TODO-nrf spell check.
TODO-nrf Include functions (app, abs)?
TODO-nrf Change icheck in eval rule to Python.

1. Introduction

In web applications and other settings, incorrect input sanitation often causes security vulnerabilities. For this reason, frameworks often provide methods for sanitizing user input. When these methods are insufficient or unavailable, developers often write develop custom input sanitation algorithms. In both cases, input sanitation techniques are ultimately implemented using the language's regex library.

Formally, input sanitation is the problem of ensuring that an arbitrary string is converted into a safe form before potentially unsafe use. For example, consider SQL injection attacks. To prevent such attacks, we might ensure that any string used as input to a query does not contain unescaped SQL common sequences.

This appendix presents a type system called λ_{CS} for ensuring that input sanitation algorithms are implemented correctly with respect to use site specifications.

Unlike frameworks provided by general purpose languages such as Haskell and Ruby, our type system provides a *static* guarantee that input is always properly sanitized before use. We achieve this by moving portions of the regular expression library into the type system. Therefore, type safety relies upon several closure and decidability results about regular languages.

Although Ur/Web achieves a similar safety guarantee, its type system is significantly more complicated than the system defined below. We believe this simplicity demonstrates the power of extensibility. Instead of introducing new abstractions based upon subtle analyses and requiring the adoption of new programming languages, extension designers may capture natural programming idioms directly. TODO-nrf this seems unsubstantiated or dangerously vague.

Finally, XDuce typechecks XML. Like our work, XDuce relies upon some properties of regular expression to establish soundness and copmleteness results. Unlike XDuce, our work is motivated by input sanitation and therefore considers arbitrary strings (as opposed to tree-structured XML documents). Furthermore, our static treatment of the ubiquitous "filter" function for regular expression is novel.

An outline of this appendix follows:

- In §2, we define the type system's static and dynamic semantics.
- Section 3 recalls some classical results about regular expressions and presents a type safety proof for λ_{CS} based upon these properties.
- Finally, §4 discusses our implemention of λ_{CS} as a type system extension within the Ace programming language.

$\langle r \rangle ::= \epsilon \mid . \mid a \mid r \cdot r \mid r + r \mid r*$	Regex ($a \in \Sigma$).
$\langle t \rangle$ ::=	terms:
`filter`$($`string_in`$[r], t)$	filter substrings
$\mid$ $[$`string_in`$[r]](t)$	safe conversion
$\mid$ `dconvert`$[$`string_in`$[r]](t)$	unsafe conversion
$\langle iv \rangle$ ::=	internal values:
's'	internal string
$\mid$ `ifilter`$(r,$ `istring`$(s))$	internal filter
$\langle v \rangle$ ::=	values:
"s"	string
$\langle T \rangle$ ::=	types:
`string`	Strings
$\mid$ `string_in`$[r]$	Regular language strings
$\langle \Gamma \rangle ::= \emptyset \mid \Gamma, x : T$	typing context

Figure 1. Syntax of λ_{CS}

2. Definition of λ_{CS}

The λ_{CS} language is characterized by a type of strings indexed by regular expressions, together with operations on such strings which correspond to common input sanitation patterns.

This section presents the grammar and semantics of λ_{CS}. The semantics are defined in terms of an internal language with at least strings and a regex filter function. These constraints are captured by the internal term valuations (`ival`). The internal language does not necessarily need a regex filter function because any dynamic conversion is easily definable using a combination of filters and safe casting.

The λ_{CS} language gives static semantics for common regular expression library functions. In this treatment, we include concatenation and filtering. The `filter` function removes all instances of a regular expression in a string, while concatenation ($+$) concatenates two strings.

$$\frac{}{\texttt{string} \equiv \texttt{string\_in}[.*]} \quad \textsc{T-Equiv-Top}$$

$$\frac{\mathcal{L}\{r\} = \mathcal{L}\{r'\}}{\texttt{string\_in}[r] \equiv \texttt{string\_in}[r']} \quad \textsc{T-Equiv-string\_in}$$

$$\frac{e \in \mathcal{L}\{r\}}{\Gamma \vdash e : \texttt{string\_in}[r]} \quad \textsc{T-Equiv-Include}$$

$$\frac{\Gamma \vdash e : \texttt{string}}{\Gamma \vdash e : \texttt{string\_in}[e]} \quad \textsc{T-Minimum}$$

$$\frac{\Gamma \vdash e_1 : \texttt{string\_in}[r_1] \qquad \Gamma \vdash e_2 : \texttt{string\_in}[r_2]}{\Gamma \vdash e_1 + e_2 : \texttt{string\_in}[r_1 \cdot r_2]} \quad \textsc{T-Concat}$$

$$\frac{\Gamma \vdash e : \texttt{string\_in}[r'] \qquad \mathcal{L}\{r'\} \subseteq \mathcal{L}\{r\}}{\Gamma \vdash [\texttt{string\_in}[r]](e) : \texttt{string\_in}[r]} \quad \textsc{T-Convert}$$

$$\frac{\Gamma \vdash e : \texttt{string\_in}[r']}{\Gamma \vdash \texttt{dconvert}[\texttt{string\_in}[r]](e) : \texttt{string\_in}[r]} \quad \textsc{T-Dconvert (optional)}$$

$$\frac{\Gamma \vdash t : \texttt{string\_in}[r]}{\Gamma \vdash \texttt{filter}(\texttt{string\_in}[r'], t) : \texttt{string\_in}[(r \backslash r') + \emptyset]} \quad \textsc{T-Filter}$$

Figure 2. Typing relation for λ_{CS}

$$\frac{}{\text{“}s\text{”} : \texttt{string\_in}[r]\ \texttt{val}} \qquad \textsc{E-strinval}$$

$$\frac{}{\text{‘}e\text{’} : \texttt{istring}\ \texttt{ival}} \qquad \textsc{E-ival}$$

$$\frac{e\texttt{ival}}{\texttt{ifilter}(r, e)\ \texttt{ival}} \qquad \textsc{E-ifilterval}$$

$$\frac{e\ \texttt{val}}{e : \texttt{string} \rightsquigarrow \text{‘}e\text{’}} \qquad \textsc{E-string}$$

$$\frac{e\ \texttt{val}}{e : \texttt{string\_in}[r] \rightsquigarrow \text{‘}e\text{’}} \qquad \textsc{E-string\_in}$$

$$\frac{}{e_1 + e_2 : \texttt{string\_in}[r_1 + r_2] \rightsquigarrow \text{‘}e_1\text{’} + \text{‘}e_2\text{’}} \qquad \textsc{E-concat}$$

$$\frac{e\ \texttt{val}}{\texttt{filter}(\texttt{string\_in}[r], e) : \texttt{string\_in}[r'] \Rightarrow \texttt{rl\_filter}(r, e)} \qquad \textsc{E-filterval}$$

$$\frac{e : T \Rightarrow e'}{\texttt{filter}(\texttt{string\_in}[r], e) : \texttt{string\_in}[r'] \Rightarrow \texttt{filter}(\texttt{string\_in}[r], e')} \qquad \textsc{E-filter}$$

$$\frac{e : T \Rightarrow e'}{\texttt{dconvert}[\texttt{string\_in}[r]](e) : \texttt{string\_in}[r'] \Rightarrow \texttt{dconvert}[\texttt{string\_in}[r]](e')} \qquad \textsc{E-dconvertval (optional)}^{1}$$

$$\frac{e\ \texttt{val}}{\texttt{dconvert}[\texttt{string\_in}[r]](e) : \texttt{string\_in}[r'] \rightsquigarrow \texttt{ifilter}(r, \text{‘}e\text{’})} \qquad \textsc{E-dconvert (optional)}^{2}$$

Figure 3. Operational semantics for λ_{CS}

2.1 Typing

The $\texttt{string\_in}[r]$ type is parameterized by regular expressions; if $e : \texttt{string\_in}[r]$, then $e \in r$. Mapping from an arbitrary `string` to a $\texttt{string\_in}[r]$ requires defining a algorithm – in terms of filter – for converting a $\texttt{string\_in}[.*]$ into a $\texttt{string\_in}[r]$. The static semantics of the language defines the types of operations on regular expressions in terms of well-understood properties about regular lanugages; we recall these properties in section 3.

2.2 Dynamics

There are two evaluation judgements: $e : T \Rightarrow e'$ and $e : T \rightsquigarrow i$. The $\Rightarrow$ relation is between λ_{CS} expressions, while the $\rightsquigarrow$ relation is a mapping from λ_{CS} expressions into internal language expressions i such that i `ival`.

Safety of the evaluation relation depends upon an injective mapping from λ_{CS} types info internal language types. This relation, h, is defined below.

Definition 1 (Type translation)**.** The type translation function $h : Type \rightarrow IType$ is defined as follows:

- $\forall r.h(\texttt{string\_in}[r]) = \texttt{istring}$
- $h(\texttt{string}) = \texttt{istring}$

3. Type Safety

The type safety proof relies upon some established results about regular languages. The important definitions and theorems follow.

3.1 Properties of Regular Languages

3.2 Type Safety Proof

Theorem 2 (Preservation)**.** *Let T be a type in λ_{CS} and $h(T) = \sigma$ the corresponding type in the internal language. For all terms e:*

- *If $e : T$ and $e : T \rightsquigarrow i$ then $i : \sigma$ for some σ.*
- *If $e : T$ and $e : T \Rightarrow e'$ then $e' : T$.*

Proof. The proof is a straightforward induction on the derivation of the combined evaluation relation. □

Theorem 3 (Progress)**.** *Let T be a type in λ_{CS} and σ a type in the internal language. For all terms e, if $e : T$ then $e : T \Rightarrow^* e' \rightsquigarrow^* i$ where i*`ival`*.*

Proof. By induction on the derivation of $e : T$. □

4. Implementation in Ace