From Program slicing to Abstract Interpretation

Manuel Oriol
January 16th, 2005
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.
What are the statements leading to the value of \( b \) at the end?

\[
a := 1 \\
b := 5 \\
\text{if } (b > 3) \text{ then} \\
\quad \text{Result} := b \\
\text{else} \\
\quad a := 2 \\
\text{end} \\
b := a
\]
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:

i:=3
fact:=1
From i:=1 until i>10 loop
    fact:=fact*i
    last_i:=i
    io.put("last I:"+last_i)  (middle)
    io.put("last I:"+last_i)
    i:=i+1
end

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

Slicing criterion:

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

Input of the program

The set of variables that matter

Statement \(S\) in \(q^{th}\) position
The dynamic slice

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

Example:

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

(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

\[ \text{State} \rightarrow \text{State} \]

\[ \alpha \]

\[ \gamma \]

\[ \text{Abstract State} \rightarrow \text{Abstract State} \]

\[ \alpha \]
Abstraction

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

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

Language:
\[ \text{eval: } e \rightarrow \text{integers} \]
\[ \text{eval}(i) = i \]
\[ \text{eval}(e1 \ast e2) = \text{eval}(e1) \ast \text{eval}(e2) \]

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

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

Language:
\[ \text{aeval: } e \rightarrow AV \]
\[ \text{aeval}(i > 0) = + \]
\[ \text{aeval}(i < 0) = - \]
\[ \text{aeval}(i = 0) = 0 \]
\[ \text{aeval}(e1 \ast e2) = \text{aeval}(e1) \ast \text{aeval}(e2) \]
\[ \text{where } +_{-} = - \]
\[ +_{+} = + \]
\[ -_{=} = + \]
\[ 0_{av} = 0, \ av_{0} = 0 \]

Adding unary minus?
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

\[\subseteq\]
Abstraction

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

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

Language:
\[
\begin{align*}
\text{eval} & : e \rightarrow \text{integers} \\
\text{eval}(i) &= i \\
\text{eval}(-e) &= -\text{eval}(e) \\
\text{eval}(e1*e2) &= \text{eval}(e1) * \text{eval}(e2) \\
\text{eval}(e1+e2) &= \text{eval}(e1) + \text{eval}(e2)
\end{align*}
\]

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

Expressions:
\[ e ::= e * e \mid -e \mid e + e \mid ai \in AV \]

Language:
\[
\begin{align*}
\text{aeval} & : e \rightarrow AV \\
\text{aeval}(\text{integer}) &= \ldots \text{as before} \\
\text{aeval}(e1*e2) &= \ldots \text{as before} \\
\text{aeval}(-e) &= \ldots \text{easy ;)} \\
\text{aeval}(e1+e2) &= \\
&= \text{aeval}(e1)+\text{aeval}(e2) \\
\text{where} & \quad +_+ = T \\
& \quad +++ = + \\
& \quad -+- = -
\end{align*}
\]

0+av=av, av+0=av
Abstraction complete?

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

Expressions:
\[ e ::= e \times e | -e | e + e | e/e | i \in V \]

Language:
\[ \text{eval: } e \to \text{integers} \]
\[ \text{eval}(i) = i \]
\[ \text{eval}(-e) = -\text{eval}(e) \]
\[ \text{eval}(e_1 \times e_2) = \text{eval}(e_1) \times \text{eval}(e_2) \]
\[ \text{eval}(e_1 + e_2) = \text{eval}(e_1) + \text{eval}(e_2) \]
\[ \text{eval}(e_1/e_2) = \text{eval}(e_1) / \text{eval}(e_2) \]

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

Expressions:
\[ e ::= e \times e | -e | e + e | e/e | a \in AV \]

Language:
\[ \text{aeval: } e \to AV \]
\[ \text{aeval}(\text{integer}) = \ldots \text{as before} \]
\[ \text{aeval}(e_1 \times e_2) = \ldots \text{as before} \]
\[ \text{aeval}(-e) = \ldots \text{easy ;} \]
\[ \text{aeval}(e_1/e_2) = \text{aeval}(e_1)/\text{aeval}(e_2) \]
\[ \text{where} \quad \text{av}/0 = \bot \]
\[ \text{av+}_\bot = \bot \]

...
Significance of the results?

- It is sound!
  (the results are correct)

- It is far from complete!!!!!
  (the results lose 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:
\[ AV := \{ \text{true}, \text{false}, T, \bot \} \]

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

Language:
\[ \text{aeval: } e \rightarrow AV \]
\[ \text{aeval}(3) = \text{yes} \]
\[ \text{aeval}(e_1 \ast e_2) = \text{yes} \text{ iff } \]
\[ \text{aeval}(e_1) = \text{yes} \text{ or } \text{aeval}(e_2) = \text{yes} \]
\[ \text{aeval}(-e) = \ldots \text{ easy ;)} \]
\[ \text{aeval}(e_1 + e_2) = \text{aeval}(e_1) \text{ and } \text{aeval}(e_2) \]
\[ \text{aeval}(e_1/e_2) = \]
\[ \text{true if } \text{aeval}(e_1) \text{ and not } \text{aeval}(e_2) \]
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)

\[
j:=0
\]

\[
\text{from } i:=1 \text{ until } i>50 \text{ loop}
\]

\[
j=j+(45-a.item(i)+a.item(2*i))
\]

\[
i:=i+1
\]

\[
\text{end}
\]
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?
Problems

- 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?