Implementing String Analysis with Soot Framework

Jaejun Choi, Sewon Kim
Programming Language Laboratory, Computer Science Dept., KAIST

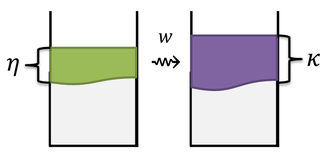
String Analysis

Motivation

Constraint generation approach
- Only use constraint that may lead to a false alarm.
- Dose not support modular analysis

Our Approach
- Abstract string literals by using reference PDA
- Abstract concatenation operation
- Abstract interpretation framework.

Configuration Change



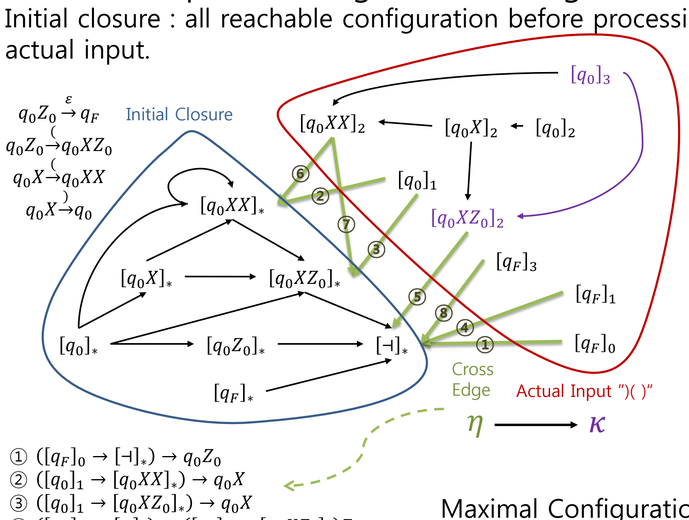
$\eta \rightsquigarrow \kappa(w)$

$w \in \Sigma^*$

$(\eta_1 \rightsquigarrow \kappa_1 \wedge \cdots \wedge \eta_n \rightsquigarrow \kappa_n)(w)$

Relation Graph of Configuration Change

Initial closure : all reachable configuration before processing actual input.



① $([q_F]_0 \rightarrow [\dashv]_*) \rightarrow q_0Z_0$
② $([q_0]_1 \rightarrow [q_0XX]_*) \rightarrow q_0X$
③ $([q_0]_1 \rightarrow [q_0XZ_0]_*) \rightarrow q_0X$
④ $([q_F]_1 \rightarrow [\dashv]_*) \rightarrow ([q_0]_1 \rightarrow [q_0XZ_0]_*)Z_0$
⑤ $([q_0XZ_0]_2 \rightarrow [\dashv]_*) \rightarrow ([q_0]_1 \rightarrow [q_0XZ_0]_*)Z_0$
⑥ $([q_0XX]_2 \rightarrow [q_0XX]_*) \rightarrow ([q_0]_1 \rightarrow [q_0XX]_*)X$
⑦ $([q_0XX]_2 \rightarrow [q_0XZ_0]_*) \rightarrow ([q_0]_1 \rightarrow [q_0XX]_*)X$
⑧ $([q_F]_3 \rightarrow [\dashv]_*) \rightarrow ([q_0XZ_0]_2 \rightarrow [\dashv]_*)$

Maximal Configuration Change

$q_0XZ_0 \rightsquigarrow q_0Z_0$
$q_0XX \rightsquigarrow q_0X$
$q_0XZ_0 \rightsquigarrow q_F$

Abstract Domain for String

We can define abstract domain $\mathfrak{D}_A = (\mathcal{C}_A, \sqsubseteq)$.

Set $\mathcal{C}_A$
- $\mathcal{C}_A = \mathcal{C'}_A \cup \{\bot\}$, $\mathcal{C'}_A$ consists of $C = \bigwedge_{i \in I} \eta_i \rightsquigarrow \kappa_i$, $\top$ is $\bigwedge\emptyset$.

Relation $\sqsubseteq$
- For $C_1, C_2 \in \mathcal{C}_A$, $C_1 \sqsubseteq C_2$ if and only if for each $c_2 \in C_2$, there exists $c_1 \in C_1\ s.t.\ c_1 \Rrightarrow c_2$.

Relation $\Rrightarrow$
- For predicates $\delta \rightsquigarrow \delta'$ and $\delta\,\eta \rightsquigarrow \delta'\eta$, $\delta \rightsquigarrow \delta'$ implies $\delta\,\eta \rightsquigarrow \delta'\eta$. We represent this syntactic implication as $\delta \rightsquigarrow \delta' \Rrightarrow \delta\,\eta \rightsquigarrow \delta'\eta$.

Least Upper bound
- And we can define least upper bound of conjunctions.

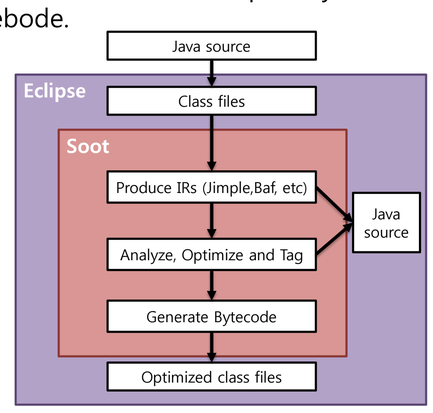
$$\sqcup_{i \in I} C_i = \bigwedge \{c \in \cup_{i \in I} \mid \forall i \in I.\ \exists c_i \in C_i.\ c_i \Rrightarrow c\}$$

Abstract concatenation operation

$$\eta_1 \rightsquigarrow \kappa_1 \odot \eta_2 \rightsquigarrow \kappa_2 \begin{cases} \eta_1 \rightsquigarrow \kappa_2\delta & \text{if } \kappa_1 = \eta_2\delta, \\ \eta_1\delta \rightsquigarrow \kappa_2 & \text{if } \kappa_1\delta = \eta_2 \text{ and } \exists\eta_1,\ q_0Z_0\delta \xrightarrow{*} \eta_1\delta\eta, \\ true & \text{otherwise.} \end{cases}$$

Soot Framework

Soot is a Java Optimization Framework.
- It provides intermediate representations for analyzing and transforming Java bytecode.
- Soot cat be used as a stand alone tool to optimize or inspect class file
- As well as a framework to develop analysis or transformations on Java bytebode.



Soot Intermediate Representations

Soot provides four intermediate representations Baf, Jimple, Shimple, and Grimp
- **Baf** is a compact representation of bytecode – stack based.
- **Jimple** is a simple, typed 3-address representation – stackless.

Baf

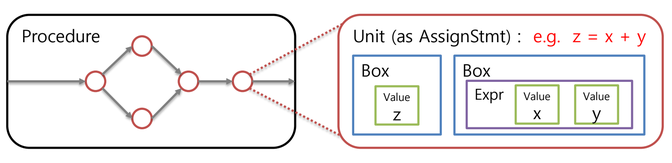
```
word x, y, z
load.i x
load.i y
add.i
Store.i z
```

Jimple

```
int $s0, $s1, x, y, z
$s0 = x
$s1 = y
$s0 = $s0 + $s1
z = #s0
```

Control Flow Graph - Intraprocedure

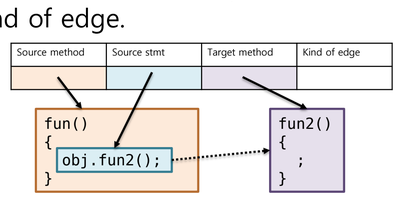
Soot provides control flow graph which each vertex is program point. In Soot, we call vertex of CFG Unit. Unit is a :
- **Instruction(Inst)** in Baf
- **Statement(Stmt)** is Jimple and Jimple-like code.



Call Graph - Interprocedure

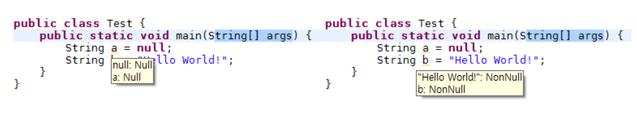
Call graph is a collection of Edges representing all known method invocation. This includes :
- Explicit method invocations.
- Implicit invocations of static initializers
- Implicit call of …

Each Edge contains source method, source statement, target method and kind of edge.



Annotation

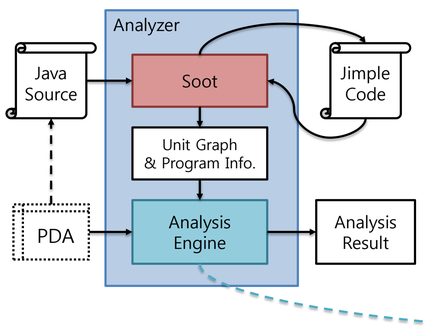
Soot provides a framework to support the embedding of custom, user-defined attributes in class file.



Implementation

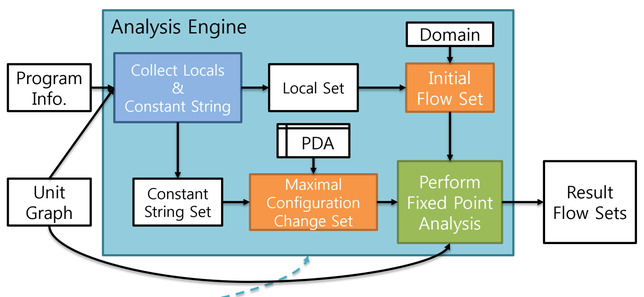
Analyzer Overview

String analyzer
- Takes two input: java source code which generates string and corresponding reference PDA.
- Transforms Java code to Jimple code and generates Unit graph by using Soot API.
- Analysis Engine analyze given Unit graph.

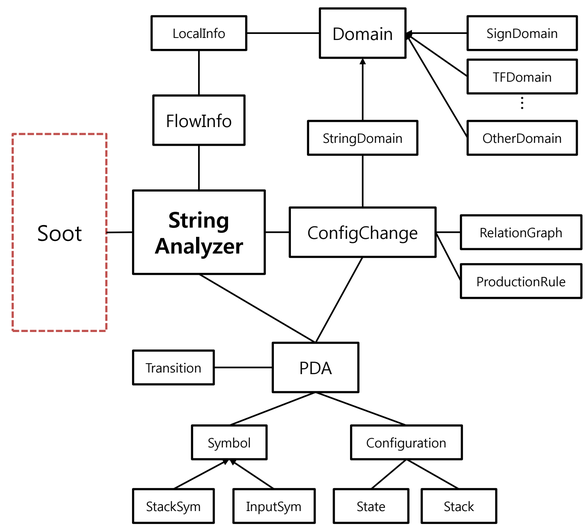


Analysis Engine Detail

Analysis Engine performs fixed point analysis.
- Collects all local variable and makes Initial Flow Set.
- Extracts all constant String from Unit graph and calculate Maximal Configuration Change Set based on reference PDA.
- Performs Fixed Point Analysis with given Flow Set and Maximal Configuration Change Set.



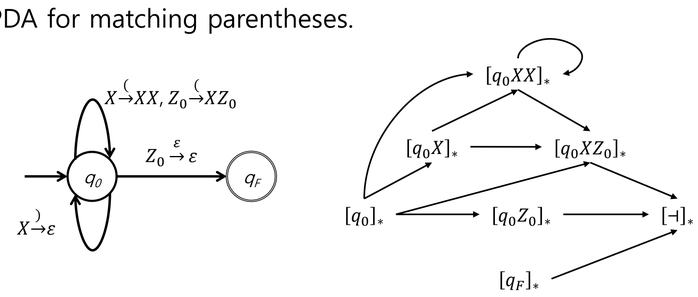
String Analyzer



Applying Example

Reference PDA and Initial Closure

A PDA for matching parentheses.



Java Code

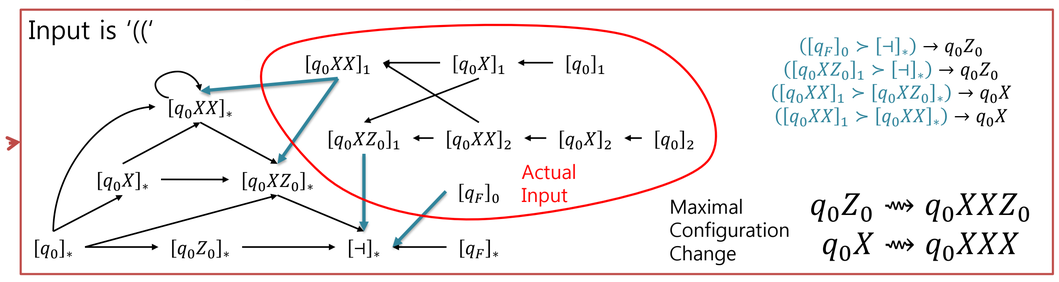
```
public static void main(String[] args)
{
    int i = 0;
    int n = 10;
    String x = "((";
    while (i < n)
    {
        x = x.concat(")(");
    }
    x = x.concat(")))");
}
```

Jimple Code

```
public static void main(java.lang.String[])
{
  java.lang.String[] args;
  int i, n;
  java.lang.String x, temp$0, temp$1;

  args := @parameter0: java.lang.String[];
  i = 0;
  n = 10;
  x = "((";

label0:
  nop;
  if i < n goto label1;
  goto label2;

label1:
  nop;
  temp$0 = virtualinvoke x.<java.lang.String:
java.lang.String concat(java.lang.String)>(")(");
  x = temp$0;
  goto label0;

label2:
  nop;
  temp$1 = virtualinvoke x.<java.lang.String:
java.lang.String concat(java.lang.String)>(")))");
  x = temp$1;
  return;
}
```

Maximal Configuration Change

Input is '(('



Analysis Result

Flow set at "return" stmt.

Type	Name	Value
java.lang.string[]	args	$\bot$
java.lang.string	x	$q_0X \rightsquigarrow q_0$, $q_0XZ_0 \rightsquigarrow q_F$
int	n	> 0
int	i	> 0
java.lang.string	Temp$0	$q_0Z_0 \rightsquigarrow q_0XXZ_0$, $q_0X \rightsquigarrow q_0XXX$
java.lang.string	Temp$1	$q_0X \rightsquigarrow q_0$, $q_0XZ_0 \rightsquigarrow q_F$