Formal Methods for Java
Lecture 27: Abnormal Termination in Key

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

February 7, 2013

Abnormal Termination in Java

Abnormal termination in Java is caused by

- a `break` statement,
- a `continue` statement,
- a `return` statement,
- a `throw` statement, or
- a statement that throws a exception.

Abnormal Termination in Dynamic Logic

The formula $\langle\alpha\rangle\phi$ holds,

- iff α terminates normally and ϕ holds afterwards.

The formula $[\alpha]\phi$ holds,

- if α terminates normally and ϕ holds afterwards.
- if α terminates abnormally.
- if α does not terminate at all.

Reasoning about exceptions.

How can we express that statement α throws an exception?

- $\langle\{\alpha\}\rangle\phi$ is equivalent to **false** if α throws an exception or does not terminate
- $[\{\alpha\}]\phi$ is equivalent to **true** if α throws an exception or does not terminate
- The trick is to put an exception handler into the code:

$\langle\{$Throwable $thrown = null;$
    try $\{\alpha;\}$
    catch (Throwable $ex)\{thrown = ex;\}\}\rangle thrown \neq null$

Reasoning with try-catch blocks

When an exception is thrown, the surrounding try blocks become important:

```
\find( \<{ .. try { throw #se; #slist1 }
           catch (#t #v0) { #slist2 } ... }\> post )
```

1. throwing a handled exception: `#se instanceof #t`
   `\replacewith( \<{ .. #t #v0 = #se; #slist2 ... }\> post )`

2. throwing an unhandled exception: `! (#se instanceof #t)`
   `\replacewith( \<{ .. throw #se; ... }\> post )`

3. throwing a null pointer: `#se = null`
   ```
   \replacewith( \< { .. try { throw new NullPointerExc(); #slist1
                    catch (#t #v0) { #slist2 } ... }\> post )
   ```

The KeY system defines a single rule:
```
\replacewith( \< { .. if (#se = null) then
                  try { throw new NullPointerExc(); #slist1
                  catch (#t #v0) { #slist2 }
               else if (#se instanceof #t) then
                 #t v0 = #se; #slist2
               else throw #se;
           ... }\> post )
```

Throw without try-catch blocks

If the surrounding block is not a try block, the block is just removed:
`\find( \<{ .. #label: { throw #se; #slist1 } ... }\> post )`
`\replacewith( \<{ .. throw #se; ... }\> post )`

If there is no surrounding block it depends on modality:

1. total correctness:
   `\find( \<{ throw #se }\> post )`
   `\replacewith( false )`

2. partial correctness:
   `\find( \[{ throw #se }\] post )`
   `\replacewith( true )`

Runtime exceptions

Instructions that throw exceptions are converted to a `throw` instruction:
$\backslash find(\ \backslash \langle\{ \ .. \ \#v[\#se]=\#se0 \ ... \ \}\backslash\rangle \ post)$

- Normal Execution $\#v$ `!=` `null`
  $\backslash add(\ !\#v = null \ \&$
  $\#se < \#v.length \ \& \ \#se >= 0 \ \&$
  $arrayStoreValid(\#v, \#se0) ==>)$
  $\backslash replacewith(\ \backslash\{\#v[\#se] := \#se0\}\backslash\langle\{ \ .. \ ...\}\backslash\rangle \ post)$

- Null Reference $\#v$ `==` `null`
  $\backslash add(\ \#v = null ==>)$
  $\backslash replacewith(\ \backslash\langle\{ \ .. \ $ `throw new` $NullPointerException(); \ ...\}\backslash\rangle \ post)$

- Index Out Of Bounds:
  $\backslash add(\ !\#v = null \ \&$
  $\#se >= \#v.length \ | \ \#se < 0 ==>)$
  $\backslash replacewith(\ \backslash\langle\{ \ .. \ $ `throw new` $ArrIdxOOBException(); \ ...\}\backslash\rangle \ post)$

- Array Store Exception:
  $\backslash add(\ !\#v = null \ \&$
  $\#se < \#v.length \ \& \ \#se >= 0 \ \&$
  $!arrayStoreValid(\#v, \#se0) ==>)$
  $\backslash replacewith(\ \backslash\langle\{ \ .. \ $ `throw new` $ArrayStoreException(); \ ...\}\backslash\rangle \ post)$

Abnormal termination by `break`

The handling of `break` statements is very similar to `try-catch`:

- If the surrounding block has that label, the break is executed:
  $\mathit{find}(\ \backslash\langle\{\ ..\ \#\mathit{label}:\ \{\ \mathtt{break}\ \#\mathit{label};\ \#\mathit{slist1}\ \}\ ...\ \}\backslash\rangle\ \mathit{post}\ )$
  $\mathit{replacewith}(\ \backslash\langle\{\ ..\ ...\ \}\backslash\rangle\ \mathit{post}\ )$

- If the surrounding block has not the right label the block is removed.
  $\mathit{find}(\ \backslash\langle\{\ ..\ \#\mathit{label2}:\ \{\ \mathtt{break}\ \#\mathit{label};\ \#\mathit{slist1}\ \}\ ...\ \}\backslash\rangle\ \mathit{post}\ )$
  $\mathit{replacewith}(\ \backslash\langle\{\ ..\ \mathtt{break}\ \#\mathit{label};\ ...\ \}\backslash\rangle\ \mathit{post}\ )$

- The same for `try-catch` blocks:
  $\mathit{find}(\ \backslash\langle\{\ ..\ \mathtt{try}\ \{\ \mathtt{break}\ \#\mathit{label};\ \#\mathit{slist1}\ \}$
  $\qquad\qquad \mathtt{catch}\ (\#t\ \#v)\ \{\ \#\mathit{slist2}\ \}\ ...\ \}\backslash\rangle\ \mathit{post}\ )$
  $\mathit{replacewith}(\ \backslash\langle\{\ ..\ \mathtt{break}\ \#\mathit{label};\ ...\ \}\backslash\rangle\ \mathit{post}\ )$

Loops with break/continue

break/continue statements are translated to labelled break.

rule loop_unwind:

```
\find( \<{ .. while (#expr) {.... continue; .... break; ....} ... }\> post
\replacewith( \<{ .. if (#expr) {
                    #lab1: {
                      #lab2: {
                          ....
                          break #lab2;
                          ....
                          break #lab1;
                          ....
                      }
                      while (#expr) {.... continue; .... break; ....}
                } } ... } \> post)
```