



T-106.420

Concurrent Programming

Formal semantics & correctness proofs



How to convince ourselves that a program satisfies a property p ?

- testing: show that p is not violated for any input data
 - testing proves the existence of bugs, but their absence
 - cannot test all cases in practice
- operational reasoning: try all possible histories for p
 - “exhaustive case analysis”, i.e. all possible histories enumerated
 - for n processes with m atomic actions: $(n-m)!/(m!^n)$ histories;
 $n=3, m=2 \Rightarrow 90$ different histories!
 - a (pathological) trace causing a serious system fault may appear only after months of successful operation



How to convince ourselves that a program satisfies a property p ?

- Construct *a proof* : p holds for any history of the program
 - a proper proof covers all histories for a key aspect in a compact, abstract form
 - requires formal reasoning, i.e. *a programming logic*
 - like the program, it's proof is also a formal object, which can be manipulated automatically \Rightarrow automatic proof checkers
 - proving large systems “completely” is still impractical, but proofs focusing on their non-trivial aspects, like concurrency, are becoming doable
 - construct the program together with its informal proof focusing on key aspects of *system consistency*
 - already an attempt to prove (i.e. understand precisely) a property often reveals a possible anomaly ignored thus far in the design.

***In this course correctness proofs are used only informally,
i.e. like mathematics, not as in logic.***



interesting properties

- true for every possible history
- *safety* property
 - the program never enters a *bad* state
 - generalisation of *partial correctness*:
 - if the program terminates, its final state provides the correct answer
- *liveness* property
 - the program reaches a desired state eventually
 - generalization of *termination*:
 - every loop & procedure call terminates, i.e. every history is finite
- *total correctness*
 - partial correctness + termination
 - the program always terminates and with a correct answer
 - classical property of sequential programs



Hoare triple $\{P\} S \{Q\}$

- $\{P\} S \{Q\}$ is a true theorem iff:
 - if the execution of program S begins in a state satisfying predicate P and if it terminates, the resulting state satisfies predicate Q
- *precondition* P and *postcondition* Q are called *assertions*, because they assert that the program state must satisfy the predicate in order for the interpretation of the triple to be true
- an *assertion* specifies an acceptable program state
- special assertions:
 - *true* a special predicate which is satisfied in *all program states*
 - *false* a special predicate which is satisfied in *no program state*



Example: facts and theorems in our programming domain

Soundness: all theorems should be true statements about the domain

- Following is not true, so it should NOT be a theorem, ie. not provable in our logic!:

$\{x==0\} x=x+1; \{y==1\}$ cannot miraculously set y to 1

Completeness: all true statements should be provable as theorems

- this fact should be a theorem, ie. it should be provable!):

$\{x==0\} x=x+1; \{x==1\}$



Axioms of the programming logic

- skip does not change any variable. Thus, if any predicate P is true before execution of *skip*, it remains true when skip terminates

$$\{P\} \text{ skip } \{P\}$$

- How about these:

$$\{P\} S \{\text{true}\}$$

$$\{\text{true}\} S \{P\} \text{ for any } S \text{ and } P ?$$



Assignment axiom: $\{P_{x \leftarrow e}\} x=e \{P\}$

- where $P_{x \leftarrow e}$ is P with "x replaced by e"

Example: $\{(x+1) \geq 5\} x = x+1 \{x \geq 5\}$

- $P_{x \leftarrow e}$ denotes *textual substitution*, i.e. substituting expression e for every free occurrence of x in P .

Example: $\{(\forall i: 1 \leq i \leq 7: a[i]=0)\} i = 3; n=7; \{(\forall i: 1 \leq i \leq n: a[i]=0)\}$

- If a variable in e has the same name as some bound variable x in P , then x in P must be renamed before the substitution, i.e. *resolving the name clash*:

Example: $\{(\forall i: 1 \leq i \leq n: a[i]=0)\} n=i; \{(\forall i: 1 \leq i \leq n: a[i]=0)\}$
 $\{(\forall j: 1 \leq j \leq i: a[j]=0)\} n=i; \{(\forall j: 1 \leq j \leq n: a[j]=0)\}$

Example of a proof: $\{\text{true}\} x=5 \{x==5\}$ is a theorem, since: $\{x==5\}_{x \leftarrow 5} == (5==5) = \text{true}$.



inference rules and proof outline

- composition rule
- *if* statement rule
- *while* statement rule
- rule of consequence

$$\frac{\{P\}S_1\{Q\}, \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}}$$

$$\frac{\{P \wedge B\}S\{Q\}, \{P \wedge \neg B\} \Rightarrow \{Q\}}{\{P\}if(B)S; \{Q\}}$$

$$\frac{\{I \wedge B\}S \{I\}}{\{I\} while \{B\} S; \{I \wedge \neg B\}}$$

$$\frac{P' \Rightarrow P, \{P\}S_1\{Q\}, Q \Rightarrow Q'}{\{P'\}S \{Q'\}}$$

- our proof obligation about a sequence of statements:
 - theorem to be proved: $\{P_0\}S_1; S_2; \dots; S_N; \{Q\}$
 - show that every triple in the *proof outline* holds:
 - $\{P_0\}S_1; \{P_1\}S_2; \dots \{P_{N-1}\}S_N; \{Q\}$
 - i.e. if the sequence starts from P_0 and terminates, it ends up in Q .



semantics of concurrent execution

- **await** *statement* has same effect as **if** *statement* w.r.t. partial correctness:

$$\frac{\{P \wedge B\}S\{Q\}, \{P \wedge \neg B\} \Rightarrow \{Q\}}{\{P\}\text{if}(B)S;\{Q\}}$$

- \Rightarrow **if** *statement* inference rule can be rewritten for await as:

$$\frac{\{P \wedge B\}S\{Q\}}{\{P\}\langle \text{await}(B)S;\rangle\{Q\}}$$

- **await** *statement* rule

- reads as:

“if execution of S begins in a state in which both P and B are true, and S terminates, then Q will be true”



co statement rule

assume

- `co S1; // S2; // ... // Sn; oc;`
- $\{P_i\} S_i \{Q_i\}$ true for each
- **co** statement rule:

$\{P_i\} S_i \{Q_i\}$ are interference free

$\{P_1 \wedge \dots \wedge P_n\}$

`co S1; // S2; // ... // Sn; oc,`

$\{Q_1 \wedge \dots \wedge Q_n\}$



interference

- A process **interferes** (*with the proof of*) another process if it executes an assignment that *invalidates an assertion* in the (*proof of*) other one
- Example: one process assigns to a shared variable and thus invalidates an assertion in the other process:

$\{x==0\}$	$\{x==0\}$
$x=x+1;$	$x=x+2;$
$\{x==1\}$	$\{x==2\}$



avoiding interference

- a set of processes is interference-free if no assignment action in one process interferes with any critical assertion in another
- techniques for avoiding interference
 - proving techniques: (no change in the program!)
 - disjoint variables
 - weakened assertions
 - global invariants
 - programming techniques: (these change in the program!)
 - bigger atomic actions: critical sections
 - synchronization



disjoint variables

- **write set** of a process: the variables that it alters
- **read set** of a process: the variables it reads but does not alter
- **reference set** of a process: the set of variables that appear in the assertions in a proof of that process.
- If the *write set* of process p is disjoint from the *reference set* of process q , then p cannot interfere with q .



weakened assertions

- even if the *write* and *reference sets* of processes overlap interference may be avoided by the *weakening assertions*
- a *weakened assertion* admits more correct states
- two processes (proofs) in isolation:

$$\begin{array}{ll} \{x==0\} & \{x==0\} \\ x=x+1; & x=x+2; \\ \{x==1\} & \{x==2\} \end{array}$$

- challenge: How to prove the following about their concurrent combination?

```
{x==0}
co ⟨x=x+1;⟩
// ⟨x=x+2;⟩
oc
{x==3}
```



weakened assertions (2)

- let's weaken the pre- and postconditions so that they cover both histories:

$$\begin{array}{ll} \langle x=x+1; \rangle & \langle x=x+2; \rangle \\ \langle x=x+2; \rangle & \langle x=x+1; \rangle \end{array}$$

```
{x==0}
co {x==0 ∨ x==2}
    ⟨x=x+1;⟩
    {x==1 ∨ x==3}
// {x==0 ∨ x==1}
    ⟨x=x+2;⟩
    {x==2 ∨ x==3}
oc
{x==3}
```

- our proof obligation:
 - show that all triples $\{P\}S\{Q\}$ in the proof outline are true
 - show that the processes do not *interference* with each other



non-interference

- an assertion in the proof outside $\langle \mathbf{await} \dots \rangle$ is called *critical*.
let: - s action in process p
 - $\text{pre}(s)$ precondition of s
 - C critical assertion in the proof of process q
- s does not interfere with C if the following is a theorem of p :

$$\{C \wedge \text{pre}(s)\} s \{C\}$$

- i.e. the execution of s in p does not invalidate the assertion C in the proof of q .

Notes: - $\langle \mathbf{await \ true \ s;} \rangle = \langle \mathbf{s;} \rangle$

- in s , $\text{pre}(s)$ and C all names clashes have been resolved.



non-interference: example

Prove that $\langle x=x+2 \rangle$ doesn't interfere with $\{x==0 \vee x==2\}$ in the proof of: $\langle x=x+1 \rangle$:

$$\begin{array}{c} \{x==0 \vee x==2\} \wedge \{x==0 \vee x==1\} \\ \langle x=x+2 \rangle \\ \{x==0 \vee x==2\} \end{array}$$

$$\begin{array}{c} \{C \wedge \text{pre}(s)\} \\ S \\ \{C\} \end{array}$$

Proof:

- $\{x==0 \vee x==2\} \wedge \{x==0 \vee x==1\} \Rightarrow \{x==0\}$
- $\{x==0\} \langle x=x+2; \rangle \{x==2\}$
- $\{x==2\} \Rightarrow \{x==0 \vee x==2\}$

Other non-interference can be proved similarly:

$$\begin{array}{c} \{x==0 \vee x==1\} \wedge \{x==0 \vee x==2\} \\ \langle x=x+1 \rangle \\ \{x==0 \vee x==1\} \end{array}$$

•So we have proved the whole program:

$$\begin{array}{l} \{x==0\} \\ \text{co } \{x==0 \vee x==2\} \\ \quad \langle x=x+1; \rangle \\ \quad \{x==1 \vee x==3\} \\ // \{x==0 \vee x==1\} \\ \quad \langle x=x+2; \rangle \\ \quad \{x==2 \vee x==3\} \\ \text{oc} \\ \{x==3\} \end{array}$$



global invariants

- A predicate I is a *global invariant*:
 - I is true when the processes begin execution
 - we can prove that I is preserved by every atomic action in any process (I is not interfered)
- I captures the *consistency criteria* of global state of the whole system



global invariants (2)

- suppose
 - I is a global invariant
 - every critical assertion C in the proof of every process p has the form $I \wedge L$, where L is *local assertion of* p , i.e.
 - each variable referenced in L is either local to process p or it is a global variable that is written by p only.
- if all critical assertions can be put into the form $I \wedge L$, then the proofs of the processes will be interference-free, because
 - I is not interfered by any action
 - no assignment action in one process can interfere with a local predicate L in another process



example: global invariant 1/3

```
x= y= z = 0;  
co while true < x = z+1>;  
// while true < y = x >;  
// while true < z = y >;  
oc
```

Many properties like could be proved, e.g.

$$I = \{ x \geq 0 \text{ and } y \geq 0 \text{ and } z \geq 0 \}$$

or:

$$I = \{ x \geq y \geq z \geq x-1 \}$$

...



example: global invariant 2/3

- How to prove that $I = \{x \geq y \geq z \geq x-1\}$ is a global invariant ?
- 1. Prove that I holds initially: $x=y=z=0; \{x=y=z\} \Rightarrow \{I\}$
- 2. Prove that no atomic action $\langle \dots \rangle$ invalidates I.
 - 2.1 Apply the assignment axiom to calculate the precondition P for each assignments a so that I will hold after a , *i.e.* $\{P\}a; \{I\}$

P' (reduced)	$\{P\}$	$a;$	$\{I\}$
$\{z+1 \geq y \geq z\}$	$\{z+1 \geq y \geq z \geq (z+1)-1\}$	$\langle x = z+1; \rangle$	$\{I\}$
$\{x \geq z \geq x-1\}$	$\{x \geq x \geq z \geq x-1\}$	$\langle y = x; \rangle$	$\{I\}$
$\{x \geq y \geq x-1\}$	$\{x \geq y \geq y \geq x-1\}$	$\langle z = y; \rangle$	$\{I\}$

2.2 Show that for each P' : $I \Rightarrow P'$

$$I = \{x \geq y \geq z \geq x-1\} = \{(x \geq y \geq z) \text{ and } (z+1 \geq x)\} \Rightarrow \{z+1 \geq y \geq z\}.$$

$$I = \{x \geq y \geq z \geq x-1\} \Rightarrow \{x \geq z \geq x-1\}.$$

$$I = \{x \geq y \geq z \geq x-1\} \Rightarrow \{x \geq y \geq x-1\}.$$

Remark: inside assertions and proof we use “ \Rightarrow ” instead of “ $=$ ” for equality when no confusion can arise.



example: global invariant 3/3

We have proven: $I = \{x \geq y \geq z \geq x-1\}$ for:

```
x= y= z =0; {I}
```

```
co while true {I} < x = z+1; > {I}
```

```
// while true {I} < y = x; > {I}
```

```
// while true {I} < z = y; > {I}
```

```
oc {I}
```



exercise: global invariant (1/3)

```
x= y= z= 0; {I}                                #I = {x ≥y ≥z ≥x-1}
co while true x = z+1; {I}
// while true y = x; {I}
// while true z = y; {I}
oc {I}
```

Prove that I holds with machine level atomic actions, load and store. Let's do this for the first assignment: “ $x=z+1$ ” by replacing it with two at-most-once assignments: “ $c=z+1; x=c;$ ” where c is local, i.e. they are atomic on the classical machine level. The other assignments can be modelled likewise.



exercise: global invariant (2/3)

$I = \{x \geq y \geq z \geq x-1\}$

Both $c=z+1$ and $x=c$ are at-most-once, and thus they can be considered atomic.

Prove the following outline:

```
x= y= z= c= 0 ; {I}
co while true  {I} c = z+1; {I ∧ (z+1 ≥ c ≥ x)} x = c; {I}
// while true  {I} y = x; {I}
// while true  {I} z = y; {I}
oc
```

Proof:

- calculate the precondition of “ $c=z; \{I\}$ ” using assignment axiom: $\{I \wedge (z+1 \geq c \geq x)\}$
- show that it's not interfered by “ $z=y;$ ”: $\{I \wedge (z+1 \geq c \geq x)\} z=y; \{I \wedge (z+1 \geq c \geq x)\}$
- show that it's not interfered by “ $y=z;$ ”: trivial
- show that it implies the precondition of “ $x=c;$ ” to maintain I:
 $\{I \wedge (z+1 \geq c \geq x)\} \Rightarrow \{c \geq y \geq z \geq c-1\} x=c; \{x \geq y \geq z \geq x-1\} = \{I\}$.
- a new global invariant : $I' = \{c \geq x \geq y \geq z \geq c-1\}$, such that: $I' \Rightarrow I$.



exercise: global invariant (3/3)

We proved $I' = \{c \geq x \geq y \geq z \geq c-1\}$ for

```
x= y= z= c= 0 ; {I'}
co while true  {I'} c = z+1; {I'} x = c; {I'}
// while true {I'} y = x; {I'}
// while true {I'} z = y; {I'}
oc
```

In this example we were able to break one non at-most-once action to two at-most-once atomic actions, by extending global invariance with a new (local) variable.



exercise2: global invariant

```
int x[n]; for [i =0 to n-1] x[i] = i; {I}
co [i =0 to n-1]{
    while true x[i]= x[i+1 (mod n) ]+1; {I}
} {I}
```

I?

What can we prove about it ?



avoid interference by mutual exclusion

- make bigger atomic actions with mutual exclusion to hide internal assertions from other processes:
 $\langle S1; \{C\}S2 \rangle$ #makes C invisible
- the *entire sequence* inside a $\langle \mathbf{await} \dots \rangle$ can be considered as one sequential program in isolation
- C is not interfered by others
- $\langle S1; \{C\}S2 \rangle$ can only interfere with others as one atomic action
- example $\langle p = p+1; \text{buff}[p] = x; \rangle$
- Increasing the grain of interleaving exclude harmful traces to happen



mutual exclusion

- internal states within angle brackets are invisible
=> no other process can interfere with it
- example: `I = {buff[p] = "next data item to be fetched"}`
`I < p=p+1; #I is temporarily broken here!!!`
`buff[p]= "new data";> I`
- Mind you: `<...>` implies always locking, which has it's price.



synchronization

consider

- `co P1: ... a; ...`
 `// P2: ... S1; {C} S2; ...`
 `oc`

where

- **a** assignment statement in process P1
- **C** critical assertion, precondition of S2
- **suppose:** **a** interferes with **C**



conditional synchronization

- `<await (!C or B) a;>`
where
B is a **predicate** such that executing **a** will make **C** true.
- Mind you: In general this is may cause deadlocks.



Literature

- <http://www.springer-ny.com/detail.tpl?cart=989601891383187&ISBN=0387949429>

On Concurrent Programming

Series: [Graduate Texts in Computer Science](#)

Fred B. Schneider, Cornell University of Ithaca, NY

Price: \$64.95

473 pages hardcover

ISBN: 0-387-94942-9, published 1997

Availability: In stock: Order now! Usually ships in 2-3 days.



Appendix: Formal Logic





formal logic: key concepts (1)

- a formal logical system is a mathematical abstraction
 - a collection of symbols and relations between them
- consists of *rules* defined in terms of
 - a set of *symbols*
 - a set of *formulas* constructed from these symbols (well-formed)
 - a set of distinguished formulas called *axioms* (assumed to be *a priori* true), and
 - a set of *inference* rules H_i is a *hypothesis*, C is a *conclusion*
If all the hypotheses are true, then we can infer that the conclusion is also true:

$$\frac{H_1, H_2, \dots, H_n}{C}$$



formal logic: key concepts (2)

- *formal proof*: a sequence of lines each of which is
 - an *axiom* or
 - derivable from previous lines by *inference rule*
- *theorem*: any line in a formal proof
- a logical system has a *domain* if :
 - its formulas represent statements in the domain
 - theorems are true statements in the domain
- *an interpretation* (of a logic) maps each formula to true or false in the domain



formal logic: key concepts (3)

- *sound logic* (with respect to an interpretation)
 - all axioms are sound: they map to true
 - all *inference rules* are sound: if all hypotheses map to true, conclusions maps to true
 - i.e. all theorems are true statements about the domain
 - if so, the interpretation is called a *model* for the logic
- *complete logic* (with respect to an interpretation)
 - every formula is provable in the logic



formal logic: key concepts (4)

- If
 - FACTS is the set of true statements expressible as formulas in a logic
 - THEOREMS is the set of theorems of the logic
- then
 - *soundness*: $\text{THEOREMS} \subseteq \text{FACTS}$
 - *completeness*: $\text{FACTS} \subseteq \text{THEOREMS}$
- if a logic is both sound and complete then all true statements expressible in it can be proved



formal logic: key concepts (5)

- any logic including arithmetics is bound to be incomplete (Gödel's incompleteness theorem)
- a logic that extends another one can be *relatively complete*, i.e. it does not introduce any (more) incompleteness
- relative completeness is good enough for our programming logic



predicate logic (1)

- extensions to a propositional logic
 - any expression such as $x < y$ that maps to true or false can be used in place of a propositional variable
 - \forall Universal quantifier
 - \exists Existential quantifier
- symbols
 - those of propositional logic, plus
 - variables, relational operators, and quantifiers
- formulas, called predicates
 - propositional formulas in which **relational** and **quantified expressions** can be used in place of propositional variables



predicate logic (2)

- relational expression
 - two terms separated by a relational operator
- relational operators
 - equals ($=$), not equal (\neq), greater than ($>$), member of (\in), subset (\subset)
- \exists existential quantifier, “there is”
($\exists x: R: P$)
- \forall universal quantifier, “for all”
($\forall x: R: P$)

– x *bound* variable
– R *range of values*
of the bound
variable x
– P predicate



free and bound variables

- in a quantified expression

$$(\exists b_1, \dots, b_n: R: P) \text{ or } (\forall b_1, \dots, b_n: R: P)$$

an occurrence of b_i is called a *bound* variable within the scope of the expression. Otherwise an occurrence of a variable is called *free*:

- it is not within the scope of a quantifier or
- it is within a quantifier and is different from the name of any bound variables

Example: in $(\forall i: 1 \leq i \leq n: a[i]=0)$ all occurrences of a and n are free and all occurrences of i are bound.



Symbols

- $\forall \exists \Leftrightarrow \Leftrightarrow \emptyset \supset \supset \subseteq \in \notin \Sigma \langle \rangle \neq \leq \geq$