From Program slicing to Abstract Interpretation

Manuel Oriol
January 15th, 2007
What is program slicing?

A technique for analyzing programs regarding to a specific criterion.

More specifically, the analysis is meant to find the statements that participate to a result.
Intuition

What are the statements leading to the value of b at the end?

\[
\begin{align*}
a &: = 1 \\
b &: = 5 \\
\text{if } (b > 3) \text{ then} \\
& \quad \text{Result} := b \\
\text{else} \\
& \quad a := 2 \\
\text{end} \\
b &: = a
\end{align*}
\]
Key Idea: static slicing criteria

Slicing criterion:

$$(S, \{\text{variables}\})$$

- A statement, a point in the program
- The set of variables that matter
The static slice

The set of statements that lead to the state of the variables at the chosen statement.

Example:

\[\begin{align*}
i &:= 3 \\
\text{fact} &:= 1 \\
\text{For } i := 1 \text{ until } i > 10 \text{ loop} \\
& \quad \text{fact} := \text{fact} \times i \\
& \quad \text{last}_i := i \quad \text{(middle)} \\
& \quad \text{io.put(“last I:”+last_i)} \\
& \quad i := i + 1 \\
\text{end}
\end{align*}\]

(end,i)? (end,fact)? (middle,i)?
Key Idea: dynamic slicing criteria

Slicing criterion:

\[(x, S^q, \{\text{variables}\})\]

- **Input of the program**
- **Statement \(S\) in \(q^{th}\) position**
- **The set of variables that matter**
The dynamic slice

The set of statements that lead to the state of the variables at the chosen statement given input x.

Example:

\[
\begin{align*}
n &:= \text{read\_int}() \\
i &:= 3 \\
fact &:= 1 \\
\text{From } i &:= 1 \text{ until } i > n \text{ loop} \\
& \quad \text{fact} := \text{fact} \times i \\
& \quad \text{last\_i} := i \quad \text{(middle)} \\
& \quad \text{io.put("last I:" + last\_i)} \\
& \quad i := i + 1 \\
& \end{align*}
\]

(10,end\(^1\),i)? (0,end\(^1\),fact)? (5,middle\(^2\),i)?
Application: Debugging

- **Simpler:** Easier to understand what’s wrong
- **Remove statements:** Detect dead code
- **By comparing to an intended behavior:** detects bugs in the behavior
Other Applications

- Software maintenance
- Testing
- Optimizations
What is abstract interpretation?

A technique for analyzing the programs by modeling their values and operations.

In fact it is an execution that one can make to prove facts.
Intuition

Set of values:
\[ V ::= \text{integers} \]

Expressions:
\[ e ::= e \ast e \mid i \in V \]

Language:
\[ \text{eval} : e \rightarrow \text{integers} \]
\[ \text{eval}(i) = i \]
\[ \text{eval}(e_1 \ast e_2) = \text{eval}(e_1) \times \text{eval}(e_2) \]

How can we decide on the sign of the evaluated expressions?
Key Idea: the Abstraction!

How is this called?

Homomorphism

State $\rightarrow$ State

Abstract State $\rightarrow$ Abstract State

$\alpha$ $\gamma$

next

next
Abstraction

Set of values:
\[ V ::= \text{integers} \]

Expressions:
\[ e ::= e \times e \mid i \in V \]

Language:
\[
\begin{align*}
\text{eval}: & \quad e \rightarrow \text{integers} \\
\text{eval}(i) & = i \\
\text{eval}(e_1 \times e_2) & = \text{eval}(e_1) \times \text{eval}(e_2)
\end{align*}
\]

Addition unary minus?

Set of abstract values:
\[ AV ::= \{+, -, 0\} \]

Expressions:
\[ e ::= e \times e \mid ai \in AV \]

Language:
\[
\begin{align*}
\text{aeval}: & \quad e \rightarrow AV \\
\text{aeval}(i>0) & = + \\
\text{aeval}(i<0) & = - \\
\text{aeval}(i=0) & = 0 \\
\text{aeval}(e_1 \times e_2) & = \text{aeval}(e_1) \times \text{aeval}(e_2)
\end{align*}
\]

where
\[
\begin{align*}
+*- & = - \\
+*+= & = + \\
-*= & = - \\
0*av & = 0, av*0 & = 0
\end{align*}
\]
If only the world would be so great...

How is this called?

Semi-Homomorphism

State \( \xrightarrow{\text{next}} \) State

\( \alpha \)

Abstract State \( \xrightarrow{\text{next}} \) Abstract State

\( \subseteq \)

\( \alpha \)
Abstraction

Set of values:
V::= integers

Expressions:
e::= e * e | -e | e + e | i ∈ V

Language:
eval: e -> integers
eval(i) = i
eval(-e)=-eval(e)
eval(e1*e2) = eval(e1) * eval(e2)
eval(e1+e2) = eval(e1) + eval(e2)

Set of abstract values:
AV::= {+, -, 0, T}

Expressions:
e::= e * e | -e | e + e | av ∈ AV

Language:
aeval: e -> AV
aeval(integer) = ... as before
aeval(e1*e2) = ... as before
aeval(-e) = ... easy ;)
aeval(e1+e2) =
    aeval(e1)+ aeval(e2)
where
    ++- = T
    +++ =+
    -+-=-
0+av=av, av+0=av
Abstraction complete?

Set of values:
\[ V ::= \text{integers} \]

Expressions:
\[ e ::= e \ast e \mid -e \mid e + e \mid e/e \mid i \in V \]

Language:
\[
\begin{align*}
\text{eval}: & e \rightarrow \text{integers} \\
\text{eval}(i) &= i \\
\text{eval}(-e) &= -\text{eval}(e) \\
\text{eval}(e_1 \ast e_2) &= \text{eval}(e_1) \ast \text{eval}(e_2) \\
\text{eval}(e_1 + e_2) &= \text{eval}(e_1) + \text{eval}(e_2) \\
\text{eval}(e_1/e_2) &= \text{eval}(e_1) / \text{eval}(e_2)
\end{align*}
\]

Set of abstract values:
\[ AV ::= \{+, -, 0, T, \bot\} \]

Expressions:
\[ e ::= e \ast e \mid -e \mid e + e \mid e/e \mid av \in AV \]

Language:
\[
\begin{align*}
\text{aeval}: & e \rightarrow AV \\
\text{aeval}(\text{integer}) &= \ldots \text{as before} \\
\text{aeval}(e_1 \ast e_2) &= \ldots \text{as before} \\
\text{aeval}(-e) &= \ldots \text{easy ;}) \\
\text{aeval}(e_1/e_2) &= \\
\quad \text{aeval}(e_1) / \text{aeval}(e_2) \\
\quad \text{where} \quad av/0 = \bot \\
\quad av+\bot = \bot
\end{align*}
\]

...
Significance of the results?

- It is sound!  
  (the results are correct)

- It is far from complete!!!!!  
  (the results loose too much information)
Condition for Soundness

It should be a Galois insertion:

\[ \gamma \text{ and } \alpha \text{ monotonic } (x \geq y \Rightarrow f(x) \geq f(y)) \]

for all \( S \): \( S \subseteq \gamma(\alpha(S)) \)

\( \alpha(\gamma(\text{av})) = \text{av} \)
Monotonic Functions

In the example:

for $\alpha$: $(S, \subseteq) \rightarrow (AV, \leq)$

for $\gamma$: $(av, \leq) \rightarrow (S, \subseteq)$
Exercise

Prove that the expression is divisible by 3.

Set of abstract values:
AV ::= \{true, false, T, \bot\}

Expressions:
e ::= e * e | e - e | e + e | e/e | a_i ∈ AV

Language:
aeval: e -> AV
aeval(3) = yes
aeval(e1 * e2) = yes iff
  aeval(e1) = yes or
  aeval(e2) = yes
aeval(-e) = ... easy ;)
aeval(e1 + e2) = aeval(e1) and
  aeval(e2)
aeval(e1 / e2) =
  true if aeval(e1) and not
  aeval (e2)
Presenting it...

Usually presented through the definition of transitions...
Prove that this program does not try to access a value outside the array's definition, a of size 10 (from 1 to 10)

\[
\begin{align*}
j &:= 0 \\
\text{from } i &:= 1 \text{ until } i > 50 \text{ loop} \\
& \quad j := j + (45 - a.\text{item}(i) + a.\text{item}(2i)) \\
& \quad i := i + 1 \\
\end{align*}
\]
Using abstract interpretation...

- What abstraction would you use to compute the call graph of a program?

- What abstraction would you use to optimize the tests within a program?
How would you verify that loops terminate?  
- Is it sound? Is it complete?

How would you verify that a password read on the keyboard is not sent through a socket?  
- Is it sound? Is it complete?
Applications to Trusted Components

- Dataflow Analysis?
- Program Slicing?
- Abstract Interpretation?