Concurrent Programming

Example

Example

```
int y = 0, z = 0;
```

Example

```
int y = 0, z = 0;

P1::  x = y + z;
```

Example

```
int y = 0, z = 0;

P1::  x = y + z;        P2::  y = 1;
                              z = 2;
```

Example

```
int y = 0, z = 0;

P1::  x = y + z;        P2::  y = 1;
                              z = 2;
```

- What do the final values of x, y, and z depend on?

Example

```
int y = 0, z = 0;

P1::  x = y + z;        P2::  y = 1;
                              z = 2;
```

- What do the final values of x, y, and z depend on?
  - Execution order

Example

```
int y = 0, z = 0;

P1::  x = y + z;        P2::  y = 1;
                              z = 2;
```

- What do the final values of x, y, and z depend on?
  - Execution order
  - What is atomic

States, Atomic Actions, History, Property

States, Atomic Actions, History, Property

- State: values of variables at a point in time

States, Atomic Actions, History, Property

- State: values of variables at a point in time
- Atomic action: indivisible state change
  - fine-grained (hardware): load, store, add, ...
  - coarse-grained (software): critical sections, process state encapsulation

States, Atomic Actions, History, Property

- State: values of variables at a point in time
- Atomic action: indivisible state change
  - fine-grained (hardware): load, store, add, ...
  - coarse-grained (software): critical sections, process state encapsulation
- History: a trace of ONE execution; an interleaving of atomic actions

States, Atomic Actions, History, Property

- State: values of variables at a point in time
- Atomic action: indivisible state change
  - fine-grained (hardware): load, store, add, ...
  - coarse-grained (software): critical sections, process state encapsulation
- History: a trace of ONE execution; an interleaving of atomic actions
- Property: an attribute of ALL histories of a program, e.g., correctness

Coarse-grained Atomicity

Coarse-grained Atomicity

- Atomicity may be required when fine-grained atomicity is insufficient

Coarse-grained Atomicity

- Atomicity may be required when fine-grained atomicity is insufficient
  - Need mechanism for constructing coarse-grained atomic actions

Coarse-grained Atomicity

- Atomicity may be required when fine-grained atomicity is insufficient
  - —Need mechanism for constructing coarse-grained atomic actions
    - Sequence of fine-grained atomic actions

Coarse-grained Atomicity

- Atomicity may be required when fine-grained atomicity is insufficient
  - Need mechanism for constructing coarse-grained atomic actions
    - Sequence of fine-grained atomic actions
    - Appearance of indivisibility

Specifying Atomic Actions

Specifying Atomic Actions

- < S; >
  - execute statement list S indivisibly (mutual exclusion)

Specifying Atomic Actions

- < S; >
  - execute statement list S indivisibly (mutual exclusion)
- < await(B); >
  - wait for B to be true (conditional synchronization)

Specifying Atomic Actions

- < S; >
  - execute statement list S indivisibly (mutual exclusion)
- < await(B); >
  - wait for B to be true (conditional synchronization)
- < await(B) S; >
  - wait for B to be true, then execute S ALL AS A SINGLE ATOMIC ACTION

Histories

Histories

- How many histories are there in a program with n processes and m atomic actions per process?

Histories

- How many histories are there in a program with n processes and m atomic actions per process?

```
(n * m)! / (m!)**n
```

Histories

- How many histories are there in a program with n processes and m atomic actions per process?

```
(n * m)! / (m!)**n
```

if n = 2 and m = 4, this is 70 histories!

Proving Properties

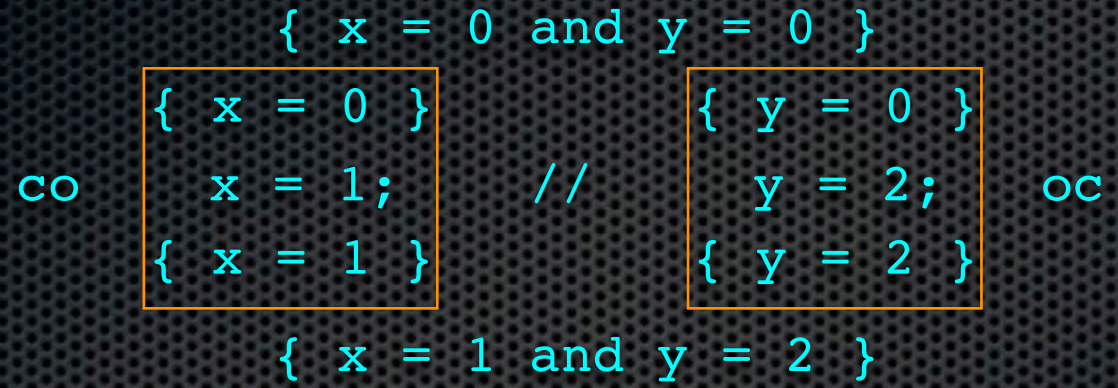
Proving Properties

```
co    x = 1;    //     y = 2;   oc
```

Proving Properties

```
co  { x = 0 }      //   { y = 0 }   oc
      x = 1;             y = 2;
    { x = 1 }          { y = 2 }
```

Proving Properties



- The precondition of the concurrent program is the conjunction (and) of the preconditions of the processes.

Proving Properties

```
                { x = 0 and y = 0 }
          { x = 0 }          { y = 0 }
     co     x = 1;     //      y = 2;     oc
          { x = 1 }          { y = 2 }
                { x = 1 and y = 2 }
```

- The precondition of the concurrent program is the conjunction (and) of the preconditions of the processes.
- If the processes are disjoint -- as here -- then the "proofs" don't interfere, so the postcondition is the conjunction (and) of the postconditions of the processes.

Proving Properties

Proving Properties

- Assertions
  - Predicates that are true at points in a program

Proving Properties

- Assertions
  - Predicates that are true at points in a program
- Interference
  - Statements in one process invalidating assertions in another

Proving Properties

- Assertions
  - Predicates that are true at points in a program
- Interference
  - Statements in one process invalidating assertions in another
  - Caused by shared variables

Ways to Avoid Interference

Ways to Avoid Interference

1) disjoint variables -- independent processes

Ways to Avoid Interference

1) disjoint variables -- independent processes
2) weakened assertions -- say less than you could in isolation

Ways to Avoid Interference

1) disjoint variables -- independent processes
2) weakened assertions -- say less than you could in isolation
3) synchronization -- hide states and/or delay execution

Ways to Avoid Interference

1) disjoint variables -- independent processes
2) weakened assertions -- say less than you could in isolation
3) synchronization -- hide states and/or delay execution
4) global invariants -- predicates that are true in ALL visible states

Synchronization

Synchronization

- Synchronization restricts the number of histories

Synchronization

- Synchronization restricts the number of histories
- Prevents undesirable interleavings by:

Synchronization

- Synchronization restricts the number of histories
- Prevents undesirable interleavings by:
  - Combining fine-grained atomic actions into coarse-grained atomic actions

Synchronization

- Synchronization restricts the number of histories
- Prevents undesirable interleavings by:
  - Combining fine-grained atomic actions into coarse-grained atomic actions
    - *Mutual exclusion*

Synchronization

- Synchronization restricts the number of histories
- Prevents undesirable interleavings by:
  - Combining fine-grained atomic actions into coarse-grained atomic actions
    - *Mutual exclusion*
  - Delaying process execution until some predicate is satisfied

Synchronization

- Synchronization restricts the number of histories
- Prevents undesirable interleavings by:
  - Combining fine-grained atomic actions into coarse-grained atomic actions
    - *Mutual exclusion*
  - Delaying process execution until some predicate is satisfied
    - *Condition Synchronization*