Unreachable statements

JLS §14.20: Conservative flow analysis at compile-time.

Static predicates:

$$\mathit{reachable}(\alpha) \iff \text{the phrase at position } \alpha \text{ is reachable}$$
$$\mathit{normal}(\alpha) \iff \text{the phrase at position } \alpha \text{ can complete normally}$$

Fact: $\mathit{normal}(\alpha)$ implies $\mathit{reachable}(\alpha)$.

Conditional compilation:

$$\texttt{while (false)}^{\alpha}\,\mathit{stm} \implies \mathit{reachable}(\alpha) = \mathit{False}$$
$$\texttt{if (false)}^{\alpha}\,\mathit{stm} \implies \mathit{reachable}(\alpha) = \mathit{True}$$

Constraints for method bodies:

$$\mathit{reachable}(\mathit{firstPos}) = \mathit{True}$$
$$\mathit{normal}(\mathit{firstPos}) = \mathit{False}$$

Reachability constraints

$^{\alpha};$	$normal(\alpha) \Leftrightarrow reachable(\alpha)$
$^{\alpha}(^{\beta}exp;)$	$normal(\alpha) \Leftrightarrow reachable(\alpha)$
$^{\alpha}\{^{\beta_1}stm_1 \ldots {}^{\beta_n}stm_n\}$	$reachable(\beta_1) \Leftrightarrow reachable(\alpha)$, $reachable(\beta_{i+1}) \Leftrightarrow normal(\beta_i)$, $normal(\alpha) \Leftrightarrow normal(\beta_n)$
$^{\alpha}$`if` $(^{\beta}exp)\ {}^{\gamma}stm_1$ `else` $^{\delta}stm_2$	$reachable(\gamma) \Leftrightarrow reachable(\alpha)$, $reachable(\delta) \Leftrightarrow reachable(\alpha)$, $normal(\alpha) \Leftrightarrow normal(\gamma) \vee normal(\delta)$
$^{\alpha}$`while` $(^{\beta}exp)\ {}^{\gamma}stm$	$reachable(\gamma) \Leftrightarrow reachable(\alpha)$ and $^{\beta}exp$ is not a constant expression with value $False$, $normal(\alpha) \Leftrightarrow reachable(\alpha)$ and $^{\beta}exp$ is not a constant expression with value $True$

Reachability constraints (continued)

$^{\alpha}lab\colon {}^{\beta}stm$	$reachable(\beta) \Leftrightarrow reachable(\alpha)$, $normal(\alpha) \Leftrightarrow normal(\beta)$ or there exists a reachable statement `break` lab inside $^{\beta}stm$ that can exit $^{\beta}stm$
$^{\alpha}$`break` lab;	$\neg\ normal(\alpha)$
$^{\alpha}$`continue` lab;	$\neg\ normal(\alpha)$
$^{\alpha}$`return`;	$\neg\ normal(\alpha)$
$^{\alpha}$`return` $^{\beta}exp$;	$\neg\ normal(\alpha)$
$^{\alpha}$`throw` $^{\beta}exp$;	$\neg\ normal(\alpha)$

Reachability constraints (continued)

${}^{\alpha}\texttt{try}\ {}^{\beta}block_t$ $\texttt{catch}\ (E_1\ x_1)\ {}^{\gamma_1}block_1$ $\vdots$ $\texttt{catch}\ (E_n\ x_n)\ {}^{\gamma_n}block_n$	$reachable(\beta) \Leftrightarrow reachable(\alpha)$, $reachable(\gamma_i) \Leftrightarrow reachable(\alpha)$ and $E_i \not\preceq_{\mathrm{h}} E_j$ for $1 \leq j < i$ and $block_t$ can throw an exception F with $F \preceq_{\mathrm{h}} E_i$ or $E_i \preceq_{\mathrm{h}} F$, $normal(\alpha) \Leftrightarrow normal(\beta) \vee \bigvee_{1 \leq i \leq n} normal(\gamma_i)$
${}^{\alpha}({}^{\beta}stm\ \texttt{finally}\ {}^{\gamma}block)$	$reachable(\beta) \Leftrightarrow reachable(\alpha)$, $reachable(\gamma) \Leftrightarrow reachable(\alpha)$, $normal(\alpha) \Leftrightarrow normal(\beta) \wedge normal(\gamma)$
${}^{\alpha}\texttt{synchronized}\ ({}^{\beta}exp)$ ${}^{\gamma}stm$	$reachable(\gamma) \Leftrightarrow reachable(\alpha)$, $normal(\alpha) \Leftrightarrow normal(\gamma)$

“can exit” and “can throw”

Definition. An abruption at position α can exit stm, if for every substatement ${}^{\beta}({}^{\gamma}s\ \texttt{finally}\ {}^{\delta}b)$ of stm such that α is in s the predicate $normal(\delta)$ is true.

Definition. A statement stm can throw an exception E, if one of the following conditions is true:

- $E = \texttt{RuntimeException}$ or $E = \texttt{Error}$
- stm contains a reachable statement ${}^{\alpha}\texttt{throw}\,{}^{\beta}exp$ such that $\mathcal{T}(\beta) = E$, the exception E is not caught in stm and an abruption at position α can exit stm
- stm contains a reachable method invocation ${}^{\alpha}c/m(exps)$ such that E occurs in the `throws` clause of m in c, the exception E is not caught in stm and an abruption at position α can exit stm.

The rules of definite assignment (JLS §16)

$x \in \mathit{before}(\alpha)$
: The variable x is definitely assigned before the evaluation of the statement or expression at position α.

$x \in \mathit{after}(\alpha)$
: The variable x is definitely assigned after the statement or expression at position α when this statement or expression completes normally.

$x \in \mathit{true}(\alpha)$
: The variable x is definitely assigned after the evaluation of the expression at position α when this expression evaluates to true.

$x \in \mathit{false}(\alpha)$
: The variable x is definitely assigned after the evaluation of the expression at position α when this expression evaluates to false.

$x \in \mathit{vars}(\alpha)$
: The position α is in the scope of the local variable, formal parameter or `catch` parameter x.

Definite assignment for boolean expressions

${}^{\alpha}\mathtt{true}$	$\mathit{true}(\alpha) = \mathit{before}(\alpha)$, $\mathit{false}(\alpha) = \mathit{vars}(\alpha)$
${}^{\alpha}\mathtt{false}$	$\mathit{true}(\alpha) = \mathit{vars}(\alpha)$, $\mathit{false}(\alpha) = \mathit{before}(\alpha)$
${}^{\alpha}(\,!\,{}^{\beta}e)$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{true}(\alpha) = \mathit{false}(\beta)$, $\mathit{false}(\alpha) = \mathit{true}(\beta)$
${}^{\alpha}({}^{\beta}e_0\,?\,{}^{\gamma}e_1 : {}^{\delta}e_2)$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{before}(\gamma) = \mathit{true}(\beta)$, $\mathit{before}(\delta) = \mathit{false}(\beta)$, $\mathit{true}(\alpha) = \mathit{true}(\gamma) \cap \mathit{true}(\delta)$, $\mathit{false}(\alpha) = \mathit{false}(\gamma) \cap \mathit{false}(\delta)$

Constraint: $\mathcal{T}(\alpha) = \mathtt{boolean}$

Constraint: $\mathit{after}(\alpha) = \mathit{true}(\alpha) \cap \mathit{false}(\alpha)$

Definite assignment for boolean expressions (continued)

${}^{\alpha}({}^{\beta}e_1\ \texttt{\&\&}\ {}^{\gamma}e_2)$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{before}(\gamma) = \mathit{true}(\beta)$, $\mathit{true}(\alpha) = \mathit{true}(\gamma)$, $\mathit{false}(\alpha) = \mathit{false}(\beta) \cap \mathit{false}(\gamma)$
${}^{\alpha}({}^{\beta}e_1\ \texttt{||}\ {}^{\gamma}e_2)$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{before}(\gamma) = \mathit{false}(\beta)$, $\mathit{true}(\alpha) = \mathit{true}(\beta) \cap \mathit{true}(\gamma)$, $\mathit{false}(\alpha) = \mathit{false}(\gamma)$

Constraint: If $\mathcal{T}(\alpha) = \texttt{boolean}$ and ${}^{\alpha}\mathit{exp}$ is of a different kind, then $\mathit{true}(\alpha) = \mathit{after}(\alpha)$ and $\mathit{false}(\alpha) = \mathit{after}(\alpha)$.

Definite assignment for arbitrary expressions

${}^{\alpha}loc$	$after(\alpha) = before(\alpha)$, $loc \in before(\alpha)$
${}^{\alpha}lit$	$after(\alpha) = before(\alpha)$
${}^{\alpha}(loc = {}^{\beta}e)$	$before(\beta) = before(\alpha)$, $loc \in vars(\alpha)$, $after(\alpha) = after(\beta) \cup \{loc\}$
${}^{\alpha}({}^{\beta}e_0\,?\,{}^{\gamma}e_1 : {}^{\delta}e_2)$	$before(\beta) = before(\alpha)$, $before(\gamma) = true(\beta)$, $before(\delta) = false(\beta)$, $after(\alpha) = after(\gamma) \cap after(\delta)$
${}^{\alpha}c.f$	$after(\alpha) = before(\alpha)$

Definite assignment for arbitrary expressions

Constraints for an expression ${}^{\alpha}exp$ with direct subexpressions ${}^{\beta_1}exp_1, \ldots, {}^{\beta_n}exp_n$:

- $before(\beta_1) = before(\alpha)$,
- $before(\beta_{i+1}) = after(\beta_i)$ for $i = 1, \ldots, n-1$,
- $after(\alpha) = after(\beta_n)$.

The direct subexpressions of an expression

Expression at position α	Direct subexpressions of α
${}^{\alpha}(uop\ {}^{\beta}exp)$	${}^{\beta}exp$
${}^{\alpha}({}^{\beta}exp_1\ bop\ {}^{\gamma}exp_2)$	${}^{\beta}exp_1, {}^{\gamma}exp_2$
${}^{\alpha}(c.f = {}^{\beta}exp)$	${}^{\beta}exp$
${}^{\alpha}({}^{\beta_1}exp_1, \ldots, {}^{\beta_n}exp_n)$	${}^{\beta_1}exp_1, \ldots, {}^{\beta_n}exp_n$
${}^{\alpha}(c.m^{\beta}(exps))$	${}^{\beta}(exps)$

The direct subexpressions of an expression

Expression at position α	Direct subexpressions of α
${}^{\alpha}({}^{\beta}exp\ \texttt{instanceof}\ c)$	${}^{\beta}exp$
${}^{\alpha}((c)\ {}^{\beta}exp)$	${}^{\beta}exp$
${}^{\alpha}({}^{\beta}exp.c/f)$	${}^{\beta}exp$
${}^{\alpha}({}^{\beta}exp_1.c/f = {}^{\gamma}exp_2)$	${}^{\beta}exp_1,\ {}^{\gamma}exp_2$
${}^{\alpha}({}^{\beta}exp.c/m^{\gamma}(exps))$	${}^{\beta}exp,\ {}^{\gamma}(exps)$
${}^{\alpha}\texttt{new}\ c.c/m^{\beta}(exps)$	${}^{\beta}(exps)$
${}^{\alpha}({}^{\beta}exp_1[{}^{\gamma}exp_2])$	${}^{\beta}exp_1,\ {}^{\gamma}exp_2$
${}^{\alpha}({}^{\beta}exp_1[{}^{\gamma}exp_2] = {}^{\delta}exp_3)$	${}^{\beta}exp_1,\ {}^{\gamma}exp_2,\ {}^{\delta}exp_3$
${}^{\alpha}(\texttt{new}\ A[{}^{\beta_1}exp_1]\ldots[{}^{\beta_n}exp_n][]\ldots[])$	${}^{\beta_1}exp_1, \ldots, {}^{\beta_n}exp_n$

Definite assignment for statements

$^{\alpha};$	$\mathit{after}(\alpha) = \mathit{before}(\alpha)$
$^{\alpha}(^{\beta}\mathit{exp};)$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{after}(\alpha) = \mathit{after}(\beta)$
$^{\alpha}\{^{\beta_1}\mathit{stm}_1 \ldots {}^{\beta_n}\mathit{stm}_n\}$	$\mathit{before}(\beta_1) = \mathit{before}(\alpha)$, $\mathit{before}(\beta_{i+1}) = \mathit{after}(\beta_i)$ for $i = 1, \ldots, n-1$, $\mathit{after}(\alpha) = \mathit{after}(\beta_n) \cap \mathit{vars}(\alpha)$
$^{\alpha}$`if` $(^{\beta}\mathit{exp})\,^{\gamma}\mathit{stm}_1$ `else` $^{\delta}\mathit{stm}_2$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{before}(\gamma) = \mathit{true}(\beta)$, $\mathit{before}(\delta) = \mathit{false}(\beta)$, $\mathit{after}(\alpha) = \mathit{after}(\gamma) \cap \mathit{after}(\delta)$
$^{\alpha}$`while` $(^{\beta}\mathit{exp})\,^{\gamma}\mathit{stm}$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{before}(\gamma) = \mathit{true}(\beta)$, $\mathit{after}(\alpha) = \mathit{false}(\beta)$

Definite assignment for statements (continued)

${}^{\alpha}lab\colon {}^{\beta}stm$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{after}(\alpha) = \mathit{after}(\beta) \cap \mathit{break}(\beta, lab)$
${}^{\alpha}\texttt{break}\ lab;$	$\mathit{after}(\alpha) = \mathit{vars}(\alpha)$
${}^{\alpha}\texttt{continue}\ lab;$	$\mathit{after}(\alpha) = \mathit{vars}(\alpha)$
${}^{\alpha}\texttt{return};$	$\mathit{after}(\alpha) = \mathit{vars}(\alpha)$
${}^{\alpha}\texttt{return}\ {}^{\beta}exp;$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{after}(\alpha) = \mathit{vars}(\alpha)$
${}^{\alpha}\texttt{throw}\ {}^{\beta}exp;$	$\mathit{before}(\beta) = \mathit{before}(\alpha)$, $\mathit{after}(\alpha) = \mathit{vars}(\alpha)$

Definition: $x \in \mathit{break}(\alpha, lab)\ :\Longleftrightarrow$

- x is in $\mathit{before}(\beta)$ for each statement ${}^{\beta}\texttt{break}\ lab$ inside the statement at position α that can exit α and
- x is in $\mathit{after}(\beta)$ for each statement ${}^{\beta}(s\ \texttt{finally}\ b)$ inside α such that s contains a $\texttt{break}\ lab$ that can exit α.

Definite assignment for statements (continued)

${}^{\alpha}\texttt{try}\ {}^{\beta}block_t$ $\texttt{catch}\ (E_1\ x_1)\ {}^{\gamma_1}block_1$ $\vdots$ $\texttt{catch}\ (E_n\ x_n)\ {}^{\gamma_n}block_n$	$before(\beta) = before(\alpha)$, $before(\gamma_i) = before(\alpha) \cup \{x_i\}$, $after(\alpha) = after(\beta) \cap \bigcap_{1 \le i \le n} after(\gamma_i)$
${}^{\alpha}({}^{\beta}stm\ \texttt{finally}\ {}^{\gamma}block)$	$before(\beta) = before(\alpha)$, $before(\gamma) = before(\alpha)$, $after(\alpha) = \{x \in after(\beta) \mid$ there is no $x = exp$ in ${}^{\gamma}block\} \cup after(\gamma)$
${}^{\alpha}\texttt{synchronized}\ ({}^{\beta}exp)$ ${}^{\gamma}stm$	$before(\beta) = before(\alpha)$, $before(\gamma) = after(\beta)$, $after(\alpha) = after(\gamma)$

Run-time compatible

Definition. $A \sqsubseteq B \;:\Longleftrightarrow$ one of the following conditions is true:

- A and B are primitive types and $A = B$
- A and B are reference types and $A \preceq B$

Lemma:

- $A \sqsubseteq A$.
- If $A \sqsubseteq B$ and $B \sqsubseteq C$, then $A \sqsubseteq C$.
- If $A \sqsubseteq B$ and $B \sqsubseteq A$, then $A = B$.
- $A[\,] \sqsubseteq B[\,] \Leftrightarrow A \sqsubseteq B$.

Definition. f is a frame in state n of thread q, iff one of the following conditions is true:

- $f = (meth_n^q, restbody_n^q, pos_n^q, locals_n^q)$
- f is an element of $frames_n^q$

Reference is used in a state

Definition. A reference ref is used in state n, iff one of the following conditions is true:

- there exists a field c/f such that $globals_n(c/f) = ref$
- there exists an r and a field c/f such that $getField_n(r, c/f) = ref$
- there exists an r and an $i \in \mathbb{N}$ such that $getElement_n(r, i) = ref$
- there exists a frame $(\_, restbody^*, \_, locals^*)$ in state n of a thread q and one of the following conditions is true:
  - there exists a variable loc such that $locals^*(loc) = ref$
  - there exists a position α such that $restbody^*/\alpha = ref$
  - there exists a position α such that $restbody^*/\alpha = Return(ref)$
  - there exists a position α such that $restbody^*/\alpha = Exc(ref)$

Theorem: Java is type safe

Theorem. Assume that $(meth^*, restbody^*, pos^*, locals^*)$ is a frame in state n of thread q. Then the following invariants are satisfied:

(def1) $before(pos^*) \subseteq dom(locals^*)$.

(def2) If $restbody^*/pos^*$ is normal, then $after(pos^*) \subseteq dom(locals^*)$.

(def3) If $restbody^*/pos^* = True$, then $true(pos^*) \subseteq dom(locals^*)$.

(def4) If $restbody^*/pos^* = False$, then $false(pos^*) \subseteq dom(locals^*)$.

(def5) If $restbody^*/pos^* = Break(l)$, then $break(pos^*, l) \subseteq dom(locals^*)$.

(def6) If the frame is not the current frame of q and $body(meth^*)/pos^*$ is a method invocation then $after(pos^*) \subseteq dom(locals^*)$.

(reach) $reachable(pos^*)$.

Theorem: Java is type safe (continued)

(norm) If $restbody^*/\alpha = Norm$, then $normal(\alpha)$.

(val) If $restbody^*/\alpha$ is a value of type B, then $B \sqsubseteq \mathcal{T}(\alpha)$, where $\mathcal{T}(\alpha)$ is the compile-time type of position α in $body(meth^*)$.

(undef) The constant $undef$ does not occur in $restbody^*$.

(loc1) If $x \in dom(locals^*)$, then $locals^*(x) \in \mathit{Val}$.

(loc2) If pos^* is in the scope of a local variable declaration of a variable x of type A and $x \in dom(locals^*)$, then $locals^*(x)$ is a value of type $B \sqsubseteq A$.

(loc3) If pos^* is in the scope of a formal parameter x of type A, then $locals^*(x)$ is a value of type $B \sqsubseteq A$.

(loc4) If pos^* is in the scope of a `catch` parameter x of type E, then $locals^*(x)$ is a value of type $F \preceq_{\mathrm{h}} E$.

(loc5) If pos^* is in class A and pos^* is in the body of an instance method or in the body of a constructor, then $locals^*(\mathtt{this})$ is a value of type $B \preceq_{\mathrm{h}} A$.

Theorem: Java is type safe (continued)

(abr1) If $restbody^*/\alpha = Break(l)$, then α is in a statement with label l and $body(meth^*)/\alpha$ contains a reachable `break` lab which can exit $body(meth^*)/\alpha$.

(abr2) If $restbody^*/\alpha = Continue(l)$, then α is in a `while` statement with label l.

(abr3) If $restbody^*/\alpha = Return$, then α is in the body of a method with return type `void`.

(abr4) If $restbody^*/\alpha = Return(v)$, then α is in the body of a method with return type A and v is a value of type $B \sqsubseteq A$.

(abr5) If $restbody^*/\alpha = Exc(ref)$, then $classOf(ref) = E$, $E \preceq_{\mathrm{h}}$ `Throwable`, E is allowed at position α and $body(meth^*)/\alpha$ can throw an exception F such that $E \preceq_{\mathrm{h}} F$.

Theorem: Java is type safe (continued)

Assume that $(\_, restbody^*, \beta, \_)$ is the parent frame of $(c/m, \_, \_, locals^*)$ in state n of thread q. Then the dynamic method invocation chain has the following properties:

(chain1) If the return type of c/m is A and $A \neq \texttt{void}$, then $\mathcal{T}(\beta) = A$.

(chain2) If E occurs in the `throws` clause of c/m, then E is allowed at position β.

(chain3) If c/m is a constructor and $restbody^*/\beta = ref.c/m(\_)$, then $locals^*(\texttt{this}) = ref$.

Theorem: Java is type safe (continued)

The following global invariants are true in state n:

(global) If c/f is a `static` field of declared type A, then $globals_n(c/f)$ is a value of type $B \sqsubseteq A$.

(ref) If a reference ref is used in state n, then $ref \in dom(heap_n)$.

(object1) If $heap_n(ref) = Object(c, fields)$, then c is a non `abstract` class and $dom(fields) = instanceFields(c)$.

(object2) If $heap_n(ref) = Object(\_, fields)$, $fields(f) = v$ and f is of declared type A, then v is a value of type $B \sqsubseteq A$.

(array) If $heap_n(ref) = Array(A, elems)$ and $elems(i) = v$, then v is a value of type $B \sqsubseteq A$.