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

Motivating Example

```
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;   }
```

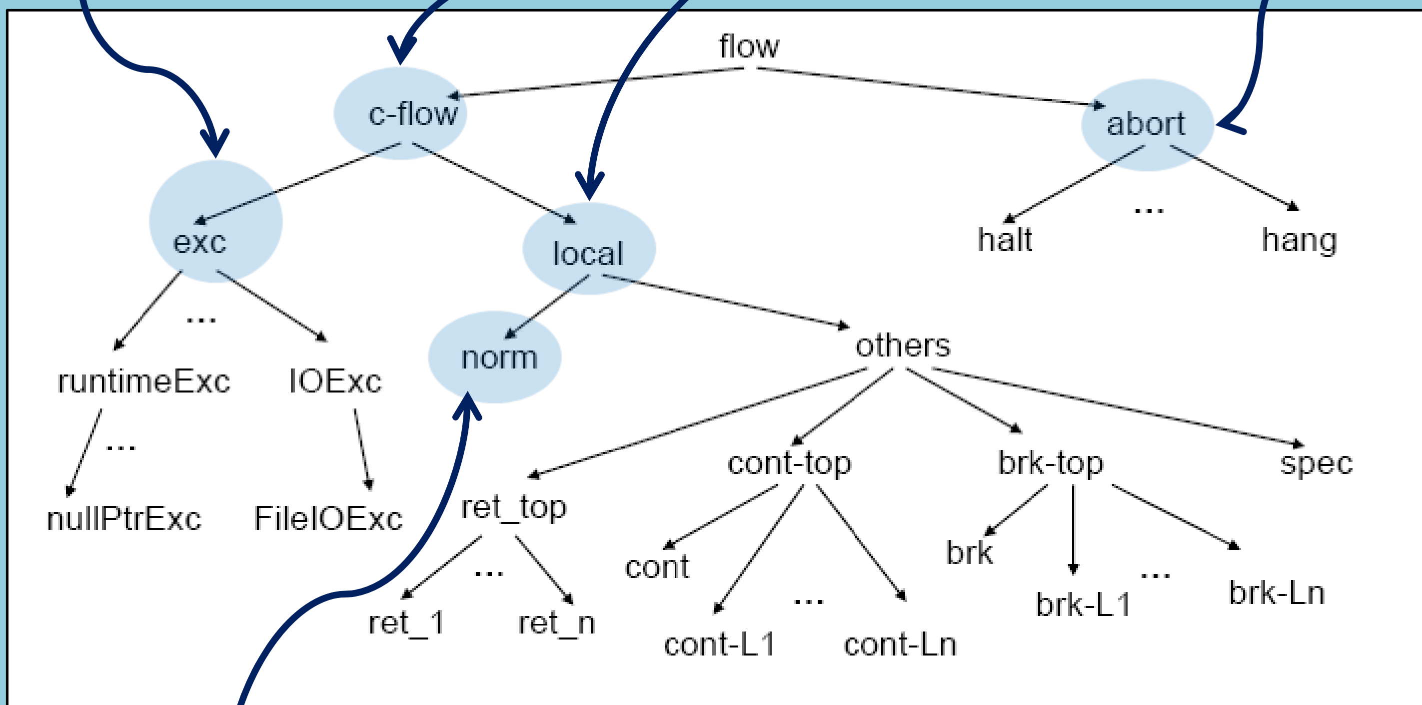
Unified Control Flow Hierarchy

dynamic control flows due to exceptions

can be caught

static control flows

cannot be caught



normal execution

Specification Formulae

- Based on separation logic formulae Φ
- Enriched with
  - Flow variables, used to capture the exact flow type at different program points
  - τ captures the current flow
  - β captures constraints on flow variables

$$
\begin{array}{lll}
\Phi & ::= \bigvee(\exists v^*{\cdot}\kappa{\wedge}\pi)^* & \qquad\qquad \pi ::= \gamma{\wedge}\phi{\wedge}\tau \\
\gamma & ::= v_1{=}v_2 \mid v{=}\mathtt{null} \mid v_1{\neq}v_2 \mid v{\neq}\mathtt{null} \mid \gamma_1{\wedge}\gamma_2 & \\
\kappa & ::= \mathtt{emp} \mid v{::}c\langle v^*\rangle \mid \kappa_1 * \kappa_2 \qquad \beta \;::=\; fv{=}c \mid fv_1{=}fv_2 & \\
\Delta & ::= \Phi \mid \Delta_1{\vee}\Delta_2 \mid \Delta{\wedge}\pi \mid \Delta_1{*}\Delta_2 \mid \exists v{\cdot}\Delta \mid \Delta{\wedge}\beta & \\
\phi & ::= \mathtt{true} \mid \mathtt{false} \mid a_1{=}a_2 \mid a_1{\leq}a_2 \mid c{<}v \mid \phi_1{\wedge}\phi_2 & \\
 & \quad\;\; \mid \phi_1{\vee}\phi_2 \mid \neg\phi \mid \exists v \cdot \phi \mid \forall v \cdot \phi & \\
\tau & ::= \mathtt{flow} \;=\; fset \qquad fset ::= \mathtt{Ex(c)} \mid \mathtt{c} - \{\mathtt{c_1}, .., \mathtt{c_n}\} & \\
a & ::= \; k \mid v \mid k{\times}a \mid a_1{+}a_2 \mid -a \mid max(a_1{,}a_2) \mid min(a_1{,}a_2) &
\end{array}
$$

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

$$
\begin{array}{ll}
x & \Rightarrow_T \texttt{norm}\#x \\
\texttt{break} & \Rightarrow_T \texttt{brk}\#() \\
\texttt{break}\ L & \Rightarrow_T \texttt{brk}-L\#() \\
\texttt{continue} & \Rightarrow_T \texttt{cont}\#() \\
\texttt{continue}\ L & \Rightarrow_T \texttt{cont}-L\#() \\
\texttt{ret}-i\ x & \Rightarrow_T \texttt{ret}-i\#x \\
\texttt{raise}\ v & \Rightarrow_T \texttt{type}(v)\#v \\
\texttt{raise new}\ c & \Rightarrow_T c\#\texttt{new}\ c
\end{array}
$$

Verification Rules

$$\frac{\vdash \{\Delta\}\, e_1\, \{\Delta_1\} \quad (\Delta_2, \Delta_3) = split(\Delta_1, c, fv, v) \quad \vdash \{\Delta_2\}\, e_2\, \{\Delta_4\}}{\vdash \{\Delta\}\, \texttt{try}\; e_1\; \texttt{catch}\; (c@fv\;\; v)\;\; e_2\, \{\exists v, fv \cdot (\Delta_3 \vee \Delta_4)\}}\;[\textbf{FV-TRY-CATCH}]$$

$$\frac{\Delta_1 = (\exists \texttt{flow}\; .[res {\mapsto} v]\Delta \wedge \texttt{flow}{<:}c \wedge \texttt{flow}{=}fv) \quad \Delta_2 = \Delta \wedge \neg(\texttt{flow}\;\; <: c)}{split(\Delta, c, fv, v) = (\Delta_1, \Delta_2)}\;[\textbf{FV-SPLIT}]$$

$$\frac{\begin{array}{c} resolve(\Delta, ft) = fset \\ \Delta_1{=}\exists res.(\Delta \wedge flow = fset) \wedge res = v' \end{array}}{\vdash \{\Delta\}\, ft\#v\, \{\Delta_1\}}\;[\textbf{FV-OUTPUT-VAR}]$$

$$\frac{(type(v) = c) \in \Delta}{resolve(\Delta, \texttt{ty}(v)) = c}\;[\textbf{FV-RESOLVE-TYPE}] \qquad \frac{}{resolve(\Delta, \texttt{Ex}(c)) = \texttt{Ex}(c)}\;[\textbf{FV-RESOLVE-CONST}] \qquad \frac{(fv = fset) \in \Delta}{resolve(\Delta, fv) = fset}\;[\textbf{FV-RESOLVE-FV}]$$

Experiments

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

Programs	LOC	Time (seconds)	Focus
Break	20	0.11	`break` handling
MyClass	33	0.10	exception hierarchy
While	130	2.47	while loops and `break`
ContinueLabel	100	0.95	imbricated while loops and `continue`
PayCard	70	0.91	general exception handling
SPECjvm2008	190	1.20	general exception handling