Interprocedural Analysis: Sharir-Pnueli’s Call-strings Approach

Deepak D’Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

14 September 2017
Outline

1. Motivation
2. Call-strings method
3. Correctness
4. Approximate call-string method
5. Bounded call-string method
How would we extend an abstract interpretation to handle programs with procedures?

```plaintext
main()
  x := 0;
  f();
  g();
  print x;
}

f()
  x := x+1;
  return;
}

g()
  f();
  return;
}
```

Question: what is the collecting state before the `print x` statement in `main`?

Answer: `x ↦ 2`
Handling programs with procedure calls

How would we extend an abstract interpretation to handle programs with procedures?

```c
main(){
    x := 0;
    f();
    g();
    print x;
}

f(){
    x := x+1;
    return;
}

g(){
    f();
    return;
}
```

Question: what is the collecting state before the `print x` statement in `main`?
How would we extend an abstract interpretation to handle programs with procedures?

main()
{x := 0; f(); g(); print x;}

f()
{x := x+1; return;}

g()
{f(); return;}

Question: what is the collecting state before the print x statement in main? Answer: x \rightarrow 2.
Handling programs with procedure calls

- Add extra edges
  - **call edges**: from call site (call p) to start of procedure (p)
  - **ret edges**: from return statement (in p) to point after call sites (“ret sites”) (call p).
Handling programs with procedure calls

- Assume variables are uniquely named across the program.
- Transfer functions for call/return edges?

```
A
x := 0
B
print x
call f
G
ret ret
x:=x+1 call f
call g
H
I
main f g
D
C
E
F J
K
L
```
Handling programs with procedure calls

- Assume variables are uniquely named across the program.
- Transfer functions for call/return edges? Identity if we assume no parameters/return values; else treat like an assignment statement.
Handling programs with procedure calls

- Assume variables are uniquely named across the program.
- Transfer functions for call/return edges? Identity if we assume no parameters/return values; else treat like an assignment statement.
- Now compute JOP in this extended control-flow graph.
Problem with JOP in this graph

Ex. 1. Actual collecting state at C?

What we want is Join over "Interprocedurally-Valid" Paths (JVP).
Problem with JOP in this graph

Ex. 1. Actual collecting state at C? \( \{x \mapsto 2\} \).
Problem with JOP in this graph

Ex. 1. Actual collecting state at C? \( \{x \mapsto 2\} \).
Ex. 2. JOP at C using naive extension of collecting analysis?

What we want is Join over "Interprocedurally-Valid" Paths (JVP).
Problem with JOP in this graph

Ex. 1. Actual collecting state at C? \(\{x \mapsto 2\}\).
Ex. 2. JOP at C using naive extension of collecting analysis? \(\{x \mapsto 1, \ x \mapsto 2, \ x \mapsto 3, \ldots\}\).
Problem with JOP in this graph

Ex. 1. Actual collecting state at C? \( \{x \mapsto 2\} \).
Ex. 2. JOP at C using naive extension of collecting analysis?
\( \{x \mapsto 1, \ x \mapsto 2, \ x \mapsto 3, \ldots\} \).

- JOP is sound but very imprecise.
- Reason: Some paths don’t correspond to executions of the program: Eg. ABDFGILC.
Problem with JOP in this graph

Ex. 1. Actual collecting state at C? \( \{x \mapsto 2\} \).
Ex. 2. JOP at C using naive extension of collecting analysis?
\( \{x \mapsto 1, \ x \mapsto 2, \ x \mapsto 3, \ldots\} \).

- JOP is sound but very imprecise.
- Reason: Some paths don’t correspond to executions of the program: Eg. ABDFGILC.
   What we want is Join over “Interprocedurally-Valid” Paths (JVP).
Informally a path $\rho$ in the extended CFG $G'$ is inter-procedurally valid if every return edge in $\rho$ “corresponds” to the most recent “pending” call edge.

For example, in the example program the ret edge $E$ corresponds to the call edge $D$.

The call-string of a valid path $\rho$ is a subsequence of call edges which have not been “returned” as yet in $\rho$.

For example, $cs(ABDFGEKJHF)$ is “$KH$“.
Interprocedurally valid paths and their call-strings

- A path \( \rho = ABDFGEKJHF \) in \( IVP_{G'} \) for example program:

  
  ![Diagram](image)

  - Associated call-string \( cs(\rho) = KH \).
  - For \( \rho = ABDFGEK \) \( cs(\rho) = K \).
  - For \( \rho = ABDFGE \) \( cs(\rho) = \epsilon \).
Sharir and Pnueli’s approaches to interprocedural analysis

Interprocedurally valid paths and their call-strings

More formally: Let $\rho$ be a path in $G'$. We define when $\rho$ is interprocedurally valid (and we say $\rho \in IVP(G')$) and what is its call-string $cs(\rho)$, by induction on the length of $\rho$.

- If $\rho = \epsilon$ then $\rho \in IVP(G')$. In this case $cs(\rho) = \epsilon$.
- If $\rho = \rho' \cdot N$ then $\rho \in IVP(G')$ iff $\rho' \in IVP(G')$ with $cs(\rho') = \gamma$ say, and one of the following holds:
  1. $N$ is neither a call nor a ret edge.
     In this case $cs(\rho) = \gamma$.
  2. $N$ is a call edge.
     In this case $cs(\rho) = \gamma \cdot N$.
  3. $N$ is ret edge, and $\gamma$ is of the form $\gamma' \cdot C$, and $N$ corresponds to the call edge $C$.
     In this case $cs(\rho) = \gamma'$.

- We denote the set of (potential) call-strings in $G'$ by $\Gamma$. Thus $\Gamma = C^*$, where $C$ is the set of call edges in $G'$. 
Join over interprocedurally-valid paths (JVP)

1. Let $P$ be a given program, with extended CFG $G'$.
2. Let $path_{I,N}(G')$ be the set of paths from the initial point $I$ to point $N$ in $G'$.
3. Let $\mathcal{A} = ((D, \leq), f_{MN}, d_0)$ be a given abstract interpretation.
4. Then we define the join over all interprocedurally valid paths (JVP) at point $N$ in $G'$ to be:

$$\bigsqcup_{\rho \in \text{path}_{I,N}(G') \cap IVP(G')} f_{\rho}(d_0).$$
One approach to obtain JVP

- Find JOP over same graph, but modify the abs int.
- Modify transfer functions for call/ret edges to detect and invalidate invalid edges.
- Augment underlying data values with some information for this.
- Natural thing to try: “call-strings”.

```
x := 0
print x
call f
g
ret
x:=x+1 call f
call g
main f g
c
D
E
F J
K
L
```
Overall plan

- Define an abs int $\mathcal{A}'$ which extends given abs int $\mathcal{A}$ with call-string data.
- Show that JOP of $\mathcal{A}'$ on $G'$ coincides with JVP of $\mathcal{A}$ on $G'$.
- Use Kildall (or any other technique) to compute LFP of $\mathcal{A}'$ on $G'$. This value over-approximates JVP of $\mathcal{A}$ on $G'$.
Call-string abs int $\mathcal{A}'$: Lattice $(D', \leq')$

- Elements of $D'$ are maps $\xi : \Gamma \to D$
  
  $\begin{array}{|c|c|c|c|}
  \hline
  & \epsilon & c_1 & c_1c_2c_2 \\
  \hline
  d_0 & d_1 & d_2 & d_3 \\
  \hline
  \end{array}$

- Ordering on $D': \leq'$ is the pointwise extension of $\leq$ in $D$.
  That is $\xi_1 \leq' \xi_2$ iff for each $\gamma \in \Gamma$, $\xi_1(\gamma) \leq \xi_2(\gamma)$.
Call-string abs int $A'$: Lattice $(D', \leq')$

- Elements of $D'$ are maps $\xi : \Gamma \rightarrow D$

$$
\begin{array}{cccccc}
\text{e} & c_1 & c_1c_2 & c_1c_2c_2 & \ldots \ldots \\
\text{d}_0 & \text{d}_1 & \text{d}_2 & \text{d}_3 & \ldots \ldots \\
\end{array}
$$

- Ordering on $D'$: $\leq'$ is the pointwise extension of $\leq$ in $D$.
- That is $\xi_1 \leq' \xi_2$ iff for each $\gamma \in \Gamma$, $\xi_1(\gamma) \leq \xi_2(\gamma)$.

$$
\begin{array}{cccccc}
\text{e} & c_1 & c_1c_2 & c_1c_2c_2 & \ldots \ldots \\
\text{d}_0 \sqcup \text{e}_0 \text{d}_1 \sqcup \text{e}_1 \text{d}_2 \sqcup \text{e}_2 \text{d}_3 \sqcup \text{e}_3 & \ldots \ldots \\
\end{array}
$$

- Check that $(D', \leq')$ is also a complete lattice.
Meaning of abstract values in $\mathcal{A}'$

- A call-string table $\xi$ at program point $N$ represents the fact that, for each call-string $\gamma$, there are some (initial) paths with call-string $\gamma$ reaching $N$, and the join of the abstract states (obtained by propagating $d_0$) along these paths is (dominated by) $\xi(\gamma)$.
- The transfer functions of $\mathcal{A}'$ should keep this meaning in mind.
Initial value $\xi_0$ is given by

$$\xi_0(\gamma) = \begin{cases} 
d_0 & \text{if } \gamma = \epsilon \\
\bot & \text{otherwise.}
\end{cases}$$

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$c_1$</th>
<th>$c_1c_2$</th>
<th>$c_1c_2c_2$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_0$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

Call-string abs int $A'$: transfer functions

- Transfer functions for non-call/ret edge $N$:
  \[ f'_{MN}(\xi) = f_{MN} \circ \xi. \]

- Transfer functions for call edge $N$:
  \[ f'_{MN}(\xi) = \lambda \gamma \cdot \left\{ \begin{array}{ll}
  \xi(\gamma') & \text{if } \gamma = \gamma' \cdot N \\
  \bot & \text{otherwise}
  \end{array} \right. \]

- Transfer functions for ret edge $N$ whose corresponding call edge is $C$:
  \[ f'_{MN}(\xi) = \lambda \gamma \cdot \xi(\gamma \cdot C) \]

- Transfer functions $f'_{MN}$ is monotonic (distributive) if each $f_{MN}$ is monotonic (distributive).
Transfer functions $f'_{MN}$ for example program

- **Non-call/ret edge $B$:**
  \[ \xi_B = f_{AB} \circ \xi_A. \]

- **Call edge $D$:**
  \[ \xi_D(\gamma) = \begin{cases} 
  \xi_B(\gamma') & \text{if } \gamma = \gamma' \cdot D \\
  \bot & \text{otherwise}
  \end{cases} \]

- **Return edge $E$:**
  \[ \xi_E(\gamma) = \xi_G(\gamma \cdot D). \]
Exercise 1

Let $A$ be the standard collecting state analysis. For brevity, represent a set of concrete states as $\{0, 1\}$ (meaning the 2 concrete states $x \mapsto 0$ and $x \mapsto 1$). Assume an initial value $d_0 = \{0\}$.

Show the call-string tagged abstract states (in the lattice $A'$) along the paths
1. ABDFGEKJHFGIL (interprocedurally valid)
2. ABDFGIL (interprocedurally invalid).
Exercise 2

Use Kildall’s algo to compute the LFP of the $\mathcal{A}'$ analysis for the example program. Start with initial value $d_0 = \{0\}$. 
Exercise 2

Use Kildall’s algo to compute the LFP of the $A'$ analysis for the example program. Start with initial value $d_0 = \{0\}$.
Exercise 2

Use Kildall’s algo to compute the LFP of the $A'$ analysis for the example program. Start with initial value $d_0 = \{0\}$. 

Diagram:

- **main**
  - $x := 0$
  - call $f$
  - call $g$
  - print $x$

- **$f$**
  - $x := x + 1$
  - ret

- **$g$**
  - call $f$
  - ret

Initial values:

- $0$
- $D$
Exercise 2

Use Kildall’s algo to compute the LFP of the $A'$ analysis for the example program. Start with initial value $d_0 = \{0\}$. 

```
x := 0
print x
call f
g
```

```
x := x + 1
call f
call g
```
Correctness claim

Assumption on $\mathcal{A}$: Each transfer function satisfies $f_{MN}(\perp) = \perp$.

Claim

Let $N$ be a point in $G'$. Then

$$JVP_{\mathcal{A}}(N) = \bigcup_{\gamma \in \Gamma} JOP_{\mathcal{A}'}(N)(\gamma).$$

Proof: Use following lemmas to prove that LHS dominates RHS and vice-versa.
Correctness claim: Lemma 1

**Lemma 1**

Let $\rho$ be a path in $IVP_{G'}$. Then

$$f'_\rho(\xi_0) = \lambda \gamma. \begin{cases} f_\rho(d_0) & \text{if } \gamma = cs(\rho) \\ \bot & \text{otherwise.} \end{cases}$$

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$c_1$</th>
<th>$cs(\rho)$</th>
<th>$c_1c_2c_2$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$d$</td>
<td>$\bot$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

Proof: by induction on the length of $\rho$. 
Correctness claim: Lemma 2

Lemma 2
Let $\rho$ be a path not in $IVP_{G'}$. Then

$$f'_\rho(\xi_0) = \lambda \gamma . \bot .$$

<table>
<thead>
<tr>
<th>$\varepsilon$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_1 c_2 c_2$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

Proof:
- $\rho$ must have an invalid prefix.
- Consider smallest such prefix $\alpha \cdot N$. Then it must be that $\alpha$ is valid and $N$ is a return edge not corresponding to $cs(\alpha)$.
- Using previous lemma it follows that $f'_{\alpha \cdot N}(\xi_0) = \lambda \gamma . \bot$.
- But then all extensions of $\alpha$ along $\rho$ must also have transfer function $\lambda \gamma . \bot$. 
Computing JOP for abs int $A'$

- Problem is that $D'$ is infinite in general (even if $D$ were finite). So we cannot use Kildall’s algo to compute an over-approximation of JOP.
- We give two methods to bound the number of call-strings
  - Use “approximate” call-strings.
  - Give a bound on largest call-string needed.
Approximate (suffix) call-string method

Idea:

- Consider only call-strings of upto length \( \leq l \), that may additionally be prefixed by a “∗”.
- A “∗” prefix means that we have left out some initial calls.
- For \( l = 2 \), call strings can be of the form “\( c_1 c_2 \)” or “\( ∗c_1 c_2 \)” etc. So each table \( ξ \) is now a finite table.
- Transfer functions for non-call/ret edges remain same.
- Transfer functions for call edge \( C \): Shift \( γ \) entry to \( γ \cdot C \) if \( |γ \cdot C| \leq l \); else shift it to \( ∗ \cdot γ' \cdot C \) where \( γ \) is of the form \( A \cdot γ' \), for some call \( A \).
- Transfer functions for ret edge \( N \):
  - If \( γ = γ' \cdot C \) and \( N \) corresponds to call edge \( C \), then shift \( γ' \cdot C \) entry to \( γ' \) entry.
  - If \( γ = ∗ \) then copy its entry to \( ∗ \) entry at the return site.
Exercise: approximate call-strings

Assume approximate call-string length of 2. Use Kildall’s algo to compute the $\xi$ table values for the example program. Start with initial value $d_0 = 0$. 

```
read a, b

C

D

P

O

L

M

N

J

I

Q

H

G

F

ret

a := a + 1

b := b + 1

call p

call p

print t

a == 0

a := a - 1

G

H

call p

B

A

1

2

3

4

5

6

7

8

9

10

11
```
Exercise: approximate call-strings

Assume approximate call-string length of 2. Use Kildall’s algo to compute the $\xi$ table values for the example program. Start with initial value $d_0 = 0$. 

Diagram:
- Read $a, b$ (Node A)
- Compute $t := a \times b$ (Node B)
- Call $p$ (Node C)
- Compute $t := a \times b$ (Node D)
- Print $t$ (Node E)
- Compute $t := a \times b$ (Node F)
- Check $a = 0$ (Node G)
- Compute $a := a - 1$ (Node H)
- Call $p$ (Node I)
- Compute $t := a \times b$ (Node J)
- Return (Node K)
Exercise: approximate call-strings

Assume approximate call-string length of 2. Use Kildall’s algo to compute the $\xi$ table values for the example program. Start with initial value $d_0 = 0$. 

```
a := a-1
7
F
G
t := a*b
1
A
read a,b
B

t := a*b
2
C
\xi
0
read a,b
B

call p
3
P
\xi
0
\xi
1
call p
P

t := a*b
4
D

print t
5
N
M
I
J
K

ret
11

a == 0
5
B
C
O
L
M
N
2
3
4
6
9
8
ret
10
I
J
K
P
H
Q
```
Exercise: approximate call-strings

Assume approximate call-string length of 2. Use Kildall’s algo to compute the $\xi$ table values for the example program. Start with initial value $d_0 = 0$. 

```
a := a−1
read a,b
t := a*b
call p
print t
call p
a == 0
a := a−1
call p
t := a*b
t := a*b
print t
ret
```

```
A
B
C
D
E
F
G
H
I
J
K
```

```
\text{null}
\epsilon_0
\epsilon_0
\epsilon_1
\text{null}
c_1
\epsilon_1
\epsilon_0
\epsilon_1
\epsilon_1
\epsilon_0
\epsilon_1
\epsilon_0
\epsilon_0
\epsilon_0
\epsilon_1
\epsilon_1
\epsilon_1
```

```
a == 0
a := a−1
call p
```
Exercise: approximate call-strings

Assume approximate call-string length of 2. Use Kildall’s algo to compute the $\xi$ table values for the example program. Start with initial value $d_0 = 0$. 

```
a := a - 1
read a, b
t := a * b
print t
call p
a == 0
5
B
C
O
L
M
N
2
3
4
6
9
8
ret
```
Possible to bound length of call-strings $\Gamma$ we need to consider.

For a number $l$, we denote the set of call-strings (for the given program $P$) of length at most $l$, by $\Gamma_l$.

Define a new analysis $A''$ ($M$-bounded call-string analysis) in which call-string tables have entries only for $\Gamma_M$ for a certain constant $M$, and transfer functions ignore entries for call-strings of length more than $M$.

We will show that $\text{JOP}(G', A'') = \text{JOP}(G', A')$.

\[ \text{LFP}(G', A') \]
\[ \text{LFP}(G', A'') \]

\[ \text{JOP}(G', A'') \quad \text{JOP}(G', A') \quad \text{JVP}(G', A) \]
Consider any fixpoint $V'$ (a vector of tables) of $\mathcal{A}'$. 

Truncate each entry of $V'$ to (call-strings of) length $M$, to get $V''$. 

Clearly $V'$ dominates $V''$. 

Further, observe that $V''$ is a post-fixpoint of the transfer functions for $\mathcal{A}''$. 

By Knaster-Tarski characterisation of LFP, we know that $V''$ dominates $\text{LFP}(\mathcal{A}'')$. 

\[
\text{LFP}(G', \mathcal{A}') \\
\text{LFP}(G', \mathcal{A}'') \\
\text{JOP}(G', \mathcal{A}'') \\
\text{JOP}(G', \mathcal{A'}) \\
\text{JVP}(G', \mathcal{A})
\]
Sufficiency (or safety) of bound

Let $k$ be the number of call sites in $P$.

**Claim**

For any path $p$ in $IVP(r_1, N)$ with a prefix $q$ such that $|cs(q)| > k|D|^2 = M$ there is a path $p'$ in $IVP(r_1, N)$ with $|cs(q')| \leq M$ for each prefix $q'$ of $p'$, and $f_p(d_0) = f_{p'}(d_0)$.

**Paths with bounded call-strings**

\[
\begin{align*}
M
\end{align*}
\]
Proving claim

Claim
For any path $p$ in $IVP(r_1, N)$ such that for some prefix $q$ of $p$, $|cs(q)| > M = k|D|^2$, there is a path $p'$ in $IVP_{\Gamma M}(r_1, N)$ with $f_{p'}(d_0) = f_p(d_0)$.

Sufficient to prove:

Subclaim
For any path $p$ in $IVP(r_1, N)$ with a prefix $q$ such that $|cs(q)| > M$, we can produce a smaller path $p'$ in $IVP(r_1, N)$ with $f_{p'}(d_0) = f_p(d_0)$.

...since if $|p| \leq M$ then $p \in IVP_{\Gamma M}$. 
A path $\rho$ in $IVP(r_1, n)$ can be decomposed as

$$\rho_1 \parallel (c_1, r_{p_2}) \parallel \rho_2 \parallel (c_2, r_{p_3}) \parallel \sigma_3 \parallel \cdots \parallel (c_{j-1}, r_{p_j}) \parallel \rho_j.$$  

where each $\rho_i \ (i < j)$ is a valid and complete path from $r_{p_i}$ to $c_i$, and $\rho_j$ is a valid and complete path from $r_{p_j}$ to $n$. Thus $c_1, \ldots, c_{j-1}$ are the unfinished calls at the end of $\rho$. 
Proving subclaim

- Let $p_0$ be the first prefix of $p$ where $|cs(p_0)| > M$.
- Let decomposition of $p_0$ be
  $$\rho_1 \parallel (c_1, r_{p_2}) \parallel \rho_2 \parallel (c_2, r_{p_3}) \parallel \sigma_3 \parallel \cdots \parallel (c_{j-1}, r_{p_j}) \parallel \rho_j.$$  
- Tag each unfinished-call $c$ in $p_0$ by $(c, f_q.c(d_0), f_q.cq'e(d_0))$ where $e$ is corresponding return of $c$ in $p$.
- If no return for $c$ in $p$ tag with $(c, f_q.c(d_0), \perp)$.
- Number of distinct such tags is $k \cdot |D|^2$.
- So there are two calls $qc$ and $qcq'c$ with same tag values.
Proving subclaim – tag values are $\perp$
Proving subclaim – tag values are not $\perp$
Example

Motivation
Call-strings method
Correctness
Approximate call-string method
Bounded call-string method

Example

A
read a,b

B
6
return

C
call p2

t:=a*b

D
E
F
G
K
I
call p1 call p1

T

L

M

N
return

e2
m2
n2
k

e1
n1
m1
r1

P
Q
R
S
E
6

c1

a := 0

k

K
Transfer functions $f'_{MN}$ for Example 2

- **Non-call/ret edge $C$:**
  \[ \xi_C = f_{BC} \circ \xi_B. \]

- **Call edge $O$:**
  \[ \xi_O(\gamma) = \begin{cases} \xi_C(\gamma') & \text{if } \gamma = \gamma' \\ \bot & \text{otherwise} \end{cases} \]

- **Return edge $N$:**
  \[ \xi_N(\gamma) = \xi_J(\gamma \cdot O). \]