llbracket e' \rrbracket) \llbracket v_2 \rrbracket \\ \llbracket [v_2/x]e' \rrbracket &= \llbracket \llbracket v_2 \rrbracket / x \rrbracket \llbracket e' \rrbracket \end{aligned}$$

By PS-E-AppAbs, we have that $(\lambda x : \llbracket \sigma \rrbracket. \llbracket e' \rrbracket) \llbracket v_2 \rrbracket \mapsto \llbracket \llbracket v_2 \rrbracket / x \rrbracket \llbracket e' \rrbracket$.

define
multistep
semantics

Case SS-E-Concat-Left. Suppose $\text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e'_1; e_2)$ and $e_1 \mapsto e'_1$. We have by definition that

$$\begin{aligned} \llbracket \text{rconcat}(e_1; e_2) \rrbracket &= \text{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \\ \llbracket \text{rconcat}(e'_1; e_2) \rrbracket &= \text{pconcat}(\llbracket e'_1 \rrbracket; \llbracket e_2 \rrbracket) \end{aligned}$$

The only typing rule that applies is S-T-Concat, so $\emptyset \vdash e_1 : \text{stringin}[r_1]$.

Inductively, we have that $\llbracket e_1 \rrbracket \mapsto^* \llbracket e'_1 \rrbracket$.

By Assumption M.3, we have that $\text{pconcat}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \mapsto^* \text{pconcat}(\llbracket e'_1 \rrbracket; \llbracket e_2 \rrbracket)$.

Case SS-E-Concat-Right. Suppose $\text{rconcat}(\text{rstr}[s]; e_2) \mapsto \text{rconcat}(\text{rstr}[s]; e'_2)$ and $e_2 \mapsto e'_2$. We have by definition that

$$\begin{aligned} \llbracket \text{rconcat}(\text{rstr}[s]; e_2) \rrbracket &= \text{pconcat}(\text{str}[s]; \llbracket e_2 \rrbracket) \\ \llbracket \text{rconcat}(\text{rstr}[s]; e'_2) \rrbracket &= \text{pconcat}(\text{str}[s]; \llbracket e'_2 \rrbracket) \end{aligned}$$

The only typing rule that applies is S-T-Concat, so $\emptyset \vdash e_2 : \text{stringin}[r_2]$.

Inductively, we have that $\llbracket e_2 \rrbracket \mapsto^* \llbracket e'_2 \rrbracket$.

By Assumption M.4, we have that $\text{pconcat}(\text{str}[s]; \llbracket e_2 \rrbracket) \mapsto^* \text{pconcat}(\text{str}[s]; \llbracket e'_2 \rrbracket)$.

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

$$\begin{aligned} \llbracket \text{rconcat}(\text{rstr}[s_1]; \text{rstr}[s_2]) \rrbracket &= \text{pconcat}(\text{str}[s_1]; \text{str}[s_2]) \\ \llbracket \text{rstr}[s_1 s_2] \rrbracket &= \text{str}[s_1 s_2] \end{aligned}$$

By PS-E-Concat, we have $\text{pconcat}(\text{str}[s_1]; \text{str}[s_2]) \mapsto \text{str}[s_1 s_2]$.

Case SS-E-Case-Left Suppose $\text{rstrcase}(e_1; e_2; x, y.e_3) \mapsto \text{rstrcase}(e'_1; e_2; x, y.e_3)$ and $e_1 \mapsto e'_1$. We have by definition that:

$$\begin{aligned} \llbracket \text{rstrcase}(e_1; e_2; x, y.e_3) \rrbracket &= \text{pstrcase}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket; x, y. \llbracket e_3 \rrbracket) \\ \llbracket \text{rstrcase}(e'_1; e_2; x, y.e_3) \rrbracket &= \text{pstrcase}(\llbracket e'_1 \rrbracket; \llbracket e_2 \rrbracket; x, y. \llbracket e_3 \rrbracket) \end{aligned}$$

The only typing rule that applies is S-T-Case, so $\emptyset \vdash e_1 : \text{stringin}[r]$.

Inductively, $\llbracket e_1 \rrbracket \mapsto^* \llbracket e'_1 \rrbracket$.

By Assumption M.5, we have that $\text{pstrcase}(\llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket; x, y. \llbracket e_3 \rrbracket) \mapsto^* \text{pstrcase}(\llbracket e'_1 \rrbracket; \llbracket e_2 \rrbracket; x, y. \llbracket e_3 \rrbracket)$.

Case SS-E-Case- ϵ -Val. Suppose $\text{rstrcase}(\text{rstr}[\epsilon]; e_2; x, y.e_3) \mapsto e_2$. We have by definition that:

rename
this rule

$$\llbracket \text{rstrcase}(\text{rstr}[\epsilon]; e_2; x, y.e_3) \rrbracket = \text{pstrcase}(\text{str}[\epsilon]; \llbracket e_2 \rrbracket; x, y.\llbracket e_3 \rrbracket)$$

By PS-E-Case-Epsilon, we have that $\text{pstrcase}(\text{str}[\epsilon]; \llbracket e_2 \rrbracket; x, y.\llbracket e_3 \rrbracket) \mapsto \llbracket e_2 \rrbracket$.

Case SS-E-Case-Concat. Suppose $\text{rstrcase}(\text{rstr}[as]; e_2; x, y.e_3) \mapsto [\text{rstr}[a], \text{rstr}[s]/x, y]e_3$. We have by Assumption L and definition that

rename
this rule

$$\begin{aligned} \llbracket \text{rstrcase}(\text{rstr}[as]; e_2; x, y.e_3) \rrbracket &= \text{pstrcase}(\text{str}[as]; \llbracket e_2 \rrbracket; x, y.\llbracket e_3 \rrbracket) \\ \llbracket [\text{rstr}[a], \text{rstr}[s]/x, y]e_3 \rrbracket &= [\text{str}[a], \text{str}[s]/x, y]\llbracket e_3 \rrbracket \end{aligned}$$

By PS-E-Case-Cons, we have that $\text{pstrcase}(\text{str}[as]; \llbracket e_2 \rrbracket; x, y.\llbracket e_3 \rrbracket) \mapsto^* [\text{str}[a], \text{str}[s]/x, y]\llbracket e_3 \rrbracket$.

Case SS-E-Replace-Left. Suppose $\text{rreplace}[r](e_1; e_2) \mapsto \text{rreplace}[r](e'_1; e_2)$ and $e_1 \mapsto e'_1$. We have by definition that

$$\begin{aligned} \llbracket \text{rreplace}[r](e_1; e_2) \rrbracket &= \text{preplace}(\text{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \\ \llbracket \text{rreplace}[r](e'_1; e_2) \rrbracket &= \text{preplace}(\text{rx}[r]; \llbracket e'_1 \rrbracket; \llbracket e_2 \rrbracket) \end{aligned}$$

The only typing rule that applies is S-T-Replace, so $\emptyset \vdash e_1 : \text{stringin}[r_1]$.

Inductively, we have that $\llbracket e_1 \rrbracket \mapsto^* \llbracket e'_1 \rrbracket$.

By Assumption M.7, we have that $\text{preplace}(\text{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \mapsto^* \text{preplace}(\text{rx}[r]; \llbracket e'_1 \rrbracket; \llbracket e_2 \rrbracket)$.

Case SS-E-Replace-Right. Suppose $\text{rreplace}[r](e_1; e_2) \mapsto \text{rreplace}[r](e_1; e'_2)$ and $e_2 \mapsto e'_2$. By definition,

$$\begin{aligned} \llbracket \text{rreplace}[r](e_1; e_2) \rrbracket &= \text{preplace}(\text{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \\ \llbracket \text{rreplace}[r](e_1; e'_2) \rrbracket &= \text{preplace}(\text{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e'_2 \rrbracket) \end{aligned}$$

The only typing rule that applies is S-T-Replace, so $\emptyset \vdash e_2 : \text{stringin}[r_2]$.

Inductively, we have that $\llbracket e_2 \rrbracket \mapsto^* \llbracket e'_2 \rrbracket$.

By Assumption M.8, we have that $\text{preplace}(\text{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e_2 \rrbracket) \mapsto^* \text{preplace}(\text{rx}[r]; \llbracket e_1 \rrbracket; \llbracket e'_2 \rrbracket)$.

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

$$\begin{aligned} \llbracket \text{rreplace}[r](\text{rstr}[s_1]; \text{rstr}[s_2]) \rrbracket &= \text{preplace}(\text{rx}[r]; \text{str}[s_1]; \text{str}[s_2]) \\ \llbracket \text{rstr}[\text{lreplace}(r, s_1, s_2)] \rrbracket &= \text{str}[\text{lreplace}(r, s_1, s_2)] \end{aligned}$$

By PS-E-Replace, we have that $\text{preplace}(\text{rx}[r]; \text{str}[s_1]; \text{str}[s_2]) \mapsto \text{str}[\text{lreplace}(r, s_1, s_2)]$.

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

$$\begin{aligned} \llbracket \text{rcoerce}[r](e) \rrbracket &= \llbracket e \rrbracket \\ \llbracket \text{rcoerce}[r](e') \rrbracket &= \llbracket e' \rrbracket \end{aligned}$$

The only typing rule that applies is S-T-SafeCoerce, so $\emptyset \vdash e' : \text{stringin}[r']$.

Inductively, $\llbracket e \rrbracket \mapsto^* \llbracket e' \rrbracket$.

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

$$\begin{aligned}\llbracket \text{rcoerce}[r](\text{rstr}[s]) \rrbracket &= \text{str}[s] \\ \text{rstr}[s] &= \text{str}[s]\end{aligned}$$

We have that $\text{str}[s] \mapsto^* \text{str}[s]$ because the multistep judgement is reflexive.

Case SS-E-Check-StepLeft. Suppose $\text{rcheck}[r](e_0; x.e_1; e_2) \mapsto \text{rcheck}[r](e'_0; x.e_1; e_2)$ and $e_0 \mapsto e'_0$. We have by definition that

$$\begin{aligned}\llbracket \text{rcheck}[r](e_0; x.e_1; e_2) \rrbracket &= \text{pcheck}(\text{rx}[r]; \llbracket e_0 \rrbracket; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\llbracket e_0 \rrbracket); \llbracket e_2 \rrbracket) \\ \llbracket \text{rcheck}[r](e'_0; x.e_1; e_2) \rrbracket &= \text{pcheck}(\text{rx}[r]; \llbracket e'_0 \rrbracket; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\llbracket e'_0 \rrbracket); \llbracket e_2 \rrbracket)\end{aligned}$$

Inductively, $e_0 \mapsto^* e'_0$.

By Assumption M.10, we have that

$$\begin{aligned}\text{pcheck}(\text{rx}[r]; \llbracket e_0 \rrbracket; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\llbracket e_0 \rrbracket); \llbracket e_2 \rrbracket) \\ \mapsto^* \text{pcheck}(\text{rx}[r]; \llbracket e'_0 \rrbracket; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\llbracket e_0 \rrbracket); \llbracket e_2 \rrbracket)\end{aligned}$$

Case SS-E-Check-Ok Suppose $\text{rcheck}[r](\text{rstr}[s]; x.e_1; e_2) \mapsto [\text{rstr}[s]/x]e_1$ and $s \in \mathcal{L}\{r\}$. We have by definition that

$$\begin{aligned}\llbracket \text{rcheck}[r](\text{rstr}[s]; x.e_1; e_2) \rrbracket &= \text{pcheck}(\text{rx}[r]; \text{str}[s]; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\text{str}[s]); \llbracket e_2 \rrbracket) \\ \llbracket [\text{rstr}[s]/x]e_1 \rrbracket &= [\text{str}[s]/x]\llbracket e_1 \rrbracket\end{aligned}$$

By PS-E-Check-OK, we have that

$$\text{pcheck}(\text{rx}[r]; \text{str}[s]; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\text{str}[s]); \llbracket e_2 \rrbracket) \mapsto (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\text{str}[s])$$

By PS-E-AppAbs, we have that

$$(\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\text{str}[s]) \mapsto [\text{str}[s]/x]\llbracket e_1 \rrbracket$$

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

$$\llbracket \text{rcheck}[r](\text{rstr}[s]; x.e_1; e_2) \rrbracket = \text{pcheck}(\text{rx}[r]; \text{str}[s]; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\text{str}[s]); \llbracket e_2 \rrbracket)$$

By PS-E-Check-NotOK, we have that

$$\text{pcheck}(\text{rx}[r]; \text{str}[s]; (\lambda x : \text{string}. \llbracket e_1 \rrbracket)(\text{str}[s]); \llbracket e_2 \rrbracket) \mapsto \llbracket e_2 \rrbracket$$

□

References

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

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

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

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

Figure 2: Syntax of λ_{RS}

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

Figure 3: Syntax of λ_P

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

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

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

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

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

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

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

$e \mapsto e$			
SS-E-APPLEFT $\frac{e_1 \mapsto e'_1}{e_1(e_2) \mapsto e'_1(e_2)}$	SS-E-APPRIGHT $\frac{e_2 \mapsto e'_2}{v_1(e_2) \mapsto v_1(e'_2)}$	SS-E-APPABS $\frac{}{(\lambda x : \sigma.e)v_2 \mapsto [v_2/x]e}$	SS-E-CONCAT-LEFT $\frac{e_1 \mapsto e'_1}{\text{rconcat}(e_1; e_2) \mapsto \text{rconcat}(e'_1; e_2)}$
SS-E-CONCAT-RIGHT $\frac{e_2 \mapsto e'_2}{\text{rconcat}(v_1; e_2) \mapsto \text{rconcat}(v_1; e'_2)}$		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- ϵ -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-SAFECOERCE-STEP $\frac{e \mapsto e'}{\text{rcoerce}[r](e) \mapsto \text{rcoerce}[r](e')}$	SS-E-SAFECOERCE $\frac{}{\text{rcoerce}[r](\text{rstr}[s]) \mapsto \text{rstr}[s]}$	SS-E-CHECK-STEPLLEFT $\frac{e \mapsto e'}{\text{rcheck}[r](e; x.e_1; e_2) \mapsto \text{rcheck}[r](e'; x.e_1; e_2)}$	
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 5: Small step semantics for λ_{RS} .

$\boxed{\Theta \vdash \iota : \tau} \quad \Theta ::= \emptyset \mid \Theta, x : \tau$			
$\frac{\text{P-T-VAR} \quad x : \tau \in \Theta}{\Theta \vdash x : \tau}$	$\frac{\text{P-T-ABS} \quad \Theta, x : \tau_1 \vdash \iota_2 : \tau_2}{\Theta \vdash \lambda x. \iota_2 : \tau_1 \rightarrow \tau_2}$	$\frac{\text{P-T-APP} \quad \Theta \vdash \iota_1 : \tau_2 \rightarrow \tau \quad \Theta \vdash \iota_2 : \tau_2}{\Theta \vdash \iota_1(\iota_2) : \tau}$	$\frac{\text{P-T-STRING}}{\Theta \vdash \text{str}[s] : \text{string}}$
	$\frac{\text{P-T-REGEX}}{\Theta \vdash \text{rx}[r] : \text{regex}}$	$\frac{\text{P-T-CONCAT} \quad \Theta \vdash \iota_1 : \text{string} \quad \Theta \vdash \iota_2 : \text{string}}{\Theta \vdash \text{pconcat}(\iota_1; \iota_2) : \text{string}}$	
	$\frac{\text{P-T-CASE} \quad \Theta \vdash \iota_1 : \text{string} \quad \Theta \vdash \iota_2 : \tau \quad \Theta, x : \text{string}, y : \text{string} \vdash \iota_3 : \tau}{\Theta \vdash \text{pstrcase}(\iota_1; \iota_2; x, y, \iota_3) : \tau}$		
	$\frac{\text{P-T-REPLACE} \quad \Theta \vdash \iota_1 : \text{regex} \quad \Theta \vdash \iota_2 : \text{string} \quad \Theta \vdash \iota_3 : \text{string}}{\Theta \vdash \text{preplace}(\iota_1; \iota_2; \iota_3) : \text{string}}$		
	$\frac{\text{P-T-CHECK} \quad \Theta \vdash \iota_1 : \text{regex} \quad \Theta \vdash \iota_2 : \text{string} \quad \Theta \vdash \iota_3 : \tau \quad \Theta \vdash \iota_4 : \tau}{\Theta \vdash \text{pcheck}(\iota_1; \iota_2; \iota_3; \iota_4) : \tau}$		

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

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

$$\begin{array}{c}
\text{PS-E-APPLEFT} \quad \frac{\ell_1 \mapsto \ell'_1}{\ell_1(\ell_2) \mapsto \ell'_1(\ell_2)} \quad \text{PS-E-APPRIGHT} \quad \frac{\ell_2 \mapsto \ell'_2}{\dot{\nu}_1(\ell_2) \mapsto \dot{\nu}_1(\ell'_2)} \quad \text{PS-E-APPABS} \quad \frac{}{(\lambda x : \tau. \ell) \dot{\nu}_2 \mapsto [\dot{\nu}_2/x] \ell} \quad \text{PS-E-CONCATLEFT} \quad \frac{\ell_1 \mapsto \ell'_1}{\text{pconcat}(\ell_1; \ell_2) \mapsto \text{pconcat}(\ell'_1; \ell_2)} \\
\\
\text{PS-E-CONCATRIGHT} \quad \frac{\ell_2 \mapsto \ell'_2}{\text{pconcat}(\text{str}[s_1]; \ell_2) \mapsto \text{pconcat}(\text{str}[s_1]; \ell'_2)} \quad \text{PS-E-CONCAT} \quad \frac{}{\text{pconcat}(\text{str}[s_1]; \text{str}[s_2]) \mapsto \text{str}[s_1 s_2]} \\
\\
\text{PS-E-CASELEFT} \quad \frac{\ell_1 \mapsto \ell'_1}{\text{pstrcase}(\ell_1; \ell_2; x, y. \ell_3) \mapsto \text{pstrcase}(\ell'_1; \ell_2; x, y. \ell_3)} \quad \text{PS-E-CASE-EPSILON} \quad \frac{}{\text{pstrcase}(\text{str}[\epsilon]; \ell_2; x, y. \ell_3) \mapsto \ell_2} \\
\\
\text{PS-E-CASE-CONS} \quad \frac{}{\text{pstrcase}(\text{str}[as]; \ell_2; x, y. \ell_3) \mapsto [\text{str}[a], \text{str}[s]/x, y] \ell_3} \quad \text{PS-E-REPLACELEFT} \quad \frac{\ell_1 \mapsto \ell'_1}{\text{preplace}(\ell_1; \ell_2; \ell_3) \mapsto \text{preplace}(\ell'_1; \ell_2; \ell_3)} \\
\\
\text{PS-E-REPLACEMID} \quad \frac{\ell_2 \mapsto \ell'_2}{\text{preplace}(\text{rx}[r]; \ell_2; \ell_3) \mapsto \text{preplace}(\text{rx}[r]; \ell'_2; \ell_3)} \quad \text{PS-E-REPLACERIGHT} \quad \frac{\ell_3 \mapsto \ell'_3}{\text{preplace}(\text{rx}[r]; \text{str}[s_2]; \ell_3) \mapsto \text{preplace}(\text{rx}[r]; \text{str}[s_2]; \ell'_3)} \\
\\
\text{PS-E-REPLACE} \quad \frac{}{\text{preplace}(\text{rx}[r]; \text{str}[s_2]; \text{str}[s_3]) \mapsto \text{str}[\text{replace}(r; s_2; s_3)]} \quad \text{PS-E-CHECKLEFT} \quad \frac{\ell_1 \mapsto \ell'_1}{\text{pcheck}(\ell_1; \ell_2; \ell_3; \ell_4) \mapsto \text{pcheck}(\ell'_1; \ell_2; \ell_3; \ell_4)} \\
\\
\text{PS-E-CHECKRIGHT} \quad \frac{\ell_2 \mapsto \ell'_2}{\text{pcheck}(\text{rx}[r]; \ell_2; \ell_3; \ell_4) \mapsto \text{pcheck}(\text{rx}[r]; \ell'_2; \ell_3; \ell_4)} \quad \text{PS-E-CHECK-OK} \quad \frac{s \in \mathcal{L}\{r\}}{\text{pcheck}(\text{rx}[r]; \text{str}[s]; \ell_3; \ell_4) \mapsto \ell_3} \\
\\
\text{PS-E-CHECK-NOTOK} \quad \frac{s \notin \mathcal{L}\{r\}}{\text{pcheck}(\text{rx}[r]; \text{str}[s]; \ell_3; \ell_4) \mapsto \ell_4}
\end{array}$$

Figure 7: Small step semantics for λ_P

$$\llbracket \sigma \rrbracket = \tau$$

$$\begin{aligned} \llbracket \text{stringin}[r] \rrbracket &= \text{string} \\ \llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket &= \llbracket \sigma_1 \rrbracket \rightarrow \llbracket \sigma_2 \rrbracket \end{aligned}$$

$$\llbracket \Psi \rrbracket = \Theta$$

$$\begin{aligned} \llbracket \emptyset \rrbracket &= \emptyset \\ \llbracket \Psi, x : \sigma \rrbracket &= \llbracket \Psi \rrbracket, x : \llbracket \sigma \rrbracket \end{aligned}$$

$$\llbracket e \rrbracket = \iota$$

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

Figure 8: Translation from λ_{RS} to λ_P