
Lecture 2: Conc at Progr Lang Level

Concurrency at Programming Language Level

Ch 2 [BenA 06]

Abstraction
Pseudo-language
BACI
Ada, Java, etc.

Levels of Abstraction

- Granularity of operations
  - Invoke a library module
  - Statement in high level programming language
  - Instruction in machine language

- Atomic statement
  - Anything that we can guarantee to be atomic
    - Executed completely “at once”
    - Always the same correct atomic result
    - Result does not depend on anybody else
    - Process switches may occur, but they do not affect result
  - Can be at any granularity
  - Can trust on that atomicity

Atomic Statement

- Atomicity guaranteed somehow
  - Machine instruction: HW
  - Memory bus transaction
  - Programming language statement, set of statements, or set of machine instructions
    - SW
      - Manually coded
      - Disable interrupts
      - OS synchronization primitives
    - Library module
      - SW
        - Manually coded inside
        - Provided automatically to the user by programming environment

Program State, Pseudo-language

- Sequential program

  Algorithm 2.1: Trivial sequential program

  \[
  \begin{array}{l}
  \text{integer } n = 0 \\
  \text{while } n < k \\
  \text{int } k1 = 1 \\
  \text{int } k2 = 2 \\
  \text{int } n = k1 \\
  \text{int } k = n - k2 \\
  \end{array}
  \]

  State

  - next statement to execute (cp, i.e., PC)
  - variable values

  p1: n = k1

  p2: n = k2

  (end) k1 = 1, k2 = 2

  Atomic statement

  Global state

  - All cp’s
  - All local variables
  - All global variables

Concurrent Program

- Sequential process
  - Successive atomic statements
    - Control pointer
      - (= program counter)
    - Concurrent program
      - Finite set of sequential processes working for same goal
      - Arbitrary interleaving of atomic statements in different processes

3 processes (P, R, Q) interleaved execution

(P, R, Q)

Global Program State

- Concurrent program

  Algorithm 2.1: Trivial concurrent program

  \[
  \begin{array}{l}
  \text{integer } n = 0 \\
  \text{while } n < k \\
  \text{int } k1 = 1 \\
  \text{int } k2 = 2 \\
  \end{array}
  \]

  State

  - Local state for each process
    - cp
    - Variable values
      - Local & global
    - Global state
      - for program
        - All cp’s
        - All local variables
        - All global variables

Monitors

Ch 7 [BenA 06]

Java synchronized methods

Copyright Teemu Kerola 2012
Possible Program States

- List of processes in program
  - List of values for each process
    - cp
    - value of each local/global/shared variable

- Nr of possible states can be (very) large
  - Not all states are reachable states!
  - Different executions do not go through same states (even with same input)

State Diagram and Scenarios

- Transitions from one possible state to another
  - Executed statement must be one of those in the 1st state
  - Contains all reachable states and transitions
  - All possible executions are included, they are all correct!

Algorithm 2.1: Trivial concurrent program

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer k1 = 1</td>
<td>integer k2 = 2</td>
</tr>
<tr>
<td>{ p1: n=k1 }</td>
<td>{ q1: n=k2 }</td>
</tr>
<tr>
<td>n = 0</td>
<td>n = 0</td>
</tr>
</tbody>
</table>

Atomic Statements

- Two scenarios
  - Both correct
  - Different result!

NO need to have the same result!

Statements do the same, but overall result may be different. (see p.19 [BenA 06])

- Atomic?
  - Assignment?
  - Boolean evaluation?
  - Increment?

Algorithm 2.3: Atomic assignment statements

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer n = 0</td>
<td>integer n = 0</td>
</tr>
<tr>
<td>p1: n = n + 1</td>
<td>q1: n = n + 1</td>
</tr>
</tbody>
</table>

Two scenarios for execution
- Both correct
- Both have the same result

P first, and then Q

Q first, and then P

Same statements with smaller atomic granularity:

Algorithm 2.4: Assignment statements with one global reference

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer temp</td>
<td>integer temp</td>
</tr>
<tr>
<td>p1: temp = n</td>
<td>q1: temp = n</td>
</tr>
<tr>
<td>p2: n = temp + 1</td>
<td></td>
</tr>
</tbody>
</table>

Scenario 1
- OK

Scenario 2
- Bad result

From now on
- Assignments and Boolean evaluations are atomic!
Correctness

• What is the correct answer?
• Usually clear for sequential programs
• Can be fuzzy for concurrent programs
  – Many correct answers?
  – What is the intended semantics for the program?
  – Run program 100 times, each time get different answer?
  • Each answer is correct, if program is correct!
  • Can not test all possible scenarios (too many!)
• How to define correctness for concurrent programs?
  – Safety properties = properties that are always true
  – Liveness properties = properties that eventually become true

Safety and Liveness

• Safety property
  – property must be true all the time (‘bad’ never happens)
  – “Identity”
  – memFree + memAllocated = memTotal
  – Mouse cursor is always displayed
  – System responds always to new commands
• Liveness property
  – Property must eventually become true (‘good’ eventually happens)
  – Variable n value = 2
  – System prompt for next command is shown
  – Control will resume to calling program
  – Eventually the mouse cursor is displayed
  – Program will terminate

Duality of safety and liveness properties

– { \( P_i \) is always not eating } 
  – { \( \neg P_i \) will get his turn to eat }
– not \{ n value is always \( \not\equiv 2 \) \}
  – { n value will become 2 }

Linear Temporal Logic (LTL)

• Define safety and liveness properties for certain state in some (arbitrary) scenario
  – Example of Modal Temporal Logic (MDL), logic on concepts like possibility, impossibility, and necessity
• Alternative: Branching Temporal Logic (BTL)
  – Properties true in some or all states starting from the given state
  – More complex, because all future states must be covered
  – Common Temporal Logic (CTL)
  • Can be checked automatically
  – Every time computation reaches given state
  – SMV model checker
  – NuSMV model checker

Fairness

• (Weakly) fair scenario
  – Wanted condition eventually occurs
  – Nobody is locked out forever?
  – Will a philosopher ever get his turn to eat?
  – Will an algorithm eventually stop?
  – p and q are both scheduled to run eventually

– Algorithm 2.5: Stop the loop A

\[
\begin{array}{c|c|c|c|c}
 & p & q & \text{flag} & \text{flag} = \text{true} \\
\hline
\mu & 1 & 0 & 0 & 0 \\
\nu & n & 1 & n & 0 \\
\xi & n & 1 & n & 1 \\
\phi & n & 1 & n & 1 \\
\end{array}
\]

• All scenarios should be fair
  – One requirement in correct solution

Machine Language Code

• What is atomic and what is not?
  – Assignment? \( X = Y \)
  – Increment? \( X = X + 1 \)

Critical Reference

• Reference to (shared) variable v is critical reference, if...
  – Assigned value in (process) P and read in Q
  – Program satisfies limited-critical-reference (LCR)
  – Each statement has at most one critical reference
  – Easier to analyze with LCR than without LCR
  – Each program is easy to transform into similar program with LCR

– LCR: Bad
  – LCR vs. atomicity? (ouch)
Volatile and non-atomic variables

- Volatile variable
  - Can be modified by many processes (must be in shared memory)
  - Advice for compiler (pragma)
    - Keep something in memory, not in register
  - Pseudocode—does not generate code

- Non-atomic (multitword) data always volatile?
  - Multitword data structures: long ints, arrays, records, ...
  - Force access to be indivisible (atomic) in given order

Example Program with Volatile Variables

```java
public static void main(String[] args) {
    int n = 0;
    for (int i = 0; i < 10; i++) {
        n = temp + 1;
    }
    System.out.println("The value of \( n \) is \( ^{*} \) \( n \", n);
}
```

Discuss

```java
> javac Adder8.java
> java Adder8
```

Execute on 8-processor vera.cs.helsinki.fi?

http://www.cs.helsinki.fi/u/kerola/rio/Java/examples/Adder8b.java
Run Multi-threaded Java

Execute on 8-processor vera.cs.helsinki.fi?

```
kerola@vera:~/public_html/rio/Java/examples$ javac Adder8.java
kero@vera:~/public_html/rio/Java/examples$ java Adder8
```

```
finally n = 80000 = 37358
kerola@vera:~/public_html/rio/Java/examples$ java Adder8
```

```
finally n = 80000 = 34464
```

Run them yourself?
(Copy source code in your own directory)

Why different result?
What is correct result?

BACI Overall Structure

```
add.pco
....
17 24 void main() {
18 25cobegin {add10(); add10();}
```

```
add.cm
```

C -- (Concurrent C)


(BACI, Ben-Ari Concurrency Interpreter)

Ben-Ari Concurrency Interpreter

– Write concurrent programs with

    C-- or Ben-Ari Concurrent Pascal (.cm and .pm suffixes)

– Compile and run in BACI

– GUI for Unix/Linux

jBACI

Just like BACI

– GUI for Windows

Installation

– load version 1.4.5 jBACI executable files and example programs, unzip, edit config.cfg to have correct path to bin\bacc.exe compiler, click jbaci.jar

• Use in class and with homework

jBACI IDE (integrated development environment)

Add a breakpoint to selected (PCode) line

jBACI IDE (integrated development environment)
Summary

- Abstraction, atomicity
- Concurrent program, program state
- Pseudo-language algorithms
- High level language algorithms
- BACI