Static Program Analysis
Part 6 – path sensitivity

http://cs.au.dk/~amoeller/spa/

Anders Møller & Michael I. Schwartzbach
Computer Science, Aarhus University
Information in conditions

The interval analysis (with widening) concludes:

\[ x = [-\infty, \infty], \quad y = [0, \infty], \quad z = [-\infty, \infty] \]
Modeling conditions

Add artificial “assert” statements:

The statement \texttt{assert}(E) models that $E$ is \textit{true} in the current program state

• it causes a runtime error otherwise

• but we only insert it where the condition will always be true
x = input;  
y = 0;  
z = 0;  
while (x>0) {
    assert(x>0);  
z = z+x;  
    if (17>y) { assert(17>y); y = y+1; }  
    else { assert(!(17>y)); }  
    x = x-1;  
}
assert(!(x>0));

(Alternatively, we could add dataflow constraints on the CFG edges)
Constraints for assert

• A trivial but sound constraint:

\[ [v] = JOIN(v) \]

• A non-trivial constraint for \texttt{assert}(x>E):

\[ [v] = JOIN(v)[x \mapsto gt(JOIN(v)(x), eval(JOIN(v),E))] \]

where

\[ gt([l_1, h_1], [l_2, h_2]) = [l_1, h_1] \cap [l_2, \infty] \]

• Similar constraints are defined for the dual cases

• More tricky to define for other conditions...
Exploiting conditions

The interval analysis now concludes:
\[ x = [-\infty, 0], \quad y = [0, 17], \quad z = [0, \infty] \]
• With assert we have a simple form of path sensitivity (sometimes called control sensitivity)

• But it is insufficient to handle correlation of branches:

```c
if (17 > x) { ... }
... // statements that do not change x
if (17 > x) { ... }
... 
```
Open and closed files

• Built-in functions `open()` and `close()` on a file

• Requirements:
  – never `close` a closed file
  – never `open` an open file

• We want a static analysis to check this...
  (for simplicity, let us assume there is only one file)
A tricky example

```java
if (condition) {
    open();
    flag = 1;
} else {
    flag = 0;
}
...
if (flag) {
    close();
}
```
The naive analysis (1/2)

• The lattice models the status of the file:

\[ L = (\mathcal{P}\{\text{open, closed}\}, \subseteq) \]

• For every CFG node, \( v \), we have a constraint variable \( \llbracket v \rrbracket \) denoting the status after \( v \)

• \( \text{JOIN}(v) = \bigcup_{w \in \text{pred}(v)} \llbracket w \rrbracket \)
The naive analysis (2/2)

• Constraints for interesting statements:
  \[
  \begin{align*}
  \llbracket \text{entry} \rrbracket &= \{ \text{closed} \} \\
  \llbracket \text{open()} \rrbracket &= \{ \text{open} \} \\
  \llbracket \text{close()} \rrbracket &= \{ \text{closed} \}
  \end{align*}
  \]

• For all other CFG nodes:
  \[
  \llbracket v \rrbracket = \text{JOIN}(v)
  \]

• Before the \texttt{close()} statement the analysis concludes that the file is \{open,closed\} 😞

```cpp
if (condition) {
    open();
    flag = 1;
} else {
    flag = 0;
}
...
if (flag) {
    close();
}
```
The slightly less naive analysis

- We obviously need to keep track of the flag variable
- Our second attempt is the lattice:

\[ L = (\mathcal{P}\{\text{open, closed}\}) \times \mathcal{P}\{\text{flag=0, flag} \neq 0\}, \subseteq \times \subseteq) \]

- Additionally, we add `assert(...)` to model conditionals

- Even so, we still only know that the file is \{open, closed\} and that flag is \{flag=0, flag\neq 0\} 😞

```java
if (condition) {
    open();
    flag = 1;
} else {
    flag = 0;
}
...
if (flag) {
    close();
}
```
if (condition) {
    assert(condition);
    open();
    flag = 1;
} else {
    assert(!condition);
    flag = 0;
}
...
if (flag) {
    assert(flag);
    close();
} else {
    assert(!flag);
}
Relational analysis

• We need an analysis that keeps track of *relations* between variables

• One approach is to maintain *multiple* abstract states per program point, one for each *path context*

• For the file example we need the lattice:

\[
L = \text{Paths} \rightarrow \mathcal{P}(\{\text{open, closed}\})
\]

(isomorphic to \(L = \mathcal{P}(\text{Paths} \times \{\text{open, closed}\})\))

where Paths = \{flag=0, flag\neq 0\} is the set of path contexts
Relational constraints (1/2)

• For the file statements:

\[ \text{entry} = \lambda p.\{\text{closed}\} \]
\[ \text{open()} = \lambda p.\{\text{open}\} \]
\[ \text{closed()} = \lambda p.\{\text{closed}\} \]

• For flag assignments:

\[ \text{flag} = 0 = [\text{flag}=0 \rightarrow \bigcup_{p \in P} \text{JOIN}(v)(p), \text{flag} \neq 0 \rightarrow \emptyset] \]
\[ \text{flag} = n = [\text{flag} \neq 0 \rightarrow \bigcup_{p \in P} \text{JOIN}(v)(p), \text{flag}=0 \rightarrow \emptyset] \]
\[ \text{flag} = E = \lambda q. \bigcup_{p \in P} \text{JOIN}(v)(p) \text{ for any other } E \]

where \( n \) is a non-0 constant number
Relational constraints (2/2)

• For assert statements:

\[
\left[ \text{assert}(\text{flag}) \right] = \\
\left[ \text{flag} \neq 0 \rightarrow \text{JOIN}(v)(\text{flag} \neq 0), \text{flag} = 0 \rightarrow \emptyset \right] \\
\left[ \text{assert}(\neg \text{flag}) \right] = \\
\left[ \text{flag} = 0 \rightarrow \text{JOIN}(v)(\text{flag} = 0), \text{flag} \neq 0 \rightarrow \emptyset \right]
\]

• For all other CFG nodes:

\[
\left[ v \right] = \text{JOIN}(v) = \lambda p. \bigcup_{w \in \text{pred}(v)} \left[ w \right](p)
\]
Generated constraints

\[
\begin{align*}
[\text{entry}] &= \lambda p.\{\text{closed}\} \\
[\text{condition}] &= [\text{entry}] \\
[\text{assert(\text{condition})}] &= [\text{condition}] \\
[\text{open(\()\)] &= \lambda p.\{\text{open}\} \\
[\text{flag = 1}] &= [\text{flag\neq 0}\rightarrow U [\text{open(\())\)](p), \text{flag=0}\rightarrow\emptyset] \\
[\text{assert(!condition)}] &= [\text{condition}] \\
[\text{flag = 0}] &= [\text{flag=0}\rightarrow U [\text{assert(!condition)}](p), \text{flag\neq 0}\rightarrow\emptyset] \\
[\ldots] &= \lambda p.([\text{flag = 1}](p) U [\text{flag = 0}](p)) \\
[\text{flag}] &= [\ldots] \\
[\text{assert(flag)}] &= [\text{flag\neq 0}\rightarrow [\text{flag}](\text{flag\neq 0}), \text{flag=0}\rightarrow\emptyset] \\
[\text{close(\()\)] &= \lambda p.\{\text{closed}\} \\
[\text{assert(!flag)}] &= [\text{flag=0}\rightarrow [\text{flag}](\text{flag=0}), \text{flag\neq 0}\rightarrow\emptyset] \\
[\text{exit}] &= \lambda p.([\text{close(\())\)](p) U [\text{assert(!flag)}](p))
\end{align*}
\]
Minimal solution

<table>
<thead>
<tr>
<th>Entry</th>
<th>Flag = 0</th>
<th>Flag ≠ 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>[entry]</td>
<td>{closed}</td>
<td>{closed}</td>
</tr>
<tr>
<td>[condition]</td>
<td>{closed}</td>
<td>{closed}</td>
</tr>
<tr>
<td>[assert(condition)]</td>
<td>{closed}</td>
<td>{closed}</td>
</tr>
<tr>
<td>[open()]</td>
<td>{open}</td>
<td>{open}</td>
</tr>
<tr>
<td>[flag = 1]</td>
<td>∅</td>
<td>{open}</td>
</tr>
<tr>
<td>[assert(!condition)]</td>
<td>{closed}</td>
<td>{closed}</td>
</tr>
<tr>
<td>[flag = 0]</td>
<td>{closed}</td>
<td>∅</td>
</tr>
<tr>
<td>[...]</td>
<td>{closed}</td>
<td>{open}</td>
</tr>
<tr>
<td>[flag]</td>
<td>{closed}</td>
<td>{open}</td>
</tr>
<tr>
<td>[assert(flag)]</td>
<td>∅</td>
<td>{open}</td>
</tr>
<tr>
<td>[close()]</td>
<td>{closed}</td>
<td>{closed}</td>
</tr>
<tr>
<td>[assert(!flag)]</td>
<td>{closed}</td>
<td>∅</td>
</tr>
<tr>
<td>[exit]</td>
<td>{closed}</td>
<td>{closed}</td>
</tr>
</tbody>
</table>

We now know the file is open before close() 😊
Challenges

• The static analysis designer must choose Paths
  – often as boolean combinations of predicates from conditionals
  – iterative refinement (e.g. counter-example guided abstraction refinement) can be used for gradually finding relevant predicates

• Exponential blow-up:
  – for $k$ predicates, we have $2^k$ different contexts
  – redundancy often cuts this down

• Reasoning about assert:
  – how to update the lattice elements with sufficient precision?
  – possibly involves heavy-weight theorem proving
Improvements

• Run auxiliary analyses first, for example:
  – constant propagation
  – sign analysis

will help in handling flag assignments

• Dead code propagation, change

  \[
  \llbracket \text{open()} \rrbracket = \lambda p.\{\text{open}\}
  \]

into the still sound but more precise

  \[
  \llbracket \text{open()} \rrbracket = \lambda p.\text{if } JOIN(v)(p) = \emptyset \text{ then } \emptyset \text{ else } \{\text{open}\}
  \]