Automated Verification Using Unified Control Flows

Cristian Gherghina    Cristina David

Department of Computer Science, National University of Singapore
Goals

• Verify programs with exception handling constructs

• Unify control flow types:
  – Dynamic: exceptions, program errors
  – Static: induced by break, continue and return

• Introduce a specification logic that captures the states for both normal and exceptional executions

• Handle generalized versions of raise and try-catch constructs
data Account{int number; int balance; }

class NoCredit extends exc {}

void withdraw(Account a, int s) throws NoCredit
    requires a::Account<i,b>
    ensures (a::Account<i,q>∧q=b−s∧b≥s∧flow=norm)
        ∨(a::Account<i,b>*res::NoCredit<>∧b<s∧flow=NoCredit);
{
    if (a.balance>s)
        a.balance = a.balance−s;
    else raise new NoCredit();
}

int remove10 (Account a)
    requires a::Account<n1,v1>
    ensures ((a::Account<n1,v>∧v1>10∧v=v1−10∧res=1)
        ∨ (a::Account<n1,v1>∧v1≤10∧res=0))∧flow=norm;
{
    try{
        withdraw(a,10);
    }catch (NoCredit# v){return 0;};
    return 1;  }
Unified Control Flow Hierarchy

dynamic control flows due to exceptions

can be caught

static control flows
cannot be caught

normal execution
Specification Formulae

- Based on separation logic formulae $\Phi$
- Enriched with
  - Flow variables, used to capture the exact flow type at different program points
  - $\tau$ captures the current flow
  - $\beta$ captures constraints on flow variables

\[
\begin{align*}
\Phi & ::= \bigvee (\exists v^* \cdot \kappa \land \pi)^* & \pi & ::= \gamma \land \phi \land \tau \\
\gamma & ::= v_1 = v_2 \mid v = \text{null} \mid v_1 \neq v_2 \mid v \neq \text{null} \mid \gamma_1 \land \gamma_2 \\
\kappa & ::= \text{emp} \mid v::c(v^*) \mid \kappa_1 \cdot \kappa_2 & \beta & ::= f v = c \mid f v_1 = f v_2 \\
\Delta & ::= \Phi \mid \Delta_1 \lor \Delta_2 \mid \Delta \land \pi \mid \Delta_1 \star \Delta_2 \mid \exists v \cdot \Delta \mid \Delta \land \beta \\
\phi & ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid c < v \mid \phi_1 \land \phi_2 \\
& \quad \quad \quad \quad \quad \quad \mid \phi_1 \lor \phi_2 \mid \neg \phi \mid \exists v \cdot \phi \mid \forall v \cdot \phi \\
\tau & ::= \text{flow} = f \text{set} \quad f \text{set} ::= \text{Ex}(c) \mid c - \{c_1, \ldots, c_n\} \\
a & ::= k \mid v \mid k \times a \mid a_1 + a_2 \mid -a \mid \text{max}(a_1, a_2) \mid \text{min}(a_1, a_2)
\end{align*}
\]
Specification Language

- Introduced two generalized constructs
  - Try-catch: try $e_1$ catch (c[@$v_1$] $v_2$) $e_2$
    - Capture any flow that is a subtype of $c$
    - store the exact flow type in flow var $v_1$ and the result of $e_1$ in $v_2$
  - Raise: $ft#x$
    - Change the current flow in the flow type indicated by $ft$ and set the result to $x$
    - Translate flow altering constructs into # constructs

\[
\begin{align*}
x & \rightarrow_T \text{norm}\#x \\
\text{break} & \rightarrow_T \text{brk}\#() \\
\text{break } L & \rightarrow_T \text{brk}−L\#() \\
\text{continue} & \rightarrow_T \text{cont}\#() \\
\text{continue } L & \rightarrow_T \text{cont}−L\#() \\
\text{ret}−i \ x & \rightarrow_T \text{ret}−i\#x \\
\text{raise } v & \rightarrow_T \text{type}(v)\#v \\
\text{raise new } c & \rightarrow_T c\#\text{new } c
\end{align*}
\]
Verification Rules

\[ \frac{}{\vdash \{\Delta\} e_1 \{\Delta_1\} \quad (\Delta_2, \Delta_3) = \text{split}(\Delta_1, c, f v, v) \quad \vdash \{\Delta_2\} e_2 \{\Delta_4\}}{\vdash \{\Delta\} \text{try } e_1 \text{ catch } (c \oplus f v \ v) \ e_2 \ \exists v, f v \cdot (\Delta_3 \lor \Delta_4)} \]

\[ \frac{\Delta_1 = (\exists \text{flow}. [res \rightarrow v] \Delta \land \text{flow} < c \land \text{flow} = f v) \quad \Delta_2 = \Delta \land \neg (\text{flow} < c)}{\text{split}(\Delta, c, f v, v) = (\Delta_1, \Delta_2)} \]

\[ \frac{\text{resolve}(\Delta, ft) = f \text{ set}}{\Delta_1 = \exists \text{res}(\Delta \land \text{flow} = f \text{ set}) \land \text{res} = v'} \]

\[ \vdash \{\Delta\} ft \# v \{\Delta_1\} \]

\[ \frac{(\text{type}(v) = c) \in \Delta}{\text{resolve}(\Delta, \text{ty}(v)) = c} \]

\[ \frac{\text{resolve}(\Delta, \text{Ex}(c)) = \text{Ex}(c)}{\text{resolve}(\Delta, \text{Ex}(c)) = \text{Ex}(c)} \]

\[ \frac{(f v = f \text{ set}) \in \Delta}{\text{resolve}(\Delta, f v) = f \text{ set}} \]
Experiments

• Successfully verified small test examples from:
  – KeY project, exercising specific features
  – SPEC benchmarks, broad range exception handling

<table>
<thead>
<tr>
<th>Programs</th>
<th>LOC</th>
<th>Time (seconds)</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break</td>
<td>20</td>
<td>0.11</td>
<td>break handling</td>
</tr>
<tr>
<td>MyClass</td>
<td>33</td>
<td>0.10</td>
<td>exception hierarchy</td>
</tr>
<tr>
<td>While</td>
<td>130</td>
<td>2.47</td>
<td>while loops and break</td>
</tr>
<tr>
<td>ContinueLabel</td>
<td>100</td>
<td>0.95</td>
<td>imbricated while loops and continue</td>
</tr>
<tr>
<td>PayCard</td>
<td>70</td>
<td>0.91</td>
<td>general exception handling</td>
</tr>
<tr>
<td>SPECjvm2008</td>
<td>190</td>
<td>1.20</td>
<td>general exception handling</td>
</tr>
</tbody>
</table>