A CSP Model for Java Multithreading

Paper by Peter H. Welch and Jeremy M.R. Martin
Why Java needs a CSP model

• Many Java programs using threads seem to work correctly, but
  • Data races might be hidden
  • Deadlocks might not yet be detected

• How can we be sure that data races and deadlocks in fact cannot occur?
Why Java needs a CSP model

• Solution: Model Java’s synchronization facility in CSP
  • Allows us to reason in a formal way about its correct usage in Java programs

• Synchronization primitives of Java’s client interface that are modelled:
  • synchronized
  • wait(), notify(), notifyAll()
Java Monitors in CSP

- We model *synchronized* by the following CSP processes \(^1\):
  - \texttt{STARTSYNC}( o, me) = claim.o.me \rightarrow \text{SKIP}
  - \texttt{ENDSYNC}( o, me) = release.o.me \rightarrow \text{SKIP}
Java Monitors in CSP

- \textit{wait()}, \textit{notify()}, \textit{notifyAll()} are modelled by the following CSP processes\textsuperscript{[1]}:
  - \texttt{WAIT}(o, me) = \texttt{waita.o.me} \rightarrow \texttt{release.o.me} \rightarrow \\
    \texttt{waitb.o.me} \rightarrow \texttt{claim.o.me} \rightarrow \texttt{SKIP}
  - \texttt{NOTIFY}(o, me) = \texttt{notify.o.me} \rightarrow \texttt{SKIP}
  - \texttt{NOTIFYALL}(o, me) = \texttt{notifyAll.o.me} \rightarrow \texttt{SKIP}
Java Monitors in CSP

- Java’s internal monitor facility for an object o is modelled by [1]:
  
  - $\text{MONITOR}(o) = \text{MLOCK}(o) \lor \text{MWAIT}(o, \{\})$
  
- $\text{MLOCK}(o) = \text{claim}.o.t \rightarrow \text{MLOCKED}(o, t)$
- $\text{MLOCKED}(o, t) = \text{release}.o.t \rightarrow \text{MLOCK}(o)$
  - $\text{notify}.o.t \rightarrow \text{MLOCKED}(o, t)$
  - $\text{notifyall}.o.t \rightarrow \text{MLOCKED}(o, t)$
  - $\text{waita}.o.t \rightarrow \text{MLOCKED}(o, t)$
Java Monitors in CSP

- **MWAIT** (using RELEASE) is defined as follows [1]:
  - \[ MWAIT(o, ws) = (waita.o?t \rightarrow MWAIT(o, ws \cup \{t\})) \]
  - \[ \square (notify.o?t \rightarrow \text{if } (|ws| > 0) \text{ then} \]
  - \[ \bigcap s \in ws \quad \text{waitb.o!s} \rightarrow MWAIT(o, ws - \{s\}) \]
  - \[ \text{else} \]
  - \[ MWAIT(o, \{\}) \]
  - \[ \square (notifyall.o?t \rightarrow RELEASE(o, ws)) \]

- **RELEASE** (o, ws) = \[\text{if } (|ws| > 0) \text{ then} \]
  - \[ \bigcap t \in ws \quad \text{waitb.o!t} \rightarrow RELEASE(o, ws - \{t\}) \]
  - \[ \text{else} \]
  - \[ MWAIT(o, \{\}) \]
Case Study

CSP model of One2OneChannel

• So far we have set up our CSP model of Java’s synchronization facility

• Let’s apply our CSP model to an example: The One2OneChannel of JCSP
Case Study
CSP model of One2OneChannel

- Allows exactly two threads to communicate with each other
- Communication complies with rendez-vous pattern
  - Reading thread and writing thread meet at some point in time

- Internal attributes:
  - **Object channel_hold**: Data being transmitted via the channel
  - **boolean channel_empty**: Indicates whether channel is empty

- Methods:
  - **public Object synchronized read()**
  - **public synchronized void write(Object mess)**
Case Study
CSP model of One2OneChannel

• Variables (restricted to boolean values) are managed by the VARIABLE process as follows \[1\]:
  • \text{VARIABLE}(o, v) = \text{VAR2}(o, v, \text{TRUE})
  • \text{VAR2}(o, v, d) = \left( \begin{array}{c}
  \text{getvar}.o.v.t!d \rightarrow \text{VAR2}(o, v, d) \\
  t \in \text{Threads}
  \end{array} \right)
  \begin{align*}
  & \text{setvar}.o.v.t?x \rightarrow \text{VAR2}(o, v, x) \\
  & t \in \text{Threads}
  \end{align*}

• \text{VARIABLES}(o) = \text{VARIABLE}(o, \text{channel_empty})
  \begin{align*}
  & | | \text{VARIABLE}(o, \text{channel_hold})
  \end{align*}
Case Study
CSP model of One2OneChannel

Java Code

```java
public synchronized Object read()
throws InterruptedException{
    if (channel_empty) {
        channel_empty = false;
        wait();
        wait();
        notify();
    } else {
        channel_empty = true;
        notify();
    }

    return channel_hold;
}
```

CSP Model

Java code and CSP model as in [1]
Case Study
CSP model of One2OneChannel

Java Code

```java
public synchronized Object read() throws InterruptedException{
    if (channel_empty) {
        channel_empty = false; wait();
        notify();
    } else {
        channel_empty = true; notify();
    }

    return channel_hold;
}
```

CSP Model

```
READ( o, t) =
    ready.o.t \rightarrow claim.o.t \rightarrow release.o.t \rightarrow
    READ( o, t)
```

Java code and CSP model as in [1]
Case Study

CSP model of One2OneChannel

Java Code

```java
public synchronized Object read() throws InterruptedException {
    if (channel_empty) {
        channel_empty = false;
        wait();
        notify();
    } else {
        channel_empty = true;
        notify();
    }

    return channel_hold;
}
```

CSP Model

```csp
READ( o, t) =
  ready.o.t ➔ claim.o.t ➔
  getvar.o.channel_empty.t?c ➔ (if (c = TRUE) then
    else
  ); ➔ release.o.t ➔ READ( o, t)
```

Java code and CSP model as in [1]
Case Study
CSP model of One2OneChannel

Java Code

```java
public synchronized Object read() throws InterruptedException{
    if (channel_empty) {
        channel_empty = false;
        wait();
        notify();
    } else {
        channel_empty = true;
        notify();
    }

    return channel_hold;
}
```

CSP Model

```
READ( o, t) =
ready.o.t ➔ claim.o.t ➔
getvar.o.channel_empty.t?c ➔ ( if (c = TRUE) then
    setvar.o.channel_empty.t!FALSE
else
    setvar.o.channel_empty.t!TRUE
);
getvar.o.channel_hold.t?mess ➔
release.o.t ➔ read.o.t!mess ➔
READ( o, t)
```

Java code and CSP model as in [1]
Case Study

CSP model of One2OneChannel

Java Code

```java
public synchronized Object read() throws InterruptedException{
    if (channel_empty) {
        channel_empty = false;
        wait();
        notify();
    } else {
        channel_empty = true;
        notify();
    }

    return channel_hold;
}
```

CSP Model

```csp
READ( o, t) =
    ready.o.t → claim.o.t →
    getvar.o.channel_empty.t?c → ( if (c = TRUE) then
        setvar.o.channel_empty.t!FALSE →
        WAIT( o, t); NOTIFY( o, t)
    else
        setvar.o.channel_empty.t!TRUE →
        NOTIFY( o, t)
    ); getvar.o.channel_hold.t?mess →
    release.o.t → read.o.t!mess →
    READ( o, t)
```

Java code and CSP model as in [1]
Case Study
CSP model of One2OneChannel

Java Code

```java
public synchronized Object read()
    throws InterruptedException{
  if (channel_empty) {
    channel_empty = false; wait(); notify();
  } else {
    channel_empty = true; notify();
  }

  return channel_hold;
}
```

CSP Model

```csp
READ( o, t) =
  ready.o.t ➔ claim.o.t ➔
  getvar.o.channel_empty.t?c ➔ (if (c = TRUE) then
    setvar.o.channel_empty.t!FALSE ➔
    WAIT( o, t); NOTIFY( o, t)
  else
    setvar.o.channel_empty.t!TRUE ➔
    NOTIFY( o, t)
  ); getvar.o.channel_hold.t?mess ➔
  release.o.t ➔ read.o.t!mess ➔
  READ( o, t)
```

Java code and CSP model as in [1]
Case Study

CSP model of One2OneChannel

- CSP model of write method similar to that of read method

- The final JCSP One2OneChannel is \[1\]:
  - \(\text{JCSPCHANNEL}(o, t_1, t_2) = \text{READ}(o, t_1) \parallel \text{WRITE}(o, t_2) \parallel \text{MONITOR}(o) \parallel \text{VARIABLES}(o)\)

- It is still possible that a thread, for which the channel is not destined, tries to access the channel
Case Study
Simplified Model of One2OneChannel

• The previous model of the One2OneChannel is simplified such that monitors are no longer needed

• The simplified model represents the One2OneChannel by two processes
  • **LEFT** process: Models ending to which input is written
  • **RIGHT** process: Models ending from which output is read
Case Study
Simplified Model of One2OneChannel

• The simplified channel is defined as follows [1]:
  • $\text{CHANNEL}(o, t_1, t_2) = (\text{LEFT}(o, t_2) \ | \ | \text{RIGHT}(o, t_1))$
    \{transmit.o.m | m \in \text{Data}\}

  • $\text{LEFT}(o, t_1) = \text{write.o.t}_1?\text{mess} \rightarrow \text{transmit.o!\text{mess} \rightarrow} \newline \text{ack.o.t}_1 \rightarrow \text{LEFT}(o, t_1)$

  • $\text{RIGHT}(o, t_2) = \text{ready.o.t}_2 \rightarrow \text{transmit.o?\text{mess} \rightarrow} \newline \text{read.o.t}_2!\text{mess} \rightarrow \text{RIGHT}(o, t_2)$
Case Study
Simplified Model of One2OneChannel

• Equivalence of the simplified model and the original model was verified using FDR

• From now on, we can therefore rely on the simplified model instead of the more complex model of the One2OneChannel
  • Makes reasoning about it easier
Conclusion

- CSP model of Java’s synchronization facility can be incorporated to build CSP models of Java programs.
- Reasoning about absence of deadlocks as well as data races possible using the CSP model.
- Verifying equivalence of complex to simplified models possible.
References

Questions?