A. Definitions

Definition 2 (Regular Expressions). The set $\mathcal{Y}_{\Sigma}$ of regular expressions over an ordered alphabet $\Sigma$ is recursively defined as follows.

- Every $y_j \in \Sigma \cup \{\epsilon, \\{, \}, \\], \wedge, \\|\}$, every range $y_j = l_{\min} \rightarrow l_{\max}$, where $l_{\min}, l_{\max} \in \Sigma$ and $l_{\min} < l_{\max}$, and their disjunction $|y_1 \cdots y_k|$ are regular expressions.
- If $y_1, \ldots, y_k \in \mathcal{Y}_{\Sigma}$ are regular expressions, so are the concatenation $y = y_1 \cdots y_k$, the disjunction $y = y_1 \vee \cdots \vee y_k$, $y = y_1 \wedge y_2$, and the repetitions $y = y_1^*$, $y = y_1^+$, $y = y_1^{\omega}$, and $y = y_1{[l, u]}$, where $l, u \in \mathbb{N}$ and $l \leq u$.

We now define the syntax tree, the parse tree, and the matching lists for a regular expression $y$ and a string $x \in \Sigma^*$. The shorthand $(y \rightarrow T_1, \ldots, T_k)$ denotes the tree $T = (V, E, \Gamma, \leq)$ with root node $v_0 \in V$ labeled with $\Gamma(v_0) = y$ and subtrees $T_1, \ldots, T_k$. The order $\leq$ maintains the subtree orderings $\leq_i$ and defines the root node as the minimum over the set $V$ and $v' \leq v''$ for all $v' \in V_i$ and $v'' \in V_j$, where $i < j$.

Definition 3 (Syntax Tree). The abstract syntax tree $T^y_{\text{syn}}$ for a regular expression $y$ is recursively defined as follows. Let $T^y_{\text{syn}} = (V^y_{\text{syn}}, E^y_{\text{syn}}, \Gamma^y_{\text{syn}}, \leq_y_{\text{syn}})$ be the syntax tree of the subexpression $y_j$.

- If $y \in \Sigma \cup \{\epsilon, \wedge, \|\}$, or if $y = l_{\min} \rightarrow l_{\max}$, where $l_{\min}, l_{\max} \in \Sigma$, we define $T^y_{\text{syn}} = (y \rightarrow \emptyset)$.
- If $y = (y_1)$, where $y_1 \in \mathcal{Y}_{\Sigma}$, we define $T^y_{\text{syn}} = T^y_1$.
- If $y = y_1^*$, $y = y_1^+$, $y = y_1^{\omega}$, or if $y = y_1{[l, u]}$, where $y_1 \in \mathcal{Y}_{\Sigma}$, $l, u \in \mathbb{N}$, and there exist no $y', y'' \in \mathcal{Y}_{\Sigma}$ such that $y_1 = y'y''$ or $y_1 = y'y''$, we define $T^y_{\text{syn}} = (y \rightarrow T^y_1)$.

Definition 4 (Parse Tree and Matching List). Given a syntax tree $T^y_{\text{syn}} = (V^y_{\text{syn}}, E^y_{\text{syn}}, \Gamma^y_{\text{syn}}, \leq_y_{\text{syn}})$ of a regular expression $y$ with nodes $v \in V^y_{\text{syn}}$ and a string $x \in L(y)$, a parse tree $T^y_{\text{par}}$ and the matching lists $M^y_{\text{par}}(v)$ for each $v \in V^y_{\text{syn}}$ are recursively defined as follows. Let $T^y_{\text{par}} = (V^y_{\text{par}}, E^y_{\text{par}}, \Gamma^y_{\text{par}}, \leq_y_{\text{par}})$ be the parse tree and $T^y_{\text{syn}} = (V^y_{\text{syn}}, E^y_{\text{syn}}, \Gamma^y_{\text{syn}}, \leq_y_{\text{syn}})$ the syntax tree of the subexpression $y_j$.

- If $y = x$ and $x \in \Sigma \cup \{\epsilon\}$, we define $M^y_{\text{par}}(v_0) = \{x\}$ and $T^y_{\text{par}} = (y \rightarrow \emptyset)$.
- If $y = \cdot$ and $x \in \Sigma$, $y = l_{\min} \rightarrow l_{\max}$ and $l_{\min} \leq x \leq l_{\max}$, or if $y \in \{\{, \}, \wedge, \|\}$ and $x$ is either a non-whitespace character (everything but spaces, tabs, and line breaks), a word character (letters, digits, and underscores), a character in [.,-,#,+] or a word character, or a digit,
matching list

M

ion 4 by

v

each

we denote the set of all parse trees and the unions of

v

M

T

If

x

∈

V

syn

and

\( T_{\text{par}} = (y \mapsto T_{\text{par}}^y) \).

• If \( y = (y_1) \) and \( x \in \Sigma^+ \), we define

\( M_{\text{syn}}(v) = \{x\} \) for all \( v \in V_{\text{syn}}^\Sigma \) and

\( T_{\text{par}} = T_{\text{par}}^y \).

• If \( y = y^+_1 \), \( x = x_1 \ldots x_k \), and \( k \geq 0 \), or if

\( y = y^+_1 \), and \( k > 0 \), or if

\( y = y_1 \{l, u\} \), and \( l \leq k \leq u \), or if

\( y = y_1 \{\} \), and \( k = l \),

where \( x_i \in \Sigma^+ \), and there exist no \( y', y'' \in \mathcal{Y}_\Sigma \)

such that \( y_1 = y'|y'' \) or \( y_1 = y'y'' \), we define

\( M_{\text{syn}}(v) = \{x\} \), if \( v = v_0 \)

\( \bigcup_{i=1}^k M_{\text{syn}}(v) \), if \( v \in V_{\text{syn}} \), and

\( T_{\text{par}} = (y \mapsto T_{\text{par}}^1, \ldots, T_{\text{par}}^k) \).

• If \( y = y_1 \ldots y_k \), \( x = x_1 \ldots x_k \),

where \( x_i \in \Sigma^+ \), and there exist no \( y', y'' \in \mathcal{Y}_\Sigma \)

such that \( y_j = y'|y'' \) or \( y_j = y'y'' \), we define

\( M_{\text{syn}}(v) = \{x\} \), if \( v = v_0 \)

\( M_{\text{syn}}(v) \), if \( v \in V_{\text{syn}} \), and

\( T_{\text{par}} = (y \mapsto T_{\text{par}}^1, \ldots, T_{\text{par}}^k) \).

• If \( y = [y_1 \ldots y_k] \), \( x \in \Sigma^+ \)

and there exist no \( y', y'' \in \mathcal{Y}_\Sigma \) such

that \( y_j = y'|y'' \), or if

\( y = [y_1 \ldots y_k] \), \( x \in \Sigma^+ \)

and there exist no \( y', y'' \in \mathcal{Y}_\Sigma \) such

that \( y_j = y'y'' \), we define

\( M_{\text{syn}}(v) = \{x\} \), if \( v = v_0 \)

\( M_{\text{syn}}(v) \), if \( v \in V_{\text{syn}} \), and

\( \emptyset \), otherwise

\( T_{\text{par}} = (y \mapsto T_{\text{par}}^x) \).

If \( x \notin L(y) \), that is, no parse tree can be derived by
the specification above, the empty sets \( M_{\text{syn}}(v) = \emptyset \)
for all \( v \in V_{\text{syn}} \) and \( T_{\text{par}} = \emptyset \) are returned. Otherwise,
we denote the set of all parse trees and the unions of
all matching lists for each \( v \in V_{\text{syn}} \) satisfying Defini-
tion 4 by \( T_{\text{par}}^x \) and \( M_{\text{syn}}(v) \), respectively. Finally, the
matching list \( M_{\text{syn}}(v) \) for a set of strings \( x \) for node
\( v \in V_{\text{syn}} \) is defined as \( M_{\text{syn}}(v) = \bigcup_{x \in x} M_{\text{syn}}(v) \).

B. Used Features

Let \( M \) be a matching list and \( M_{\Sigma} \) be the set of char-
acters in \( \Sigma \) which appear in the matching list \( M \). The
list of binary and continuous features, used to train
REx-SVM is shown in Table 1.
Table 1. Important features used to train REx-SVM.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon \in M$</td>
<td>Matching list contains the empty string?</td>
</tr>
<tr>
<td>$\forall x \in M \</td>
<td>x = 1$</td>
</tr>
<tr>
<td>$\exists i \in \mathbb{N} \forall x \in M \</td>
<td>x = i$</td>
</tr>
<tr>
<td>$\frac{</td>
<td>M \cap {A, \ldots, Z}</td>
</tr>
<tr>
<td>$\frac{</td>
<td>M \cap {a, \ldots, z}</td>
</tr>
<tr>
<td>$\frac{</td>
<td>M \cap {0, \ldots, 9}</td>
</tr>
<tr>
<td>$\frac{</td>
<td>M \cap {A, \ldots, F}</td>
</tr>
<tr>
<td>$\frac{</td>
<td>M \cap {g, \ldots, z}</td>
</tr>
<tr>
<td>$\forall x \in M \exists x \notin {A, \ldots, Z}$</td>
<td>No characters of A–Z in the matching list?</td>
</tr>
<tr>
<td>$\forall x \in M \exists x \notin {a, \ldots, z}$</td>
<td>No characters of a–z in the matching list?</td>
</tr>
<tr>
<td>$\forall x \in M \exists x \notin {0, \ldots, 9}$</td>
<td>No characters of 0–9 in the matching list?</td>
</tr>
<tr>
<td>$\forall x \in M \exists x \notin {a, \ldots, f}$</td>
<td>No characters of a–f in the matching list?</td>
</tr>
<tr>
<td>$\forall x \in M \exists x \notin {A, \ldots, F}$</td>
<td>No characters of A–F in the matching list?</td>
</tr>
<tr>
<td>$</td>
<td>M \cap {-, /, ?, =, :, @}</td>
</tr>
<tr>
<td>$\forall x \in M \</td>
<td>x \geq 1 \land</td>
</tr>
<tr>
<td>$\forall x \in M \</td>
<td>x \geq 6 \land</td>
</tr>
<tr>
<td>$\forall x \in M \</td>
<td>x \geq 11 \land</td>
</tr>
<tr>
<td>$\forall x \in M \</td>
<td>x \geq 20$</td>
</tr>
<tr>
<td>$</td>
<td>M</td>
</tr>
</tbody>
</table>