Sequential Programming

Dr. José Luis Zechinelli Martini

joseluis.zechinelli@udlap.mx

LDS-1101

Agenda

- Sequential programming
- Safety and liveness properties
- Logic, propositions, and predicates
- A programming logic
- Proofs in programming logic:
  - Linear search
  - Bubble sort
- Discussion

Sequential programming

- Concurrent programs extend sequential programs with mechanisms for specifying concurrency, communication, and synchronization

- A sequential program contains declarations, statements, and procedures:
  - **Declarations** defines types, variables, and constants
  - **Statements** are used to assign values to variables and to control the flow of execution within the program
  - **Procedures** define parameterized subroutines and functions

Safety and liveness properties

- A property of a program is an attribute that is true of every possible history of that program, and hence of all executions of the program
- Every property can be formulated in terms of two special kinds of properties:
  - **Safety property** asserts that the program never enters a bad state, *i.e.*, one in which some variables have undesirable values (partial correctness, mutual exclusion, absence of deadlock)
  - **Liveness property** asserts that a program eventually enters a good state, *i.e.*, one in which the variables all have desirable values (termination, eventual entry to a critical section)
- Total correctness is a property that combines partial correctness (safety) and termination (liveness)

Agenda

- ✓ Sequential programming
- ✓ Safety and liveness properties
- Logic, propositions, and predicates
- A programming logic
- Proofs in programming logic:
  - Linear search
  - Bubble sort
- Discussion

Formal logical systems

- A programming logic is a formal system that supports the assertional approach to developing and analyzing programs; it includes:
  - Predicates that characterize program states
  - Relations that characterize the effect of program execution
- Any formal logical system consists of rules defined in terms of:
  - A set of symbols
  - A set of formulas constructed from these symbols
  - A set of distinguished formulas called axioms
  - A set of inference rules

Propositions

- Propositional logic is an instance of a formal logical system that formalizes what we usually call "common sense" reasoning:
  - The formulas of the logic are called propositions; these are statements that are either true or false
  - The axioms are special propositions that are assumed to be true
  - The inference rules allow new, true propositions to be formed from existing ones
- In a propositional logic the propositional symbols are:
  - Propositional constants: true and false
  - Propositional variables: p, q, r, …
  - Propositional operators: ¬, ^, v, and =

Predicates

- A predicate logic extends a propositional logic to manipulate any kind of boolean-valued expression

- The two extensions are as follows:
  - Any expression such as $x < y$ that maps to true or false can be used in place of a propositional variable
  - Existential ($\exists$) and universal ($\forall$) quantifiers are provided to characterize sets of values

Agenda

- ✓ Sequential programming
- ✓ Safety and liveness properties
- ✓ Logic, propositions, and predicates
- A programming logic
- Proofs in programming logic:
  - Linear search
  - Bubble sort
- Discussion

A programming logic (1)

- A programming logic (PL) is a formal logical system that facilitates making precise statements about program execution
- PL contains symbols, formulas, axioms, and inference rules:
  - The symbols of PL are predicates, braces, and programming language statements
  - The formulas of PL are triples of the form: {P} S {Q}, where P and Q are predicates and S is a simple or compound statement
- The interpretation of triple {P} S {Q} is true if, whenever execution of S is begun in a state satisfying P and execution of S terminates, the resulting state satisfies Q

A programming logic (2)

- The axioms are special formulas that are a priori assumed to be true
- Inference rules specify how to derive additional true formulas from axioms and other true formulas
  - By itself, a formal logical system is a mathematical abstraction: a collection of symbols and relations between them
  - A logical system becomes interesting when the formulas represent statements about some domain of discourse and the formulas that are theorems are true statements

Agenda

- ✓ Sequential programming
- ✓ Safety and liveness properties
- ✓ Logic, propositions, and predicates
- ✓ A programming logic
- Proofs in programming logic:
  - Linear search
  - Bubble sort
- Discussion

Proofs in programming logic

- A proof is a sequence of lines:
  - Each line is an instance of an axiom or follows from previous lines by application of an inference rule
  - A theorem is any line in a proof; thus theorems are either axioms or are obtained by applying an inference rule to other theorems

- A proof outline provides a compact way in which to present the outline of a proof:
  - It consists of the statements of a program interspersed with assertions
  - A complete proof outline contains at least one assertion before and after each statement

Complete proof outline example

```
LINEAR_SEARCH(A : LIST of Integer, x : Integer)
    { P: length(A) > 0 ^ ( ∃ j: 1 ≤ j ≤ length(A): A[ j ] = x ) }
    i ← 1
    { P ^ i = 1 }
    { I: P ^ ( ∀ j: 1 ≤ j ≤ i: A[ j ] != x ) }
    while  A[ i ] != x  do
        { I ^ A[ i ] != x }
        i ← i + 1
        { I }
    end
    { I ^ A[ i ] = x }
    { LS: x = A[ i ] ^ ( ∀ j: 1 ≤ j ≤ i: A[ j ] != x ) }
end
```

Bubble sort example

```
BUBBLE_SORT(A : LIST of Integer)

    for  j ← 2  to  length(A)  do

        pivot ← A[ j ]

        % insert A[j] into ordered sequence A[ 1 .. j - 1 ]
        i ← j - 1
        while  i > 0  ^  A[ i ] > pivot  do
                A[ i + 1 ] ← A[ i ]
                i ← i - 1
        end
        A[ i + 1 ] ← pivot
    end

end
```

Example: Desktop proof (1)

Let A = [4, 3, 2, 1]:

- First iteration : j = 2
  - pivot = A[j] = 3
  - i = j – 1 = 1
  - It is satisfied that i> 0 and A [i]> pivot
    - Run once: A[i + 1] ← A[i] y i ← i - 1
    - A = [4, 4, 2, 1], i = 0

  Run A[i + 1] ← pivot ➔ leaving A = [3, 4, 2, 1]

- Second iteration : j = 3
  - pivot = A[j] = 2
  - i = j – 1 = 2
  - It is satisfied that i> 0 and A [i]> pivot
    - Run twice : A[i + 1] ← A[i] y i ← i - 1
    - A = [3, 4, 4, 1], i = 1
    - A = [3, 3, 4, 1], i = 0
  - Run A[i + 1] ← pivot ➔ leaving A = [2, 3, 4, 1]

Example: Desktop proof (2)

A = [2, 3, 4, 1]:

- Third iteration : j = 4
  - pivot = A[j] = 1
  - i = j – 1 = 3
  - It is satisfied that i> 0 and A [i]> pivot
    - Run three times: A[i + 1] ← A[i] y i ← i - 1
    - A = [2, 3, 4, 4], i = 2
    - A = [2, 3, 3, 4], i = 1
    - A = [2, 2, 3, 4], i = 0
  - Run A[i + 1] ← pivot ➔ leaving A = [1, 2, 3, 4]

Agenda

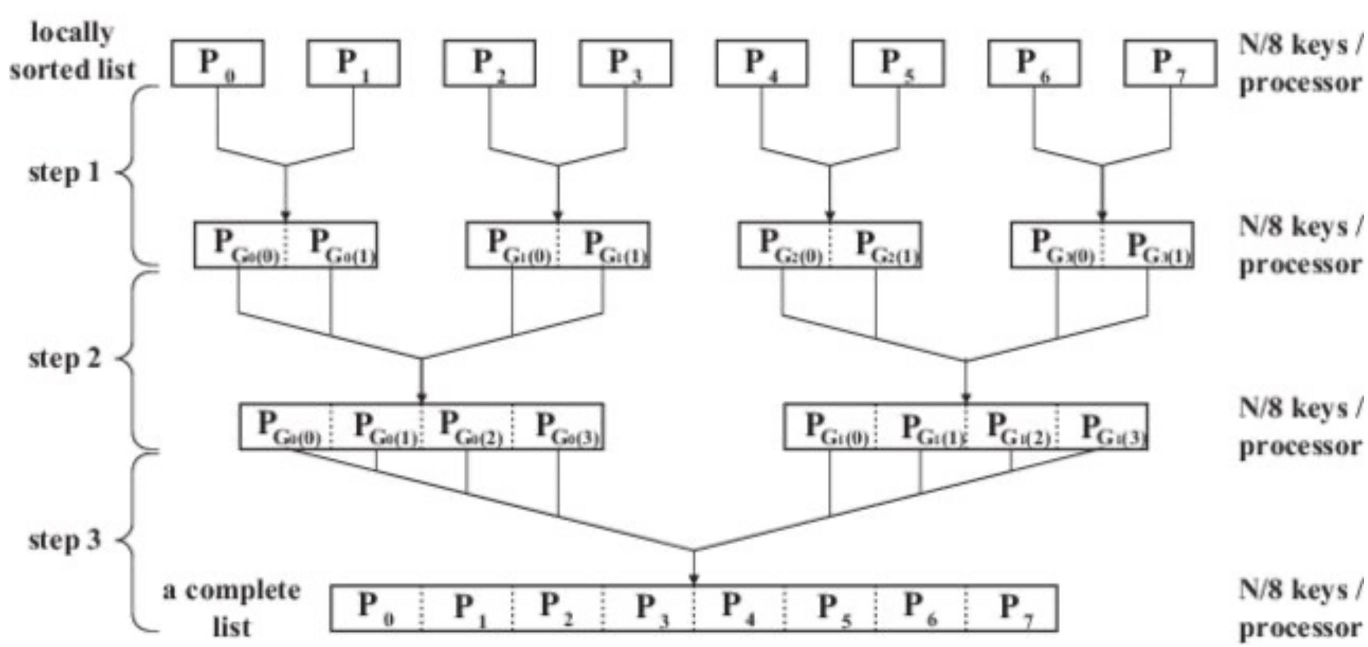
- ✓ Sequential programming
- ✓ Safety and liveness properties
- ✓ Logic, propositions, and predicates
- ✓ A programming logic
- ✓ Proofs in programming logic:
  - ✓ Linear search
  - ✓ Bubble sort
- Discussion

Discussion

- Concurrent programs are inherently more complex than sequential programs
- A concurrent program specifies two or more processes that cooperate in performing a task:
  - Each process is a sequential program that executes a sequence of statements
  - Processes cooperate by communicating; they communicate using shared variables or message passing
- Our ultimate goal is to understand how to construct correct concurrent programs

Concurrent merge sort (1)

- Define the concurrent merge sort program CMS(V : LIST of integer):



Concurrent merge sort (2)

- Run CMS(V) program with V = [2, 6, 8, 2, 3, 9, 1, 4, 9]: