Formal Specification of Java Concurrency to Assist Software Verification

Brad Long and Ben Long

Software Verification Research Centre

School of Into Tech and Elec Eng

The University of Queensland

Presented by: Paul Strooper

Overview

- Motivation
- Formal (Z) model of Java Concurrency
  - state and operation schemas: request a lock, wait, notify, ...
  - safety and liveness properties: deadlock, starvation, ...
- Discussion

Motivation

- Verification of concurrent programs is difficult
  - inherent non-determinism
  - safety and liveness properties: contention, deadlock, etc.
- Models of Java Concurrency
  - textual models insufficient
  - formal model to support static analysis, run-time checks, test case generation, etc.
- Formal Z model
  - includes specification of general correctness properties

Z Specification – State Schema

System

$$
\begin{array}{l}
active : \mathbb{P}\, THREAD \\
locked : OBJECT \nrightarrow THREAD \\
candidates : THREAD \nrightarrow OBJECT \\
blocked : THREAD \nrightarrow OBJECT \\
waiting : THREAD \nrightarrow OBJECT \\
\hline
\mathrm{ran}\, locked \cup \mathrm{dom}\, blocked \cup \mathrm{dom}\, candidates \cup \mathrm{dom}\, waiting \subseteq active \\
\forall (o, t) : locked \bullet (t, o) \notin (blocked \cup candidates \cup waiting) \\
\mathrm{ran}\, blocked \subseteq \mathrm{dom}\, locked \\
blocked \subseteq candidates \\
\mathrm{dom}\, blocked \cap \mathrm{dom}\, waiting = \varnothing \\
\mathrm{dom}\, candidates \cap \mathrm{dom}\, waiting = \varnothing
\end{array}
$$

Z Specification – Requesting a Lock

$$
\begin{array}{|l}
\hline
\textit{requestLock} \\
\Delta \textit{System} \\
t? : \textit{THREAD};\ o? : \textit{OBJECT} \\
\hline
t? \notin \operatorname{dom} \textit{candidates} \\
((o?, t?) \notin \textit{locked} \Rightarrow \textit{candidates}' = \textit{candidates} \oplus \{(t?, o?)\}) \\
((o?, t?) \in \textit{locked} \Rightarrow \textit{candidates}' = \textit{candidates}) \\
\textit{active}' = \textit{active} \land \textit{locked}' = \textit{locked} \\
\textit{blocked}' = \textit{blocked} \land \textit{waiting}' = \textit{waiting} \\
\hline
\end{array}
$$

Z Specification – Serving a Lock

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{serveLock}} \\
\hline
\Delta System \\
o? : OBJECT \\
\hline
active' = active \land waiting' = waiting \\
(o? \notin \operatorname{ran} candidates \Rightarrow \qquad [1] \\
\quad locked' = locked \land \\
\quad candidates' = candidates \land blocked' = blocked) \\
(o? \in \operatorname{ran} candidates \land o? \notin \operatorname{dom} locked \Rightarrow \qquad [2] \\
\quad \exists\, t : \operatorname{dom} candidates \mid (t, o?) \in candidates \bullet \\
\quad (locked' = locked \oplus \{(o?, t)\} \land \\
\quad candidates' = candidates \setminus \{(t, o?)\} \land \\
\quad blocked' = (blocked \setminus \{(t, o?)\}) \oplus (candidates' \rhd \{o?\}))) \\
(o? \in \operatorname{ran} candidates \land o? \in \operatorname{dom} locked \Rightarrow \qquad [3] \\
\quad locked' = locked \land \\
\quad candidates' = candidates \land \\
\quad blocked' = blocked \oplus (candidates \rhd \{o?\})) \\
\hline
\end{array}
$$

Z Specification – Releasing a Lock

$$
\begin{array}{|l}
\hline
\textit{releaseLock} \\
\Delta System \\
o? : OBJECT \\
\hline
locked' = \{o?\} \lhd locked \\
active' = active \land blocked' = blocked \\
candidates' = candidates \land waiting' = waiting \\
\hline
\end{array}
$$

Z Specification – wait()

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{wait}} \\
\hline
\Delta System \\
o? : OBJECT;\ t? : THREAD \\
\hline
(o?, t?) \in locked \\
waiting' = waiting \oplus \{(t?, o?)\} \\
locked' = locked \setminus \{(o?, t?)\} \\
active' = active \land blocked' = blocked \\
candidates' = candidates \\
\hline
\end{array}
$$

Z Specification – notify()

$$
\begin{array}{l}
\textbf{notify} \\
\hline
\Delta System \\
o? : OBJECT \\
\hline
active' = active \land locked' = locked \land blocked' = blocked \\
o? \in \operatorname{dom} locked \\
(o? \in \operatorname{ran} waiting \Rightarrow \\
\quad \exists t : \operatorname{dom} waiting \mid (t, o?) \in waiting \bullet \\
\quad (waiting' = waiting \setminus \{(t, o?)\} \land \\
\quad candidates' = candidates \oplus \{(t, o?)\})) \\
(o? \notin \operatorname{ran} waiting \Rightarrow \\
\quad (waiting' = waiting \land candidates' = candidates)) \\
\hline
\end{array}
$$

Z Specification – notifyAll()

notifyAll

$\Delta System$
$o? : OBJECT$

$o? \in \operatorname{dom} locked$
$waiting' = waiting \rhd \{o?\}$
$candidates' = candidates \oplus (waiting \rhd \{o?\})$
$active' = active \land locked' = locked \land blocked' = blocked$

Z Specification – System

$$
\begin{aligned}
&sysRequestLock \mathrel{\widehat=} requestLock \mathbin{;} serveLock \\
&sysReleaseLock \mathrel{\widehat=} releaseLock \mathbin{;} serveLock \\
&sysWait \mathrel{\widehat=} wait \mathbin{;} serveLock \\
&sysNotify \mathrel{\widehat=} notify \mathbin{;} serveLock \\
&sysNotifyAll \mathrel{\widehat=} notifyAll \mathbin{;} serveLock
\end{aligned}
$$

Z Specification – Properties 1

- Absence from permanent waiting (*dormancy*)

$$\forall t : \operatorname{dom} waiting \bullet \Diamond (t \in \operatorname{dom} candidates)$$

- Absence from permanent blocking

$$\forall (t, o) : blocked \bullet \Diamond ((o, t) \in locked)$$

- Absence from permanent locking

$$\forall (o, t) : locked \bullet \Diamond ((o, t) \notin locked)$$

Z Specification – Properties 2

- Absence from deadlock

$$DeadlockCycle == \mathrm{dom}((blocked \mathbin{;} locked)^+ \cap id_{THREAD})$$

$$DeadlockCycle = \varnothing$$

- Absence from starvation

$$Deadlocked == \mathrm{dom}((blocked \mathbin{;} locked) \rhd DeadlockCycle)$$

$$\forall\, (t, o) : blocked \setminus Deadlocked \bullet \Diamond\, ((o, t) \in locked)$$

Discussion

- Properties can serve as basis for software verification tools
  - static analysis
  - model checking
  - dynamic analysis
- Example: deadlock detection
  - manage sets of threads as specified in Z model
  - can be implemented using standard Java collection classes
  - check properties in JVM

Summary

- Formal Z model for Java concurrency
  - state schema and operations
  - safety and liveness properties
- Can serve as a basis for software verification tools