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`.
Reasoning about exceptions

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

- The trick is to put an exception handler into the code:

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

When an exception is thrown, the surrounding try blocks become important:
\[\text{find( } \langle \text{ ... try } \{ \text{ throw } \#se; \#slist1 \} \text{ catch (}\#t \#v0) \{ \#slist2 \} ... \text{ )} \rangle \text{ post } \]

1. throwing a handled exception: \#se instanceof \#t
   \[\text{replacewith( } \langle \text{ ... } \#t \#v0 = \#se; \#slist2 ... \text{ )} \rangle \text{ post } \]

2. throwing an unhandled exception: ! (\#se instanceof \#t)
   \[\text{replacewith( } \langle \text{ ... throw } \#se; ... \text{ )} \rangle \text{ post } \]

3. throwing a null pointer: \#se = null
   \[\text{replacewith( } \langle \text{ ... try } \{ \text{ throw new NullPointerException(); } \#slist1 \text{ catch (}\#t \#v0) \{ \#slist2 \} ... \text{ )} \rangle \text{ post } \]

The KeY system defines a single rule:
\[\text{replacewith( } \langle \text{ ... if (}\#se = \text{null)} \text{ then} \text{ try } \{ \text{ throw new NullPointerException(); } \#slist1 \text{ catch (}\#t \#v0) \{ \#slist2 \} \text{ else if (}\#se \text{ instanceof } \#t) \text{ then} \text{ } \#t \#v0 = \#se; \#slist2 \text{ else throw } \#se; \}\text{ post } \]

Jochen Hoenicke (Software Engineering)
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:
\(\text{find( \langle\{ .. \#v[\#se]=\#se0 ... \}\rangle \ post )}\)

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

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

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

- **Array Store Exception:**
  \(\text{add( \!\#v = \text{null} \ & \}
  \#se < \#v\.\text{length} \ & \#se \geq 0 \ & \}
  \text{!arrayStoreValid(\#v, \#se0) ==}) \)
  \(\text{replacewith( \langle\{ .. \text{throw new ArrayStoreException(); ...}\}\rangle \ post )}\)
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:
  \[\text{\texttt{find}}( \texttt{\{ .. \#label: \{ break \#label; \#slist1 \} .. \}}) \text{\texttt{post}})\]
  \[\text{\texttt{replacewith}}( \texttt{\{ .. .. \}}) \text{\texttt{post}})\]

- If the surrounding block has not the right label the block is removed.
  \[\text{\texttt{find}}( \texttt{\{ .. \#label2: \{ break \#label; \#slist1 \} .. \}}) \text{\texttt{post}})\]
  \[\text{\texttt{replacewith}}( \texttt{\{ .. break \#label; .. \}}) \text{\texttt{post}})\]

- The same for try-catch blocks:
  \[\text{\texttt{find}}( \texttt{\{ .. try \{ break \#label; \#slist1 \}}\]
  \hspace{2em} \texttt{catch (\#t \#u) \{ \#slist2 \} .. \}}) \text{\texttt{post}})\]
  \[\text{\texttt{replacewith}}( \texttt{\{ .. break \#label; .. \}}) \text{\texttt{post}})\]
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: {
      ....
      continue #lab2;
      ....
      break #lab1;
      ....
    }
  }
  while (#expr) { .... continue; .... break; ....}
} ... } \post)
In KeY, the default rule is to inline the procedures.

Advantages:

- No function contract needed.
- No separate proof for correctness of function needed.

But it has several disadvantages:

- Proof gets larger (especially important if proof is interactive).
- Proof has to be repeated for every function call.
- No recursive procedures possible.
The rule “Use Operation Contract” allows compositional proofs. It opens three subgoals:

- **Pre**: Show that pre-condition holds (this includes class invariants).
- **Post**: Show that with the post-condition, the remaining program is correct.
- **Exceptional Post**: Show that if called method throws an exception, the remaining program is correct.

**Note**: Use Operation Contract cannot be used for the method you are just proving.