String Abstractions
Given a string manipulating program, *string analysis* determines *all possible values* that a string expression can take during any program execution.

Using string analysis we can verify properties of string manipulating programs:
- For example, we can identify all possible input values of sensitive functions in a web application and then check whether inputs of sensitive functions can contain attack strings.
Configurations/Transitions are represented using word equations.

Word equations are represented/approximated using (aligned) *multi-track DFAs* which are closed under intersection, union, complement and projection.

Operations required for reachability analysis (such as equivalence checking) are computed on DFAs.
Let $X$ (the first track), $Y$ (the second track), be two string variables

- $\lambda$: a padding symbol that appears only on the tail of each track (aligned)

- A multi-track automaton that encodes $X = Y$.txt

(a,a), (b,b) …
Regular Abstraction

- Compute the post-conditions of statements

Given a multi-track automata M and an assignment statement: \( X := \text{sexp} \)

\[ \text{Post}(M, X := \text{sexp}) \text{ denotes the post-condition of } X := \text{sexp} \text{ with respect to } M \]

\[
\text{Post}(M, X := \text{sexp}) = (\exists X, M \cap \text{CONSTRUCT}(X' = \text{sexp, } +))[X/X']
\]
We implement a symbolic forward reachability computation using the post-condition operations.

The forward fixpoint computation is not guaranteed to converge in the presence of loops and recursion.

We use an automata based widening operation to over-approximate the fixpoint.

- Widening operation over-approximates the union operations and accelerates the convergence of the fixpoint computation.
The alphabet of an $n$-track automaton is $\Sigma^n$
- The size of multi-track automata could be huge during computations
- On the other hand, we may carry more information than we need to verify the property

More Abstractions:
- We propose *alphabet abstraction* to reduce $\Sigma$
- We propose *relation abstraction* to reduce $n$
Alphabet Abstraction

- Select a subset of alphabet characters ($\Sigma'$) to analyze distinctly and merge the remaining alphabet characters into a special symbol (◆)

- For example:
  Let $\Sigma=${$<, a, b, c$} and $\Sigma'=${$<$}, $L(M) = a < b^+$, we have:
  
  $\alpha_{\Sigma,\Sigma'}(M) = M_\alpha$ and $\gamma_{\Sigma,\Sigma'}(M_\alpha) = M_\gamma$, where

  $L(M_\alpha) = ◆ < ◆^+$, and $L(M_\gamma) = (a|b|c)<(a|b|c)^+$
We use an *alphabet transducer* $M_{\Sigma,\Sigma'}$ to construct abstract automata

- $\alpha$ denotes any character in $\Sigma'$
- $\beta$ denotes any character in $\Sigma \setminus \Sigma'$

![Diagram of alphabet transducer](image)
An Example of Alphabet Abstraction
\( \Sigma = \{<, a, b, c\} \) and \( \Sigma' = \{<\} \)
An Example of Alphabet Abstraction
$\Sigma=\{<, a, b, c\}$ and $\Sigma'=\{<\}$
Apply Alphabet Abstraction

1: <?php
2: $www = $_GET["www"];
3: $l_otherinfo = "URL";
4: $www = str_replace(<,,"",$www);
5: echo "<td>" . $l_otherinfo . ": " . $www . "</td>";
6: ?>

- Consider the above example, choosing $\Sigma' = \{<, s\}$ (instead of all ASCII characters) is sufficient to conclude that the echo string does not contain any substring that matches "<script"
Consider the following abstraction: We map all the symbols in the alphabet to a single symbol.

The automaton we generate with this abstraction will be a unary automaton (an automaton with a unary alphabet).

The only information that this automaton will give us will be the length of the strings.

So alphabet abstraction corresponds to length abstraction.
Select sets of string variables to analyze relationally (using multi-track automata), and analyze the rest independently (using single-track automata).

For example, consider three string variables \( n_1, n_2, n_3 \).

- Let \( \chi = \{\{n_1, n_2\}, n_3\} \) and \( \chi' = \{\{n_1\}, \{n_2\}, \{n_3\}\} \)
- Let \( M = \{M_{1,2}, M_3\} \) that consists of a 2-track automaton for \( n_1 \) and \( n_2 \) and a single track automaton for \( n_3 \)
- We have

\[
\alpha_{\chi, \chi'}(M) = M_\alpha \\
\gamma_{\chi, \chi'}(M_\alpha) = M_\gamma,
\]
where
Relation Abstraction

- \( M_\alpha = \{ M_1', M_2', M_3 \} \) such that \( M_1 \) and \( M_2 \) are constructed by the projection of \( M_{1,2} \) to the first track and the second track respectively

- \( M_\gamma = \{ M'_{1,2}', M_3 \} \) such that \( M'_{1,2} \) is constructed by the intersection of \( M_{1,*} \) and \( M_{*,2} \), where
  - \( M_{1,*} \) is the two-track automaton extended from \( M_1 \) with arbitrary values in the second track
  - \( M_{*,2} \) is the two-track automaton extended from \( M_2 \) with arbitrary values in the first track
An Example of Relation Abstraction

$M_{1,2}$

- $(b,b)$
- $(a,a)$
- $(c,c)$

$M'_{1,2}$

- $(b,b)$
- $(b,a)$
- $(a,a)$
- $(a,b)$
- $(c,c)$

$M_{1,*}$

- $(b,*)$
- $(a,*)$
- $(c,*)$

$M_{*2}$

- $(*,b)$
- $(*,a)$
- $(*,c)$

$M_{1,2}$ and $M_{1,*}$ are transformed into $M_{1,*}$ and $M_{*2}$ respectively, using the relations $\alpha$ and $\gamma$. The transformation involves merging states and transitions to abstract the relations.
Consider the above example, choosing $\chi' =\{\{\$usr, $key\}, \{\$passwd\}\}$ is sufficient to identify the echo string is a prefix of “admin1234” and does not contain any substring that matches “<script”
Both alphabet and relation abstractions form abstraction lattices, which allow different levels of abstractions.

Combining these abstractions leads a product lattice, where each point is an abstraction class that corresponds to a particular alphabet abstraction and a relation abstraction.

- The top is a non relational analysis using unary alphabet
- The bottom is a complete relational analysis using full alphabet
Some abstraction from the abstraction lattice and the corresponding analyses

\[(\chi_T, \sigma_T)\] size analysis

\[(\chi_{\perp}, \sigma_T)\] relational size analysis

\[(\chi_T, \sigma_{\perp})\] string analysis

\[(\chi_{\perp}, \sigma_{\perp})\] relational string analysis
Select an abstraction class
- Ideally, the choice should be as abstract as possible while remaining precise enough to prove the property in question

Heuristics
- Let the property guide the choice
- Collect constants and relations from assertions and their dependency graphs
  - It forms the lower bound of the abstraction class
  - Select an initial abstraction class, e.g., characters and relations appearing in assertions
  - Refine the abstraction class toward the lower bound