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 $\alpha$ terminates normally and $\phi$ holds afterwards.

The formula $[\alpha] \phi$ holds,
- if $\alpha$ terminates normally and $\phi$ holds afterwards.
- if $\alpha$ terminates abnormally.
- if $\alpha$ does not terminate at all.
Reasoning about exceptions.

How can we express that statement $\alpha$ throws an exception?

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

  $\langle\{\text{Throwables } \text{thrown }= \text{null};
  
  \text{try } \{} \alpha; \} \n  
  \text{catch (Throwables ex)}\{\text{thrown }= \text{ex}; \} \} \rangle \text{thrown }\neq \text{null}$
Reasoning with try-catch blocks

When an exception is thrown, the surrounding try blocks become important:
\begin{verbatim}
find( \{ .. try { throw #se; #slist1 } catch (#t #v0) { #slist2 } ... }\} > post )
\end{verbatim}

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

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

3 throwing a null pointer: \#se = null
\begin{verbatim}
replacewith( \{ .. try { throw new NullPointerException(); #slist1 catch (#t #v0) { #slist2 } ... }\} > post )
\end{verbatim}

The KeY system defines a single rule:
\begin{verbatim}
replacewith( \{ .. if (#se = null) then
try { throw new NullPointerException(); #slist1
catch (#t #v0) { #slist2 }
else if (#se instanceof #t) then
#t #v0 = #se; #slist2
else throw #se;
... }\} > post )
\end{verbatim}
Throw without try-catch blocks

If the surrounding block is not a try block, the block is just removed:
\[ \text{find( } \langle \{ .. \#label: \{ \text{throw } #se; \#slist1 \} .. \}\rangle \text{ post )} \]
\[ \text{replacewith( } \langle \{ .. \text{throw } #se; \ldots \}\rangle \text{ post )} \]

If there is no surrounding block it depends on modality:

1 total correctness:
\[ \text{find( } \langle \{ \text{throw } #se \}\rangle \text{ post )} \]
\[ \text{replacewith( } \text{false } \) \]

2 partial correctness:
\[ \text{find( } \langle \{ \text{throw } #se \}\rangle \text{ post )} \]
\[ \text{replacewith( } \text{true } \) \]
Runtime exceptions

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

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

- **Null Reference** `\#v == null`
  \[\text{\texttt{add}( \#v = \texttt{null} \==> )}\]
  \[\text{\texttt{replacewith}( \langle\{ . . \texttt{throw new NullPointerException(); } . . \texttt{)}\rangle \text{ post } )}\]

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

- **Array Store Exception**:
  \[\text{\texttt{add}( !\#v = \texttt{null} \&}
  
  \#se < \#v.length \& \#se >= 0 \&
  
  !arrayStoreValid(\#v, \#se0) \==>\)
  
  \text{\texttt{replacewith}( \langle\{ . . \texttt{throw new ArrayStoreException(); } . . \texttt{)}\rangle \text{ post } )}\]
Abnormal termination by \texttt{break}

The handling of \texttt{break} statements is very similar to \texttt{try-catch}:

- If the surrounding block has that label, the \texttt{break} is executed:
  \begin{verbatim}
  \texttt{\textbf{find}( \{ .. \#label: \{ break \#label; \#slist1 \} .. \} )\rightarrow post )}
  \texttt{\textbf{replacewith}( \{ .. .. \} )\rightarrow post )}
  \end{verbatim}

- If the surrounding block has not the right label the block is removed.
  \begin{verbatim}
  \texttt{\textbf{find}( \{ .. \#label2: \{ break \#label; \#slist1 \} .. \} )\rightarrow post )}
  \texttt{\textbf{replacewith}( \{ .. break \#label; .. \} )\rightarrow post )}
  \end{verbatim}

- The same for \texttt{try-catch} blocks:
  \begin{verbatim}
  \texttt{\textbf{find}( \{ .. try \{ break \#label; \#slist1 \}
  \texttt{\hspace{0.5cm} catch (#t #u) \{ \#slist2 \} .. \} )\rightarrow post )}
  \texttt{\textbf{replacewith}( \{ .. break \#label; .. \} )\rightarrow post )}
  \end{verbatim}
Loops with `break/continue`

`break/continue` statements are translated to labelled `break`.

**rule loop_unwind:**

\[
\text{\textbackslash find}( \langle \{ \ldots \text{while} (\#expr) \{ \ldots \text{continue; \ldots break; \ldots} \ldots \} \ldots \rangle) \Rightarrow \text{post} \\
\text{\textbackslash replacewith}( \langle \{ \ldots \text{if} (\#expr) \{ \\
\quad \#lab1: \{ \\
\quad \quad \#lab2: \{ \\
\quad \quad \quad \ldots \\
\quad \quad \quad \quad \text{break} \ #lab2; \\
\quad \quad \quad \ldots \\
\quad \quad \quad \text{break} \ #lab1; \\
\quad \quad \ldots \\
\quad \} \\
\quad \text{while} (\#expr) \{ \ldots \text{continue; \ldots break; \ldots} \ldots \} \\
\} \ldots \rangle) \Rightarrow \text{post})
\]