Concurrent Programming (RIO) 16.10.2009

Lecture 2 summary: 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
    • 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

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

Program State, Pseudo-language
• Sequential program

\[
\text{Algorithm 2.2: Total sequential program}
\]

\[
\begin{align*}
\text{integer } & k_1, k_2, n = 0 \\
\text{do } & \\
\text{while } & (k_1, k_2, n) = (0, 2, 2) \\
\text{at } & \text{state 1} \\
\text{variable values } & \\
& k_1 = 1, \quad k_2 = 2, \quad n = 2 \\
\text{atomic statement } & \\
& \text{do } \\
\text{while } & (k_1, k_2, n) = (0, 2, 2) \\
\text{at } & \text{state 1} \\
\end{align*}
\]

• State
  – next statement to execute (cp, i.e., PC)
  – variable values

\[
\begin{align*}
\text{Algorithm 2.3: Total concurrent program}
\end{align*}
\]

\[
\begin{align*}
\text{integer } & k_1, k_2, n = 0 \\
\text{do } & \\
\text{while } & (k_1, k_2, n) = (0, 2, 2) \\
\text{at } & \text{state 1} \\
\text{variable values } & \\
& k_1 = 1, \quad k_2 = 2, \quad n = 2 \\
\text{atomic statement } & \\
& \text{do } \\
\text{while } & (k_1, k_2, n) = (0, 2, 2) \\
\text{at } & \text{state 1} \\
\end{align*}
\]

...
Possible Program States

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

\[
\begin{align*}
\{ p_1: n \leftarrow k_1 \mid k_1 = 1 \} \\
\{ q_1: n \leftarrow k_2 \mid k_2 = 2 \},
\end{align*}
\]

- At any instant of time:
- Nr of possible states can be (very) large
- Some states are unreachable (not all states are reachable states!

State Diagram and Scenarios

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

Algorithm 2.3: Atomic assignment statements

\[
\begin{align*}
p_1: & \quad n \leftarrow n + 1 \\
q_1: & \quad n \leftarrow n + 1
\end{align*}
\]

Atomic Statements

- Two scenarios
  - Both correct
  - Different result!

Algorithm 2.4: Assignment statements with one global reference

\[
\begin{align*}
p_1: & \quad \text{temp} \leftarrow 1 \\
q_1: & \quad n \leftarrow \text{temp} + 1
\end{align*}
\]

Scenario 2
- 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 intended semantics of the program?
  - Run programs 100 times, each time get different answer?
    - Each answer is correct, if program is correct!
    - Usually 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
  - "Identity"
  - \( \text{memFree} + \text{memAllocated} = \text{memTotal} \)
  - Mouse cursor is displayed
  - System responds to new commands
- Liveness property
  - Property must eventually become true
  - \( \text{P} \) will get his turn to eat
  - \( \text{P} \) will never get his turn to eat
  - \( \text{P} \) will terminate

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
    - 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?

Critical Reference

- Reference to variable \( v \) is critical reference, if…
  - Assigned value in \( P \) and read in \( Q \)
  - Read directly or in a statement
- Program satisfies limited-critical-reference (LCR)
  - Each statement has at most one critical reference
  - Easier to analyze than without this property
  - Each program is easy to transform into similar program with LCR
Lecture 2 summary: Concurrency at Programming Language Level

- **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 variables**
    - Multitword data structures: long ints, arrays, records, ...
    - Force access to be indivisible in given order

- **Algorithm 2.6: Volatile variables**
  ```
  procedure p;
  var temp1: integer;
  begin
    for i := 1 to 10 do
      temp1 := temp1 + i;
  end
  
  procedure q;
  var temp2: integer;
  begin
    for j := 1 to 10 do
      temp2 := temp2 + j;
  end
  
  begin
    p;
    q;
  end.
  ```

- **Example Program with Volatile Variables**
  - Can implement it in any concurrent programming language
    - (Extended) Pascal and (Extended) C
    - BACI (Ben-Ari Concurrency Interpreter)
    - Code automatically compiled (from Extended Pascal or C)
    - Ada
    - Java

- **Concurrent Program in Java**
  ```
  class Count extends Thread {
      volatile int n = 0;
      Count p = new Count();
      Count q = new Count();
      Count r = new Count();
      
      public void main(String[] args) {
          System.out.println(\"The value of n is \" + n);
      }
      
      public static void main(String[] args) {
          Count p = new Count();
          Count q = new Count();
          Count r = new Count();
          
          p.start();
          q.start();
          r.start();
          
          Thread.yield();
      }
      
      public void run() {
          while (true) {
              if (n < 10) {
                  n = n + 1;
              } else {
                  break;
              }
      }
  }
  ```

- **Possibly volatile, use carefully**
  - (Ben-Ari Concurrent Pascal)
  - (Ben-Ari Concurrent C, C++)
  - (volatile, if critically referenced)

- **Concurrent Program in Ada**
  ```
  with Ada.Text_IO; use Ada.Text_IO;
  
  procedure Count is
      N : integer := 0;
      
      task type Count_Task is
      task body Count_Task is
          Temp : integer;
          begin
            for i := 1 to 10 loop
                Temp := Temp + i;
            end loop;
            end Count_Task;
      end
  
  begin
      with Ada.Text_IO; use Ada.Text_IO;
      
      procedure Count is
          N : integer := 0;
          
          task type Count_Task is
          task body Count_Task is
              Temp : integer;
              begin
                for i := 1 to 10 loop
                    Temp := Temp + i;
                end loop;
                end Count_Task;
          end
      
      begin
          with Ada.Text_IO; use Ada.Text_IO;
          
          procedure Count is
              N : integer := 0;
              
              task type Count_Task is
              task body Count_Task is
                  Temp : integer;
                  begin
                    for i := 1 to 10 loop
                        Temp := Temp + i;
                    end loop;
                    end Count_Task;
              end
          
          begin
              with Ada.Text_IO; use Ada.Text_IO;
              
              procedure Count is
                  N : integer := 0;
                  
                  task type Count_Task is
                  task body Count_Task is
                      Temp : integer;
                      begin
                        for i := 1 to 10 loop
                            Temp := Temp + i;
                        end loop;
                        end Count_Task;
                  end
              
              begin
                  with Ada.Text_IO; use Ada.Text_IO;
                  
                  procedure Count is
                      N : integer := 0;
                      
                      task type Count_Task is
                      task body Count_Task is
                          Temp : integer;
                          begin
                            for i := 1 to 10 loop
                                Temp := Temp + i;
                            end loop;
                            end Count_Task;
                      end
                  
                  begin
                      with Ada.Text_IO; use Ada.Text_IO;
                      
                      procedure Count is
                          N : integer := 0;
                          
                          task type Count_Task is
                          task body Count_Task is
                              Temp : integer;
                              begin
                                for i := 1 to 10 loop
                                    Temp := Temp + i;
                                end loop;
                                end Count_Task;
                          end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                              task type Count_Task is
                              task body Count_Task is
                                  Temp : integer;
                                  begin
                                    for i := 1 to 10 loop
                                        Temp := Temp + i;
                                    end loop;
                                    end Count_Task;
                              end
                      
                      begin
                          with Ada.Text_IO; use Ada.Text_IO;
                          
                          procedure Count is
                              N : integer := 0;
                              
                          ```
BACI

- 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 paths to bin/bacc.exe and bin/bapas.exe translators, click run.bat
  - Use in class, homeworks and in project

BACI Overall Structure

```
void main() {
cobegin {add10(); add10();}
```

jBACI

- Just like BACI, but with Java
  - requires Java v. 1.4 (SDK or JRE)
  - Built-in compiler and interpreter
  - edit state
  - run state

Summary

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