Threads in Java$_{\mathcal{T}}$

Syntax of Java$_{\mathcal{T}}$

$$\mathit{Stm} := \ldots \mid \texttt{synchronized}\ (\mathit{Exp})\ \mathit{Stm}$$

Native Methods for threads:

- `Thread/start()`
- `Thread/interrupt()`
- `Thread/interrupted()`
- `Thread/isInterrupted()`
- `Object/wait()`
- `Object/notify()`
- `Object/notifyAll()`

Example (Threads)

```
class Author extends Thread {
  private char letter;
  private StringBuffer book;

  Author(StringBuffer b,char c){book = b; letter = c;}
  public void run() {
    synchronized (book) {
      while (book.length() < 40) {
        book.append(letter);
        book.notify();
        try { book.wait(); }
        catch (InterruptedException e) { }
      }
      book.notify();
    }
  }
}
```

Example (Threads continued)

```
class Publisher extends Thread {
  private StringBuffer book;

  Publisher(StringBuffer b) { book = b; }

  public void run() {
    synchronized (book) {
      while (book.length() < 40) {
        book.notify();
        try { book.wait(); }
        catch (InterruptedException e) { }
      }
      System.out.println(book);
      book.notifyAll();
    }
  }
}
```

Example (Threads continued)

```
class Test {
  public static void main(String[] _) {
    StringBuffer book = new StringBuffer(40);
    for (int i = 0; i < 10; ++i) {
      new Author(book,(char)('A' + i)).start();
    }
    new Publisher(book).start();
  }
}
```

Possible output:

```
ABACDEFGHIJBACDEFGHIJBACDEFGHIJBACDEFGHI
```

Vocabulary of the ASM for Java$_{\mathcal{T}}$

Universes:

$$\mathit{Thread} := \{q \in \mathrm{dom}(\mathit{heap}) \mid \mathit{classOf}(q) \preceq_{\mathrm{h}} \texttt{Thread}\}$$

$$\mathit{ThreadState} := \mathit{NotStarted} \mid \mathit{Active} \mid \mathit{Synchronizing} \mid \mathit{Waiting} \mid \mathit{Notified} \mid \mathit{Dead}$$

Dynamic functions:

$$\begin{aligned}
\mathit{thread} &: \mathit{Thread} \\
\mathit{cont} &: \mathit{Thread} \to (\mathit{Frame}^*, \mathit{Frame}) \\
\mathit{exec} &: \mathit{Thread} \to \mathit{ThreadState} \\
\mathit{sync} &: \mathit{Thread} \to \mathit{Ref}^* \\
\mathit{waitSet} &: \mathit{Ref} \to \mathit{Powerset}(\mathit{Thread}) \\
\mathit{locks} &: \mathit{Ref} \to \mathit{Nat} \\
\mathit{syncObj} &: \mathit{Thread} \to \mathit{Ref} \\
\mathit{waitObj} &: \mathit{Thread} \to \mathit{Ref}
\end{aligned}$$

$$\mathit{interruptedFlag}: \mathit{Thread} \to \mathit{Bool}$$

Creation of new objects

$$
\begin{array}{l}
execJavaExp_O = \textbf{case } context(pos) \textbf{ of} \\
\quad \texttt{new } c \rightarrow \textbf{if } initialized(c) \textbf{ then create } ref \\
\qquad\qquad heap(ref) := Object(c, \{(f, defaultVal(type(f))) \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid f \in instanceFields(c)\}) \\
\qquad\qquad waitSet(ref) := \emptyset \\
\qquad\qquad locks(ref) \quad := 0 \\
\qquad\qquad \textbf{if } c \preceq_{\mathrm{h}} \texttt{Thread } \textbf{then} \\
\qquad\qquad\quad exec(ref) := NotStarted \\
\qquad\qquad\quad sync(ref) := [\,] \\
\qquad\qquad\quad interruptedFlag(ref) := False \\
\qquad\qquad yield(ref) \\
\qquad\quad \textbf{else } initialize(c)
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (`synchronized`-statement)

$$
\begin{array}{l}
execJavaStm_T = \textbf{case } cont\,ext(pos) \textbf{ of} \\
\quad \texttt{synchronized } (^{\alpha}exp)\,^{\beta}stm \longrightarrow pos := \alpha \\
\quad \texttt{synchronized } (^{\blacktriangleright}ref)\,^{\beta}stm \longrightarrow \\
\qquad \textbf{if } ref = null \textbf{ then } failUp(\texttt{NullPointerException}) \\
\qquad \textbf{else} \\
\qquad\quad \textbf{if } ref \in sync(thread) \textbf{ then} \\
\qquad\qquad sync(thread) := [ref] \cdot sync(thread) \\
\qquad\qquad locks(ref) \quad := locks(ref) + 1 \\
\qquad\qquad pos \qquad\quad := \beta \\
\qquad\quad \textbf{else} \\
\qquad\qquad exec(thread) \quad := Synchronizing \\
\qquad\qquad syncObj(thread) := ref \\
\qquad\qquad cont(thread) \quad := (frames, (meth, restbody, \beta, locals)) \\
\quad \texttt{synchronized } (^{\alpha}ref)\,^{\blacktriangleright}Norm \longrightarrow releaseLock(Norm) \\
\quad \texttt{synchronized } (^{\alpha}ref)\,^{\blacktriangleright}abr \quad \longrightarrow releaseLock(abr)
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (continued)

$$
\begin{array}{l}
releaseLock(phrase) = \\
\quad \textbf{let } [p] \cdot rest = sync(thread) \\
\quad sync(thread) := rest \\
\quad locks(p) \quad\quad := locks(p) - 1 \\
\quad yieldUp(phrase)
\end{array}
$$

$$
\begin{array}{l}
propagatesAbr(phrase) = \\
\quad phrase \neq\ lab : \ s \wedge \\
\quad phrase \neq \texttt{static}\ s \wedge \\
\quad phrase \neq \texttt{try}\ \ldots \wedge \\
\quad phrase \neq\ s_1\ \texttt{finally}\ s_2 \wedge \\
\quad phrase \neq \texttt{synchronized}\ s
\end{array}
$$

Initialization of classes

$$\mathit{ClassState} = \mathit{Linked} \mid \mathit{InProgress} \mid \mathit{Initialized} \mid \mathit{Unusable}$$
$$\mathit{classState}\colon \mathit{Class} \longrightarrow \mathit{ClassState}$$

New dynamic functions:

$$\mathit{initThread} : \mathit{Class} \longrightarrow \mathit{Thread}$$
$$\mathit{initWait} : \mathit{Class} \longrightarrow \mathit{Powerset}(\mathit{Thread})$$

$$\mathit{initialized}(c) =$$
$$\mathit{classState}(c) = \mathit{Initialized} \vee$$
$$(\mathit{classState}(c) = \mathit{InProgress} \wedge \mathit{initThread}(c) = \mathit{thread})$$

Initialization of classes (continued)

$$
\begin{array}{l}
\mathit{initialize}(c) = \\
\quad \textbf{if } \mathit{classState}(c) = \mathit{Linked} \textbf{ then} \\
\quad\quad \mathit{classState}(c) := \mathit{InProgress} \\
\quad\quad \textbf{forall } f \in \mathit{staticFields}(c) \\
\quad\quad\quad \mathit{globals}(f) := \mathit{defaultVal}(\mathit{type}(f)) \\
\quad\quad \mathit{invokeMethod}(\mathit{pos}, c/\texttt{<clinit>}, [\,]) \\
\quad\quad \mathit{initWait}(c) \quad := \emptyset \\
\quad\quad \mathit{initThread}(c) := \mathit{thread} \\
\quad \textbf{if } \mathit{classState}(c) = \mathit{InProgress} \wedge \mathit{initThread}(c) \neq \mathit{thread} \textbf{ then} \\
\quad\quad \mathit{exec}(\mathit{thread}) := \mathit{Waiting} \\
\quad\quad \mathit{cont}(\mathit{thread}) := (\mathit{frames}, (\mathit{meth}, \mathit{restbody}, \mathit{pos}, \mathit{locals})) \\
\quad\quad \mathit{initWait}(c) \;\; := \mathit{initWait}(c) \cup \{\mathit{thread}\} \\
\quad \textbf{if } \mathit{classState}(c) = \mathit{Unusable} \textbf{ then} \\
\quad\quad \mathit{fail}(\texttt{NoClassDefFoundErr})
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (continued)

$$execJavaStm_T = \textbf{case } context(pos) \textbf{ of}$$
$$\quad \texttt{static} \blacktriangleright abr \rightarrow notifyThreadsWaitingForInitialization$$
$$\quad abr \rightarrow \textbf{if } pos = firstPos \wedge null(frames) \textbf{ then } killThread$$

$$killThread =$$
$$\quad waitSet(thread) := \emptyset$$
$$\quad exec(thread) \quad := Dead$$
$$\quad \textbf{forall } q \in waitSet(thread)$$
$$\quad\quad exec(q) := Notified$$

$$notifyThreadsWaitingForInitialization =$$
$$\quad \textbf{let } c = classNm(meth)$$
$$\quad initWait(c) \quad := \emptyset$$
$$\quad initThread(c) := undef$$
$$\quad \textbf{forall } q \in initWait(c)$$
$$\quad\quad exec(q) := Active$$

Transition rules for Java$_{\mathcal{T}}$ (thread scheduling)

$$
\begin{array}{l}
execJavaThread = \\
\quad \textbf{choose } q \in dom(exec), runnable(q) \\
\quad\quad \textbf{if } q = thread \wedge exec(q) = Active \textbf{ then} \\
\quad\quad\quad execJava \\
\quad\quad \textbf{else} \\
\quad\quad\quad \textbf{if } exec(thread) = Active \textbf{ then} \\
\quad\quad\quad\quad cont(thread) := (frames, (meth, restbody, pos, locals)) \\
\quad\quad\quad thread := q \\
\quad\quad\quad run(q)
\end{array}
$$

$$
\begin{array}{l}
runnable(q) = \\
\quad \textbf{case } exec(q) \textbf{ of} \\
\quad\quad Active \quad\quad\quad\quad \rightarrow True \\
\quad\quad Synchronizing \rightarrow locks(syncObj(q)) = 0 \\
\quad\quad Notified \quad\quad\quad \rightarrow locks(waitObj(q)) = 0
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (continued)

$$
\begin{array}{l}
run(q) = \\
\quad switchCont(q) \\
\quad \textbf{if } exec(q) = Synchronizing \textbf{ then} \\
\quad\quad synchronize(q) \\
\quad \textbf{if } exec(q) = Notified \textbf{ then} \\
\quad\quad wakeup(q)
\end{array}
$$

$$
\begin{array}{l}
switchCont(q) = \\
\quad \textbf{let } (frames', (meth', restbody', pos', locals')) = cont(q) \\
\quad exec(q) := Active \\
\quad meth := meth' \\
\quad restbody := restbody' \\
\quad pos := pos' \\
\quad locals := locals' \\
\quad frames := frames'
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (continued)

$$
\begin{aligned}
&synchronize(q) = \\
&\quad sync(q) &&:= [syncObj(q)] \cdot sync(q) \\
&\quad locks(syncObj(q)) &&:= 1
\end{aligned}
$$

$$
\begin{aligned}
&wakeup(q) = \\
&\quad locks(waitObj(q)) &&:= occurrences(waitObj(q), sync(q))
\end{aligned}
$$

Transition rules for Java$_{\mathcal{T}}$ (thread methods)

$$
\begin{array}{l}
invokeNative(meth, values) \\
\quad \mid meth = \texttt{Thread/start()} \quad\quad\quad\quad\quad\quad\quad\quad = start(values(0)) \\
\quad \mid meth = \texttt{Thread/interrupt()} \quad\quad\quad\quad\quad\quad = interrupt(values(0)) \\
\quad \mid meth = \texttt{Thread/interrupted()} \quad\quad\quad\quad\quad = interrupted \\
\quad \mid meth = \texttt{Thread/isInterrupted()} = isInterrupted(values(0)) \\
\quad \mid meth = \texttt{Object/wait()} \quad\quad\quad\quad\quad\quad\quad\quad = wait(values(0)) \\
\quad \mid meth = \texttt{Object/notify()} \quad\quad\quad\quad\quad\quad\quad = notify(values(0)) \\
\quad \mid meth = \texttt{Object/notifyAll()} \quad\quad\quad\quad\quad\quad = notifyAll(values(0))
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (starting a thread)

$$
\begin{array}{l}
start(ref) = \\
\quad \textbf{if } exec(ref) \neq NotStarted \textbf{ then} \\
\quad\quad fail(\texttt{IllegalThreadStateException}) \\
\quad \textbf{else} \\
\quad\quad \textbf{let } q \quad\;\; = getField(ref, \texttt{Thread/ "target" }) \\
\quad\quad\quad\;\; meth = lookup(classOf(q), \texttt{Runnable/run()})\texttt{/run()} \\
\quad\quad exec(ref) := Active \\
\quad\quad cont(ref) := ([\,], (meth, body(meth), firstPos, \{(\texttt{ "this" }, q)\})) \\
\quad\quad yieldUp(Norm)
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (interrupting a thread)

$$
\begin{array}{l}
interrupt(q) = \\
\quad yieldUp(Norm) \\
\quad \textbf{if } exec(q) = Waiting \wedge \neg classInitialization(q) \textbf{ then} \\
\qquad \textbf{let } (frames', (meth', restbody, pos', locals')) = cont(q) \\
\qquad \textbf{let } fail = restbody[\texttt{throw new InterruptedException}(); /pos'] \\
\qquad \textbf{let } ref = waitObj(q) \\
\qquad waitSet(ref) := waitSet(ref) \setminus \{q\} \\
\qquad exec(q) := Notified \\
\qquad cont(q) := (frames', (meth', fail, pos', locals')) \\
\qquad interruptedFlag(q) := False \\
\quad \textbf{else} \\
\qquad interruptedFlag(q) := True
\end{array}
$$

$$classInitialization(q) = q \in ran(initThread) \vee q \in \bigcup ran(initWait)$$

Transition rules for Java$_{\mathcal{T}}$ (interrupt)

$$
\begin{array}{l}
interrupted = \\
\quad \textbf{if } interruptedFlag(thread) \textbf{ then} \\
\quad\quad interruptedFlag(thread) := False \\
\quad\quad yield(True) \\
\quad \textbf{else} \\
\quad\quad yield(False)
\end{array}
$$

$$
\begin{array}{l}
isInterrupted(q) = \\
\quad \textbf{if } interruptedFlag(q) \textbf{ then} \\
\quad\quad yieldUp(True) \\
\quad \textbf{else} \\
\quad\quad yieldUp(False)
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (wait)

$$
\begin{array}{l}
wait(ref) = \\
\quad \textbf{if } ref \notin sync(thread) \textbf{ then} \\
\quad\quad fail(\texttt{IllegalMonitorStateException}) \\
\quad \textbf{else} \\
\quad\quad \textbf{let } ret = restbody[Norm/up(pos)] \\
\quad\quad waitSet(ref) \quad := waitSet(ref) \cup \{thread\} \\
\quad\quad locks(ref) \quad := 0 \\
\quad\quad exec(thread) \quad := Waiting \\
\quad\quad waitObj(thread) := ref \\
\quad\quad cont(thread) \quad := (frames, (meth, ret, up(pos), locals)) \\
\quad\quad yieldUp(Norm)
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (notify)

$$
\begin{array}{l}
\mathit{notify}(\mathit{ref}) = \\
\quad \textbf{if } \mathit{ref} \notin \mathit{sync}(\mathit{thread}) \textbf{ then} \\
\qquad \mathit{fail}(\texttt{IllegalMonitorStateException}) \\
\quad \textbf{else} \\
\qquad \mathit{yieldUp}(\mathit{Norm}) \\
\qquad \textbf{choose } q \in \mathit{waitSet}(\mathit{ref}) \\
\qquad\quad \mathit{waitSet}(\mathit{ref}) := \mathit{waitSet}(\mathit{ref}) \setminus \{q\} \\
\qquad\quad \mathit{exec}(q) \qquad := \mathit{Notified}
\end{array}
$$

Transition rules for Java$_{\mathcal{T}}$ (notifyAll)

$$
\begin{array}{l}
\mathit{notifyAll}(\mathit{ref}) = \\
\quad \textbf{if } \mathit{ref} \notin \mathit{sync}(\mathit{thread}) \textbf{ then} \\
\qquad \mathit{fail}(\texttt{IllegalMonitorStateException}) \\
\quad \textbf{else} \\
\qquad \mathit{waitSet}(\mathit{ref}) := \emptyset \\
\qquad \mathit{yieldUp}(\mathit{Norm}) \\
\qquad \textbf{forall } q \in \mathit{waitSet}(\mathit{ref}) \\
\qquad\quad \mathit{exec}(q) := \mathit{Notified}
\end{array}
$$

Defined predicates

Predicates:

$$synchronizing(q, ref) =$$
$$\quad exec(q) = Synchronizing \wedge syncObj(q) = ref$$
$$waiting(q, ref) =$$
$$\quad exec(q) = Waiting \wedge waitObj(q) = ref$$
$$notified(q, ref) =$$
$$\quad exec(q) = Notified \wedge waitObj(q) = ref$$

$$locked(q, ref) =$$
$$\quad ref \in sync(q) \wedge \neg waiting(q, ref) \wedge \neg notified(q, ref)$$

$$locked(ref) =$$
$$\quad \exists q \in dom(exec)(locked(q, ref))$$

Function: $syncFromCont(q)$

Theorem: Thread invariants

(thread) $classOf(thread) \preceq_{\mathrm{h}} \texttt{Thread}$.

(exec1) $dom(exec)$ is exactly the set of all threads.

(exec2) If $exec(q) = NotStarted$, then $sync(q) = [\,]$.

(sync1) If $ref \in sync(q)$, then $ref \in dom(heap)$.

(sync2) If $synchronizing(q, ref)$, then $ref \in dom(heap)$, $ref \notin sync(q)$ and $[ref] \cdot sync(q) = syncFromCont(q)$.

(sync3) If $exec(q) \neq Synchronizing$, then $sync(q) = syncFromCont(q)$.

(wait1) If $waiting(q, ref)$, then $q \in waitSet(ref)$ and $ref \in sync(q)$.

(wait2) If $q \in waitSet(ref)$, then q is a thread and $waiting(q, ref)$.

Theorem: Thread invariants (continued)

(notified) If $notified(q, ref)$, then $q \notin waitSet(ref)$ and $ref \in sync(q)$.

(dead) If $exec(q) = Dead$, then the frame stack of q is empty, $waitSet(q) = \emptyset$ and $sync(q) = [\,]$.

(lock1) If $locked(q, ref)$, then $locks(ref) = occurrences(ref, sync(q))$.

(lock2) If $locked(q_1, ref)$ and $locked(q_2, ref)$, then $q_1 = q_2$.

(lock3) If $locks(ref) > 0$, then $locked(ref)$.