Statically Typed String Sanitation Inside a Python: Technical Report

Nathan Fulton Cyrus Omar Jonathan Aldrich

November 17, 2014

Abstract

Web applications must ultimately command systems like web browsers and database engines using strings. Strings derived from improperly sanitized user input can thus be a vector for command injection attacks.

In this report, we introduce *regular string types*, which classify strings known statically to be in a specified regular language. These types come equipped with common operations like concatenation, substitution and coercion, so they can be used to implement, in a conventional manner, the portions of a web application or application framework that must directly construct command strings. Simple type annotations at key interfaces can be used to statically verify that sanitization has been performed correctly without introducing redundant run-time checks. We specify this type system in a minimal typed lambda calculus, λ_{RS}.

We then specify a translation from λ_{RS} to a language fragment containing only standard strings and regular expressions and prove that the correctness theorem for λ_{RS} is preserved under translation.

1 Introduction

Command injection vulnerabilities are among the most common and severe security vulnerabilities in modern web applications. They arise because web applications, at their boundaries, control external systems using commands represented as strings. In [1] the authors argue that extensible type systems are an attractive solution to this important security problem. This Technical Report contains supporting evidence for claims put forth and explained in [1], including proofs of lemmas and theorems asserted in the paper, examples, and additional discussion of the paper's technical content.

Theorems and lemmas appearing in [1] are numbered, while supporting facts appearing only in the Technical Report are lettered. Numbered items correspond to the numbering in [1].

2 Proofs of Lemmas and Theorems about λ_{RS}

This section presents proofs of lemmas and theorems about the type systems presented in [1], the accompanying paper. In addition, we provide some examples to help motivate and explain definitions.

To facilitate the type safety proof, we introduce a small step semantics for both λ_{RS} and λ_P. All theorems in this section are proven as stated in [1].

2.1 Head and Tail Operations

Definition 1 (Definition of lhead(r))**.** The relation $\mathsf{lhead}(r) = r'$ is defined in terms of the structure of r:

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

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

Definition 3 (Definition of ltail(r))**.** The relation $\mathsf{ltail}(r) = r'$ is defined in terms of $\mathsf{lhead}(r)$. Note that $\mathsf{lhead}(r) = a_1 + a_2 + ... + a_i$. We define $\mathsf{ltail}(r) = \delta_{a_1}(r) + \delta_{a_2}(r) + ... + \delta_{a_i}(r) + \epsilon$.

Using these definitions of head and tail, we establish a correctness result upon which type soundness for the concatenation operator in λ_{RS} depends.

TR Lemma A (Leading characters are in the head)**.** *If* $c_1 \cdot c_2 \cdot ... \cdot c_m \in \mathcal{L}\{r\}$*, then* $c_1 \in \mathsf{lhead}(r)$*.*

Proof. By structural induction on r... □

TR Theorem B (Correctness of ltail)**.** *If* $s \in \mathcal{L}\{r\}$ *then* $s \in \mathcal{L}\{\mathsf{lhead}(r)\} \cdot \mathcal{L}\{\mathsf{ltail}(r)\}$*.*

Proof. The proof proceeds by structural induction on r. In each case, we demonstrate three facts:

- First, that the string $s = a \cdot b$ is a concatenation of two substrings (each of which may be ϵ).
- Second, $a \in \mathcal{L}\{\mathsf{lhead}(r)\}$ (or, equivalently, $s \in \mathcal{L}\{\mathsf{lhead}(r)\}$ letting $b = \epsilon$).
- Third, there exists a $c \in \mathsf{lhead}(r)$ such that $a = c \cdot a'$ and $a' \cdot b \in \mathcal{L}\{\delta_c(r)\}$, or else $s = a$.

Throughout, we consider ϵ as idenetity under all operators for expressions and $\cdot$ for strings.

Case $r = c$ **for** $c \in Sigma$. Note that if $s \in \mathcal{L}\{c\}$ then $s = c$. Note that $\mathsf{lhead}(c) = \mathsf{lhead}(c, \epsilon) = c$, and $c \in \mathcal{L}\{c\}$. So $c \in \mathcal{L}\{c\}$, which is sufficient since $\epsilon \in \mathsf{ltail}(r)$.

Case $r = r_1 \cdot r_2$. Suppose $s \in \mathcal{L}\{r\}$. We may decompose s as $s_1 \cdot ... \cdot s_n$ such that $s_1 = c_1 \cdot ... \cdot c_m \in r_1$. Note that $c_1 \in \mathsf{lhead}(r_1)$ by A. Let $a' = c_2 \cdot ... \cdot c_m$. It suffices to show that $a' \cdot b \in \mathcal{L}\{\delta_c(r)\}$. Since $s = c_1 \cdot a' \cdot b$ and $s \in \mathcal{L}\{r\}$, it follows by the definition of derivative that $a' \cdot b \in \delta_{c_1}(r)$.

Case $r = r_1 + r_2$. Suppose $s \in \mathcal{L}\{r_1 + r_2\}$ so that $s \in \mathcal{L}\{r_1\}$ or $s \in \mathcal{L}\{r_2\}$. In either case, the result follows by induction.

Case $r = q^*$. Note that $s = \epsilon$ or else $s = s_1 \cdot s_2 \cdot ...s_n$ where each $s_i \in \mathcal{L}\{q\}$[1].

Suppose $s = \epsilon$. Note that $\mathsf{lhead}(r) = \mathsf{lhead}(q^*) = \mathsf{lhead}(q^*, \epsilon) = \mathsf{lhead}(\epsilon, \epsilon) + h' = \epsilon + h'$. Therefore, $s \in \mathsf{lhead}(r)$.

[1] intuitively, s is either empty or a finitary concatenation of strings matching q.

In the other case, suppose $s = s_1 \cdot ... \cdot s_n$ where each $s_i \in \mathcal{L}\{q\}$. Let $a = s_1$ and $b = s_2 \cdot ...s_n$, so that $s = a \cdot b$. Note that $a = c_1 \cdot ... \cdot c_n$. We will show that $c_1 \in \mathcal{L}\{\mathsf{lhead}(q^*)\}$ and that $(c_2 \cdot ...c_n) \cdot b \in \delta_{c_1}(q^*)$.

The latter property is trivial. Note that $s = c_1 \cdot c_2 \cdot ... \cdot c_n \cdot b \in \mathcal{L}\{q^*\}$. The result follows immediately from the definition of derivative.

What remains to be shown is that $c_1 \in \mathcal{L}\{q^*\}$. Note that $\mathsf{lhead}(q^*) = \epsilon + \mathsf{lhead}(q, \epsilon)$. Recall that $a = s_1$ where $s_1 \in \mathcal{L}\{q\}$. Also recall that $a = c_1 \cdot ... \cdot c_n$. Therefore, $c_1 \in \mathsf{lhead}(q)$ by A.

$\square$

TR Example C (All the heads of all the tails can be more than one head and tail)**.** $r \neq \mathsf{lhead}(r) \cdot \mathsf{ltail}(r)$.

Proof. A simple counter-example is $ab + cd$. Note that $\mathsf{lhead}(ab + cd) = a + c$ and $\mathsf{ltail}(ab+cd) = b+d$. Therefore, $\{ad, bc\} \subset \mathcal{L}\{\mathsf{lhead}(ab+cd) \cdot \mathsf{ltail}(ab+cd)\}$ even though neither of these is in $\mathcal{L}\{r\}$. $\square$

Example C does not imply a counter-example to type soundness because $s \in \mathcal{L}\{r\} \implies s \in \mathsf{lhead}(r) \cdot \mathsf{ltail}(r)$ is the property required for soundness. Still, in a production implementation, it will make sense to massage the definitions of $\mathsf{lhead}(r)$ and $\mathsf{ltail}(r)$ so that type information is not unnecessarily lost during substring operations.

This is a general pattern in string operations: λ_{RS} simulates – within the type system – common operations on strings. If there is an operation for concatenating to strings, we define an operation for concatenating two regular expressions. If there is an operation for peeling off the first (n) characters of a string, then we define an operation for converting a regular expression r into a regular expression r' which recognizes any n^{th} suffix of a string in r.

It is important to note, however, that the type system need not *exactly* simulate the action of string operations. In the case of concatenation, we lose some information because more string values are possible – according the types – than are actually possible in the dynamic semantics. Soundness is not lost because the types are conservative in their approximation.

In the case of string replacement, there are *trivial* definitions of substitution (on strings) and replacement (on languages) which over-approximate the effect of a substitution. Closing these gaps in approximation is important, and motivates the string operations portion of this technical report.

2.2 Some Corollaries About Substitution and Language Replacement

Definition 4 (`subst`)**.** We consider several choices in the string operations section.

Definition 5 (`lreplace`)**.** We consider several choices in the string operations section.

Proposition 6 (Closure)**.** *If $\mathcal{L}\{r\}, \mathcal{L}\{r_1\}$ and $\mathcal{L}\{r_2\}$ are regular languages, then $\mathcal{L}\{\texttt{lreplace}(r, r_1, r_2)\}$ is also a regular language.*

Proof. This result is proven for various formulations in the next section. □

Proposition 7 (Substitution Correspondence)**.** *If $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$ then $\mathsf{subst}(r; s_1; s_2) \in \mathcal{L}\{\texttt{lreplace}(r, r_1, r_2)\}$.*

Proof. This is exactly the correctness result proven for some pairs of `subst` and `replace` in the previous section. □

Lemma 8 (Properties of Regular Languages.)**.**

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

Lemma 8 states some well-known properties about regular expressions.

Lemma 9. *If $\emptyset \vdash e : \mathsf{stringin}[r]$ then r is a well-formed regular expression.*

Proof. The proof proceeds by induction on the derivation of the premise. The only non-trivial cases (those which require more than an appeal to inversion) are S-T-Case, S-T-Replace and S-T-Concat.

In the S-T-Case case, note that lhead and ltail are total functions for well-formed regular expressions to well-formed regular expressions.

In the S-T-Concat case, note that Lemma 6 implies that if r_1 and r_2 are regular expressions then so is $r_1 \cdot r_2$.

In the S-T-Replace case, it suffices to show that $\texttt{lreplace}(r, r_1, r_2)$ is a regular expression assuming (inductively) that r, r_1 and r_2 are all regular expressions. This follows from the Closure proposition. □

2.3 The Small Step Semantics

To prove type safety and the security theorems for the big step semantics, we first prove type safety for a small step semantics in Figure 7 and then extend this to the big step semantics in Figure 5 by proving a correspondence between the semantics.

TR Conjecture D. *if* $\emptyset \vdash e : \sigma$ *then* $e \mapsto^* v$ *such that* v val.

We do not develop the full proof here, but note that the simply typed lambda calculus terminates. For the string fragment, observe that the S-T- rules do not add any non-trivial binding structure because substitutions $[e/x]e'$ may only occur in the special case where $e = \mathsf{rstr}[s]$, so that the length of the term never increases and the number of free variables strictly decreases. Therefore, the standard normalization argument proceeds without complication after fixing an evaluation order for the compatibility rules (all our other proofs are agnostic to evaluation order).

TR Lemma E (Canonical Forms)**.** *Suppose* v val.
If $\emptyset \vdash v : \mathsf{stringin}[r]$ *then* $v = \mathsf{rstr}[s]$.
If $\emptyset \vdash v : \sigma \to \sigma'$ *then* $v = \lambda x.e'$ *for some* e'.

Proof. By inspection of valuation and typing rules. □

For the sake of completeness, we include a statement of the weaker lemma stated in the paper:

Lemma 10 (Canonical Forms for the String Fragment of λ_{RS})**.** *If* $\emptyset \vdash e : \mathsf{stringin}[r]$ *and* $e \Downarrow v$ *then* $v = \mathsf{rstr}[s]$.

Proof. This fact follows directly from Lemma E. □

TR Lemma F (Progress of small step semantics.)**.** *If* $\emptyset \vdash e : \sigma$ *either* e val *or* $e \mapsto e'$ *for some* e'.

Proof. The proof proceeds by 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 type safety for the simply typed lambda calculus.

S-T-Stringin-I. Suppose $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[s]$. The $\mathsf{rstr}[s]$ val by SS-E-RStr.

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

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

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

S-T-Replace. Suppose $e = \mathsf{rreplace}[r](e_1; e_2)$ and $\emptyset \vdash e : \mathsf{stringin}[r']$ where, by inversion of S-T-Replace,

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

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

$$\mathtt{lreplace}(r, r_1, r_2) = r' \tag{3}$$

By (1), inversion and induction e_1 val or $e_1 \mapsto e_1'$ 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 val and also e_2 val. But then by Canonical Forms, $e_1 = \mathsf{rstr}[s_1]$ and $e_2 = \mathsf{rstr}[s_2]$. Therefore, $e \mapsto \mathsf{rstr}[\mathsf{subst}(r; s_1; s_2)]$ by SS-E-Replace.

S-T-SafeCoerce. Suppose that $\emptyset \vdash \mathsf{rcoerce}[r](e_1) : \mathsf{stringin}[r]$. By inversion of S-T-SafeCoerce, $\emptyset \vdash e_1 : \mathsf{stringin}[r']$ for $\mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}$.By induction, e_1 val 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 val and by Canonical Forms $e_1 = \mathsf{rstr}[s]$. In this case, $e = \mathsf{rcoerce}[r](\mathsf{rstr}[s]) \mapsto \mathsf{rstr}[s]$ by SS-E-SafeCoerce.

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

By inversion of S-T-Check:

$$\vdash e_0 : \mathsf{stringin}[r_0] \tag{4}$$
$$x : \mathsf{stringin}[r] \vdash e_1 : \sigma \tag{5}$$
$$\vdash e_2 : \sigma \tag{6}$$

By (6) and induction, $e_0 \mapsto e_0'$ or e_0 val. In the former case e steps by SS-E-Check-StepRight. Otherwise, $e_0 = \mathsf{rstr}[s]$ by Canonical Forms. By Lemma 8, 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.

□

TR Lemma G (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 $e \mapsto e'$.

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

SS-E-Concat-Left. Suppose $e = \mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)$ and $e_1 \mapsto e_1'$. By inversion of S-T-Concat, $\emptyset \vdash e_1 : \mathsf{stringin}[r_1]$ where $\emptyset \vdash e : \mathsf{stringin}[r_1 r_2]$. By induction, if $e_1 \mapsto e_1'$ then $\emptyset \vdash e_1' : \mathsf{stringin}[r_1]$. Therefore, $\emptyset \vdash \mathsf{rconcat}(e_1'; e_2) : \mathsf{stringin}[r_1 r_2]$.

SS-E-Concat-Right. Similar to SS-E-Concat-Left.

SS-E-Concat. Suppose $\emptyset \vdash \mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) : \mathsf{stringin}[r_1 r_2]$ and $\mathsf{rconcat}(\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[s_1 s_2]$. Then by inversion $\emptyset \vdash \mathsf{rstr}[s_1] : \mathsf{stringin}[r_1]$ and similarly for $\mathsf{rstr}[s_2]$. Therefore, $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$ from which it follows by Lemma 8 that $s_1 s_2 \in \mathcal{L}\{r_1 r_2\}$. Therefore, $\emptyset \vdash \mathsf{rstr}[s_1 s_2] : \mathsf{stringin}[r_1 r_2]$ by S-T-Rstr.

S-E-Case-Left. Suppose that $e = \mathsf{rstrcase}(e_1; e_2; x, y.e_3)$ and also that $e \mapsto \mathsf{rstrcase}(e_1'; e_2; x, y.e_3)$ and $\emptyset \vdash e : \mathsf{stringin}[r]$. By inversion of S-T-Case:

$$\emptyset \vdash e_1 : \mathsf{stringin}[r] \tag{7}$$
$$\emptyset \vdash e_2 : \sigma \tag{8}$$
$$x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma \tag{9}$$

By (7) and the assumption that $e_1 \mapsto e_1'$, it follows by induction that $\emptyset \vdash e_1' : \mathsf{stringin}[r]$. This fact together with (8) and (9) implies by S-T-Case that $\emptyset \vdash \mathsf{rstrcase}(e_1'; e_2; x, y.e_3) : \sigma$.

SS-E-Case-Right. We have that $e = \mathsf{rstrcase}(e_1; e_2; x, y.e_3)$, Suppose $e \mapsto \mathsf{rstrcase}(e_1; e_2'; x, y.e_3)$ and $\emptyset \vdash e : \mathsf{stringin}[r]$. By inversion of S-T-Case:

$$\emptyset \vdash e_1 : \mathsf{stringin}[r] \tag{10}$$
$$\emptyset \vdash e_2 : \sigma \tag{11}$$
$$x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma \tag{12}$$

By (11) and the assumption that $e_2 \mapsto e_2'$, it follows by induction that $\emptyset \vdash e_2' : \mathsf{stringin}[r]$. This fact together with (10) and (12) implies by S-T-Case that $\emptyset \vdash \mathsf{rstrcase}(e_1; e_2'; x, y.e_3) : \sigma$.

SS-E-Case-ϵ-Val. Suppose:

$$e = \mathsf{rstrcase}(-; e_2; -)$$
$$\emptyset \vdash e : \sigma$$
$$e \mapsto e_2$$

By inversion of S-T-Case, $e_2 : \sigma$.

SS-E-Case-Concat. Suppose that $e = \mathsf{rstrcase}(\mathsf{rstr}[as]; e_2; x, y.e_3) \mapsto [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3$ and that $\emptyset \vdash e : \sigma$. By inversion of S-T-Case:

$$\emptyset \vdash \mathsf{rstr}[as] : \mathsf{stringin}[r] \tag{13}$$
$$\emptyset \vdash \mathsf{rstr}[e_2] : \sigma \tag{14}$$
$$x : \mathsf{stringin}[\mathsf{lhead}(r)], y : \mathsf{stringin}[\mathsf{ltail}(r)] \vdash e_3 : \sigma \tag{15}$$

We know that $as \in \mathcal{L}\{r\}$ by (13) and inversion of S-T-Rstr. Therefore, $a \in \mathcal{L}\{\mathsf{lhead}(r)\}$ by definition of lhead. Furthermore, $\mathsf{ltail}(r) = ...|\delta_a r|...$ by definition of ltail. Note that $s \in \mathcal{L}\{\delta_a r\}$ by definition of the derivative, and so $s \in \mathcal{L}\{\mathsf{ltail}(r)\}$

From these facts about a and s we know by S-T-Rstr that $\emptyset \vdash \mathsf{rstr}[a] : \mathsf{stringin}[\mathsf{lhead}(r)]$ and $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[\mathsf{lhead}(r)]$. It follows by (15) that $\emptyset \vdash [\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 : \sigma$.

Cases SS-E-Replace-Left, SS-E-Replace-Right, SS-E-Check-StepLeft, SS-E-SafeCoerce-Step, SS-E-Check-StepRight. At this point the method for handling compatibility cases is clear; therefore, we elide these cases.

Case SS-E-Replace.

Suppose $e = \mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[\mathsf{subst}(r; s_1; s_2)]$. Assume $\emptyset \vdash e : \mathsf{stringin}[r']$ for $r' = \mathtt{lreplace}(r, r_1, r_2)$. Then by inversion of S-T-Replace:

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

from which follows that $s_1 \in \mathcal{L}\{r_1\}$ and $s_2 \in \mathcal{L}\{r_2\}$. Therefore, $\mathsf{subst}(r; s_1; s_2) \in \mathcal{L}\{\mathtt{lreplace}(r, r_1, r_2)\}$ by Theorem 7. It is finally derivable by S-T-Rstr that:

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

Case SS-E-SafeCoerce. Suppose that $\mathsf{rcoerce}[r](s_1) \mapsto \mathsf{rstr}[s_1]$ and that $\emptyset \vdash \mathsf{rcoerce}[r](s_1) : \mathsf{stringin}[r]$. By inversion of S-T-SafeCoerce we know that $s \in \mathcal{L}\{r\}$. Therefore, $\emptyset \vdash s : \mathsf{stringin}[r]$.

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

Case SS-E-Check-NotOk. Suppose $\mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) \mapsto e_2$, $s \notin \mathcal{L}\{r\}$, and $\emptyset \vdash \mathsf{rcheck}[r](\mathsf{rstr}[s]; x.e_1; e_2) : \sigma$. By inversion of S-T-Check, $\emptyset \vdash e_2 : \sigma$.

□

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

Proof. Follows directly from progress and preservation. □

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

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

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

Proof. We proceed by structural induction on e.

Case $e = \lambda x.e_1$. The only applicable rule is S-E-Abs, so $v = \lambda x.e_1$. Note that $\lambda x.e_2 \mapsto^* \lambda x.e_2$ by RT-Refl.

Case $e = e_1(e_2)$. The only applicable rule is S-E-App. By inversion, we establish that the following:

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

From which it follows by induction that:

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

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

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

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

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

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

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

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

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

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

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

from which it follows by induction that:

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

Note that the following rule is derivable:

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

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

Now consider the other case where $\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \Downarrow v$ was finally derived by S-E-Case-Concat. By inversion, $e_1 \Downarrow \mathsf{rstr}[as]$ and

$[\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \Downarrow v$. From these facts it follows by induction that $e_1 \mapsto^* \mathsf{rstr}[as]$ and $[\mathsf{rstr}[a], \mathsf{rstr}[s]/x, y]e_3 \mapsto^* v$.

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

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

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

From which it follows by induction that:

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

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

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

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

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

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

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

The following rule is derivable:

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

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

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

Note that the rule following is derivable:

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

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

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

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

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

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

$\square$

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

Proof. The proof proceeds by structural induction on e.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The cases for coercion and checking are straightforward. □

2.4 Extension of Safety for Small Step Semantics

Theorem 11 (Type Safety)**.** *If* $\emptyset \vdash e : \sigma$ *then* $e \Downarrow v$ *and* $\emptyset \vdash v : \sigma$*.*

Proof. If $\emptyset \vdash e : \sigma$ then $e \mapsto^* v$ for v val by termination. Therefore, $e \Downarrow v$ by part 2 of the semantic correspondence theorem.

Since $\emptyset \vdash e : \sigma$ and $e \mapsto^* v$, it follows that $\emptyset \vdash v : \sigma$ by type safety for the small step semantics. □

2.4.1 The Security Theorem

Theorem 12 (Correctness of Input Sanitation for λ_{RS})**.** *If* $\emptyset \vdash e : \mathsf{stringin}[r]$ *and* $e \Downarrow \mathsf{rstr}[s]$ *then* $s \in \mathcal{L}\{r\}$*.*

Proof. If $\emptyset \vdash e : \mathsf{stringin}[r]$ and $e \Downarrow \mathsf{rstr}[s]$ then $\emptyset \vdash \mathsf{rstr}[s] : \mathsf{stringin}[r]$ by Type Safety. By inversion of S-T-Rstr, $s \in \mathcal{L}\{r\}$. □

3 Proofs of Lemmas and Theorems About λ_P

Theorem 13 (Safety for λ_P)**.** *If* $\emptyset \vdash \iota : \tau$ *then* $\iota \Downarrow \dot{v}$ *and* $\emptyset \vdash \dot{v} : \tau$*.*

We can also define canonical forms for regular expressions and strings in the usual way:

Lemma 14 (Canonical Forms for Target Language)**.**

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

4 Proofs and Lemmas and Theorems About Translation

Theorem 15 (Translation Correctness)**.** *If $\Theta \vdash e : \sigma$ then there exists an ι such that $[\![e]\!] = \iota$ and $[\![\Theta]\!] \vdash \iota : [\![\sigma]\!]$. Furthermore, $e \Downarrow v$ and $\iota \Downarrow \dot{v}$ such that $[\![v]\!] = \dot{v}$.*

Proof. We present a proof by induction on the derivation that $\Theta \vdash e : \sigma$. we write $e \rightsquigarrow \iota$ as shorthand for the final property.

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

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

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

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

$$[\![\mathsf{rstr}[s]]\!] = \mathsf{string}s \tag{16}$$

$$[\![\mathsf{stringin}[r]]\!] = \mathsf{string} \tag{17}$$

$$[\![\emptyset]\!] = \emptyset \tag{18}$$

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

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

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

Case $e = \mathsf{rconcat}(e_1; e_2)$. This case is an obvious appeal to induction.

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

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

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

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

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

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

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

$\square$

Theorem 16 (Correctness of Input Sanitation for Translated Terms)**.** *If* $[\![e]\!] = \iota$ *and* $\emptyset \vdash e : \mathsf{stringin}[r]$ *then* $\iota \Downarrow \mathsf{str}[s]$ *for* $s \in \mathcal{L}\{r\}$*.*

Proof. By Theorem 15 and the rules given, $\iota \Downarrow \mathsf{str}[s]$ implies that $e \Downarrow \mathsf{rstr}[s]$. Theorem 12 together with the assumption that e is well-typed implies that $s \in \mathcal{L}\{r\}$. $\square$

5 String Substitution and Language Replacement

5.1 The Trivial Definition

5.2 An Automaton Construction

Insert Automaton stuff...

5.3 Toward a Precise Definition

References

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

List of Figures

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

Figure 1: Regular expressions over the alphabet Σ.

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

Figure 2: Syntax of λ_{RS}.

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

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

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

$$\textsc{S-T-Var}\quad \frac{x : \sigma \in \Psi}{\Psi \vdash x : \sigma}$$

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

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

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

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

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

$$\textsc{S-T-Replace}\quad \frac{\begin{array}{c}\Psi \vdash e_1 : \mathsf{stringin}[r_1] \qquad \Psi \vdash e_2 : \mathsf{stringin}[r_2] \\ \mathtt{lreplace}(r, r_1, r_2) = r'\end{array}}{\Psi \vdash \mathsf{rreplace}[r](e_1; e_2) : \mathsf{stringin}[r']}$$

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

$$\textsc{S-T-Check}\quad \frac{\Psi \vdash e_0 : \mathsf{stringin}[r_0] \qquad \Psi, x : \mathsf{stringin}[r] \vdash e_1 : \sigma \qquad \Psi \vdash e_2 : \sigma}{\Psi \vdash \mathsf{rcheck}[r](e_0; x.e_1; e_2) : \sigma}$$

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

$\boxed{e \Downarrow v}$

$$\textsc{S-E-Abs}\quad \frac{}{\lambda x.e \Downarrow \lambda x.e}$$

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

$$\textsc{S-E-RStr}\quad \frac{}{\mathsf{rstr}[s] \Downarrow \mathsf{rstr}[s]}$$

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

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

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

$$\textsc{S-E-Replace}\quad \frac{e_1 \Downarrow \mathsf{rstr}[s_1] \qquad e_2 \Downarrow \mathsf{rstr}[s_2] \qquad \mathsf{subst}(r; s_1; s_2) = s}{\mathsf{rreplace}[r](e_1; e_2) \Downarrow \mathsf{rstr}[s]}$$

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

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

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

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

$\boxed{e \ \mathsf{val}}$

$$\textsc{L-Val}\quad \frac{}{\lambda x : \tau . t \ \mathsf{val}}$$

$\boxed{e \mapsto e}$

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

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

$$\textsc{RT-Refl}\quad \frac{}{e \mapsto^* e} \qquad \textsc{RT-Trans}\quad \frac{e \mapsto^* e' \qquad e' \mapsto^* e''}{e \mapsto^* e''} \qquad \textsc{RT-Step}^2\quad \frac{e \mapsto^* e' \qquad e' \mapsto v}{e \mapsto^* v}$$

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

SS-E-RStr

$$\frac{}{\mathsf{rstr}[s]\ \mathsf{val}}$$

SS-E-Concat-Left

$$\frac{e_1 \mapsto e_1'}{\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1'; e_2)}$$

SS-E-Concat-Right

$$\frac{e_2 \mapsto e_2'}{\mathsf{rconcat}(e_1; e_2) \mapsto \mathsf{rconcat}(e_1; e_2')}$$

SS-E-Concat

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

SS-E-Case-Left

$$\frac{e_1 \mapsto e_1'}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \mapsto \mathsf{rstrcase}(e_1'; e_2; x, y.e_3)}$$

SS-E-Case-Right

$$\frac{e_2 \mapsto e_2'}{\mathsf{rstrcase}(e_1; e_2; x, y.e_3) \mapsto \mathsf{rstrcase}(e_1; e_2'; x, y.e_3)}$$

SS-E-Case-ϵ-val

$$\frac{}{\mathsf{rstrcase}(\mathsf{rstr}[\epsilon]; e_2; x.y.e_3) \mapsto e_2}$$

SS-E-Case-Concat

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

SS-E-Replace-Left

$$\frac{e_1 \mapsto e_1'}{\mathsf{rreplace}[r](e_1; e_2) \mapsto \mathsf{rreplace}[r](e_1'; e_2)}$$

SS-E-Replace-Right

$$\frac{e_2 \mapsto e_2'}{\mathsf{rreplace}[r](e_1; e_2) \mapsto \mathsf{rreplace}[r](e_1; e_2')}$$

SS-E-Replace

$$\frac{}{\mathsf{rreplace}[r](\mathsf{rstr}[s_1]; \mathsf{rstr}[s_2]) \mapsto \mathsf{rstr}[\mathsf{subst}(r; s_1; s_2)]}$$

SS-E-SafeCoerce-Step

$$\frac{e \mapsto e'}{\mathsf{rcoerce}[r](e) \mapsto \mathsf{rcoerce}[r](e')}$$

SS-E-SafeCoerce

$$\frac{}{\mathsf{rcoerce}[r](\mathsf{rstr}[s]) \mapsto \mathsf{rstr}[s]}$$

SS-E-Check-StepLeft

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

SS-E-Check-StepRight

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

SS-E-Check-Ok

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

SS-E-Check-NotOk

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

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

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

$$\text{P-T-VAR}\ \frac{x : \tau \in \Theta}{\Theta \vdash x : \tau} \qquad \text{P-T-ABS}\ \frac{\Theta, x : \tau_1 \vdash \iota_2 : \tau_2}{\Theta \vdash \lambda x.\iota_2 : \tau_1 \to \tau_2} \qquad \text{P-T-APP}\ \frac{\Theta \vdash \iota_1 : \tau_2 \to \tau \qquad \Theta \vdash \iota_2 : \tau_2}{\Theta \vdash \iota_1(\iota_2) : \iota}$$

$$\text{P-T-STRING}\ \frac{}{\Theta \vdash \mathsf{str}[s] : \mathsf{string}} \qquad \text{P-T-REGEX}\ \frac{}{\Theta \vdash \mathsf{rx}[r] : \mathsf{regex}} \qquad \text{P-T-CONCAT}\ \frac{\Theta \vdash \iota_1 : \mathsf{string} \qquad \Theta \vdash \iota_2 : \mathsf{string}}{\Theta \vdash \mathsf{concat}(\iota_1; \iota_2) : \mathsf{string}}$$

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

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

$$\text{P-T-CHECK}\ \frac{\Theta \vdash \iota_r : \mathsf{regex} \qquad \Theta \vdash \iota_1 : \mathsf{string} \qquad \Theta \vdash \iota_2 : \sigma \qquad \Theta \vdash \iota_3 : \sigma}{\Theta \vdash \mathsf{check}(\iota_r; \iota_1; \iota_2; \iota_3) : \sigma}$$

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

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

$$\textsc{P-E-Abs}\quad \frac{}{\lambda x.e \Downarrow \lambda x.e}$$

$$\textsc{P-E-App}\quad \frac{\iota_1 \Downarrow \lambda x.\iota_3 \qquad \iota_2 \Downarrow \dot{v}_2 \qquad [\dot{v}_2/x]\iota_3 \Downarrow \dot{v}_3}{\iota_1(\iota_2) \Downarrow \dot{v}_3}$$

$$\textsc{P-E-Str}\quad \frac{}{\mathsf{str}[s] \Downarrow \mathsf{str}[s]}$$

$$\textsc{P-E-Rx}\quad \frac{}{\mathsf{rx}[r] \Downarrow \mathsf{rx}[r]}$$

$$\textsc{P-E-Concat}\quad \frac{\iota_1 \Downarrow \mathsf{str}[s_1] \qquad \iota_2 \Downarrow \mathsf{str}[s_2]}{\mathsf{concat}(\iota_1; \iota_2) \Downarrow \mathsf{str}[s_1 s_2]}$$

$$\textsc{P-E-Case-}\epsilon\quad \frac{\iota_1 \Downarrow \mathsf{str}[\epsilon] \qquad \iota_2 \Downarrow \dot{v}_2}{\mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3) \Downarrow \dot{v}_2}$$

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

$$\textsc{P-E-Replace}\quad \frac{\iota_1 \Downarrow \mathsf{rx}[r] \qquad \iota_2 \Downarrow \mathsf{str}[s_2] \qquad \iota_3 \Downarrow \mathsf{str}[s_3] \qquad \mathsf{subst}(r; s_2; s_3) = s}{\mathsf{replace}(\iota_1; \iota_2; \iota_3) \Downarrow \mathsf{str}[s]}$$

$$\textsc{P-E-Check-OK}\quad \frac{\iota_r \Downarrow \mathsf{rx}[r] \qquad \iota \Downarrow \mathsf{str}[s] \qquad s \in \mathcal{L}\{r\} \qquad \iota_1 \Downarrow \dot{v}_1}{\mathsf{check}(\iota_r; \iota; \iota_1; \iota_2) \Downarrow \dot{v}_1}$$

$$\textsc{P-E-Check-NotOK}\quad \frac{\iota_r \Downarrow \mathsf{rx}[r] \qquad \iota \Downarrow \mathsf{str}[s] \qquad s \notin \mathcal{L}\{r\} \qquad \iota_2 \Downarrow \dot{v}_2}{\mathsf{check}(\iota_r; \iota; \iota_1; \iota_2) \Downarrow \dot{v}_2}$$

Figure 9: Big step semantics for λ_P

.

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

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

$$\textsc{SP-E-Rx}\ \frac{}{\mathsf{rx}[r] \Downarrow \mathsf{rx}[r]} \qquad \textsc{SP-E-Concat}\ \frac{\iota_1 \Downarrow \mathsf{str}[s_1] \qquad \iota_2 \Downarrow \mathsf{str}[s_2]}{\mathsf{concat}(\iota_1; \iota_2) \Downarrow \mathsf{str}[s_1 s_2]} \qquad \textsc{SP-E-Case-}\epsilon\ \frac{\iota_1 \Downarrow \mathsf{str}[\epsilon] \qquad \iota_2 \Downarrow \dot{v}_2}{\mathsf{strcase}(\iota_1; \iota_2; x, y.\iota_3) \Downarrow \dot{v}_2}$$

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

$$\textsc{SP-E-Replace}\ \frac{\iota_1 \Downarrow \mathsf{rx}[r] \qquad \iota_2 \Downarrow \mathsf{str}[s_2] \qquad \iota_3 \Downarrow \mathsf{str}[s_3] \qquad \mathsf{subst}(r; s_2; s_3) = s}{\mathsf{replace}(\iota_1; \iota_2; \iota_3) \Downarrow \mathsf{str}[s]}$$

$$\textsc{SP-E-Check-OK}\ \frac{\iota_r \Downarrow \mathsf{rx}[r] \qquad \iota \Downarrow \mathsf{str}[s] \qquad s \in \mathcal{L}\{r\} \qquad \iota_1 \Downarrow \dot{v}_1}{\mathsf{check}(\iota_r; \iota; \iota_1; \iota_2) \Downarrow \dot{v}_1}$$

$$\textsc{SP-E-Check-NotOK}\ \frac{\iota_r \Downarrow \mathsf{rx}[r] \qquad \iota \Downarrow \mathsf{str}[s] \qquad s \notin \mathcal{L}\{r\} \qquad \iota_2 \Downarrow \dot{v}_2}{\mathsf{check}(\iota_r; \iota; \iota_1; \iota_2) \Downarrow \dot{v}_2}$$

Figure 10: Small step semantics for λ_P

.

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

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

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

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

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

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

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

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

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

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