A CSP Model for Java Multithreading

Paper by Peter H. Welch and Jeremy M.R. Martin

Presented by Tim Grabowski **16.05.2012**

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]:
  - **`STARTSYNC`**(o, me) = claim.o.me → **`SKIP`**
  - **`ENDSYNC`**(o, me) = release.o.me → **`SKIP`**

Java Monitors in CSP

- **`wait()`**, **`notify()`**, **`notifyAll()`** are modelled by the following CSP processes [1]:
  - **WAIT**(o, me) = waita.o.me ➔ release.o.me ➔ waitb.o.me ➔ claim.o.me ➔ **SKIP**
  - **NOTIFY**(o, me) = notify.o.me ➔ **SKIP**
  - **NOTIFYALL**(o, me) = notifyAll.o.me ➔ **SKIP**

Java Monitors in CSP

- Java's internal monitor facility for an object o is modelled by [1]:
  - **MONITOR**(o) = **MLOCK**(o) || **MWAIT**(o, {})

  - **MLOCK**(o) = claim.o?t ➔ **MLOCKED**(o, t)
  - **MLOCKED**(o, t) = release.o.t ➔ **MLOCK**(o)

    □ notify.o.t ➔ **MLOCKED**(o, t)

    □ notifyall.o.t ➔ **MLOCKED**(o, t)

    □ waita.o.t ➔ **MLOCKED**(o, t)

Java Monitors in CSP

- **MWAIT** (using **RELEASE**) is defined as follows [1]:
  - **MWAIT**(o, ws) = (waita.o?t ➔ **MWAIT**(o, ws ∪ {t}))

    □ (notify.o?t ➔ **if** (|ws| > 0) **then**

    $\bigsqcap_{s \in ws}$ waitb.o!s ➔ **MWAIT**(o, ws – {s})

    else

    MWAIT(o, {}))

    □ (notifyall.o?t ➔ **RELEASE**(o, ws))

  - **RELEASE**(o, ws) = **if** (|ws| > 0) **then**

    $\bigsqcap_{t \in ws}$ waitb.o!t ➔ **RELEASE**(o, ws – {t})

    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]:
  - **VARIABLE**(o, v) = **VAR2**(o, v, TRUE)
  - **VAR2**(o, v, d) = (□ getvar.o.v.t!d ➔ **VAR2**(o, v, d))

    t ∈ Threads

    □ (□ setvar.o.v.t?x ➔ **VAR2**(o, v, x))

    t ∈ Threads
  - **VARIABLES**(o) = **VARIABLE**(o, channel_empty)

    || **VARIABLE**(o, channel_hold)

Case Study

CSP model of One2OneChannel

Java Code

```
public synchronized Object read()
 throws InterruptedException{
  if (channel_empty) {
   channel_empty = false;
   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

```
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 → release.o.t →
    READ( o, t)
```

Java code and CSP model as in [1]

Case Study

CSP model of One2OneChannel

Java Code

```
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
    else
    ); ➔ release.o.t ➔ READ( o, t)
```

Java code and CSP model as in [1]

Case Study

CSP model of One2OneChannel

Java Code

```
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

```
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 ➔
        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

```
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 →
        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]:
  - **JCSPCHANNEL**(o, t_1, t_2) = **READ**(o, t_1) || **WRITE**(o, t_2) || **MONITOR**(o) || **VARIABLES**(o)
- It is still possible that a thread, for which the channel is not destinated, 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]:
  - **CHANNEL**(o, t_1, t_2) = (**LEFT**(o, t_2) || **RIGHT**(o, t_1)) \ {transmit.o.m | m ∈ Data}
  - **LEFT**(o, t_1) = write.o.t_1?mess ➔ transmit.o!mess ➔ ack.o.t_1 ➔ **LEFT**(o, t_1)
  - **RIGHT**(o, t_2) = ready.o.t_2 ➔ transmit.o?mess ➔ read.o.t_2!mess ➔ **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

[1] Peter H. Welch and Jeremy M. R. Martin. 2000. A CSP Model for Java Multithreading. *International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE)*. 114-122.

Questions?