Principles of Software Construction: Objects, Design, and Concurrency

Specification and Correctness

Jonathan Aldrich

Specifications

- Contains
  - Functional behavior
  - Erroneous behavior
  - Quality attributes

- Desirable attributes
  - Complete
    - Does not leave out any desired behavior
  - Minimal
    - Does not require anything that the user does not care about
  - Unambiguous
    - Fully specifies what the system should do in every case the user cares about
  - Consistent
    - Does not have internal contradictions
  - Testable
    - Feasible to objectively evaluate
  - Correct
    - Represents what the end-user(s) need

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility

- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running

- Example: how would you specify the following:

```
public float sum(int array[], int len) {… }
```

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility

- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running

- Example: how would you specify the following:

```
/*@ requires array != null && len >= 0 && array.length == len
    @
    @ ensures \result == (\sum int j; 0<=j && j<array.length;  array[j])
    @*/
public float sum(int array[], int len) {… }
```

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility

- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running

- (Functional) correctness with respect to the specification
  - If the client of a function fulfills the function's precondition, the function will execute to completion and when it terminates, the postcondition will be fulfilled

- What does the implementation have to fulfill if the client violates the precondition?

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility

- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running

- (Functional) correctness with respect to the specification
  - If the client of a function fulfills the function's precondition, the function will execute to completion and when it terminates, the postcondition will be fulfilled

- What does the implementation have to fulfill if the client violates the precondition?
  - A: Nothing. It can do anything at all.

Quick Quiz

Assume the specification for sum given in the lecture slides:

> **requires** array != null && len >= 0 && array.length == len
> **ensures** \result == (\sum int j; 0 <= j && j < len; array[j])

Assume the following input and outputs for sum, where a 3 element array is written as [1, 2, 3]. For which of the inputs and outputs is the call and implementation of sum correct according to the specification given?

- Input: array = [1, 2, 3, 4], len = 4
  Output: 10
- Input: array = [0, 0, 3, -7], len = 4
  Output: *none (the program does not terminate)*
- Input: array = [1, 2, 3, 4], len = 3
  Output: 7
- Input: array = [1, 2, -3, 4], len = 4
  Output: 7

Quick Quiz

Assume the specification for sum given in the lecture slides:

requires array != null && len >= 0 && array.length == len
ensures \result == (\sum int j; 0 <= j && j < len; array[j])

Assume the following input and outputs for sum, where a 3 element array is written as [1, 2, 3]. For which of the inputs and outputs is the call and implementation of sum correct according to the specification given?

- Input: array = [1, 2, 3, 4], len = 4
  Output: 10
  Call and implementation correct
- Input: array = [0, 0, 3, -7], len = 4
  Output: *none (the program does not terminate)*
  Implementation incorrect, it should terminate
- Input: array = [1, 2, 3, 4], len = 3
  Output: 7
  The call is incorrect (len should be 4)
- Input: array = [1, 2, -3, 4], len = 4
  Output: 7
  Implementation incorrect, output should be 4

Erroneous Behavior Specifications

- A function can do anything at all if precondition is violated, BUT…
  - we may want the system to function even if one part fails
  - we may want to easily identify our mistakes
- Exceptional case specifications
  - Precondition: condition describing the input that leads to an error
  - Postcondition: condition established by the function under that erroneous input
- Example (BitSet.toArray() in JML)

```
/*@ public normal_behavior
  @   requires a!= null;
  @   requires (\forall Object o; containsObject(o);
  @                     \typeof(o) <: \elemtype(\typeof(a)));
  @ also
  @ public exceptional_behavior
  @   requires a == null;
  @   signals_only NullPointerException ;
  @ also
  @ public exceptional_behavior
  @   requires a != null;
  @   requires !(\forall Object o; containsObject(o);
  @                     \typeof(o) <: \elemtype(\typeof(a)));
  @   signals_only ArrayStoreException ;
  @*/
 Object[] toArray(Object[] a) throws NullPointerException, ArrayStoreException;
```

Example Java I/O Library Specification (abridged)

public int **read**(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.
- If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.
- The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len. Let *k* be the number of bytes actually read; these bytes will be stored in elements b[off] throughb[off+*k*-1], leaving elements b[off+*k*] through b[off+len-1] unaffected.
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

- **Throws:**
  - IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
  - NullPointerException - If b is null.
  - IndexOutOfBoundsException - If off is negative, len is negative, or len is greater than b.length - off

Example Java I/O Library Specification (abridged)

public int **read**(byte[] b, int off, int len) throw

- Reads up to len bytes of data from th made to read as many as len bytes, b bytes actually read is returned as an i available, end of file is detected, or ar
- If len is zero, then no bytes are read a read at least one byte. If no byte is av value -1 is returned; otherwise, at leas
- The first byte read is stored into elem The number of bytes read is, at most, read; these bytes will be stored in ele elements b[off+*k*] through b[off+len-1]
- In every case, elements b[0] through 1] are unaffected.

- **Throws:**
  - IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
  - NullPointerException - If b is null.
  - IndexOutOfBoundsException - If off is than b.length - off

- Specification of return
- Timing behavior (blocks)
- Case-by-case spec
  - len=0 ➔ return 0
  - len>0 && eof ➔ return -1
  - len>0 && !eof ➔return >0
- Exactly where the data is stored
- What parts of the array are **not** affected

- Multiple error cases, each with a precondition
- Includes “runtime exceptions” not in throws clause

Quality Attribute Specifications: Discussion

- How would you specify…
  - Availability?
  - Modifiability?
  - Performance?
  - Security?
  - Usability?

Testing and Proofs

- Testing
  - Observable properties
  - Verify program for one execution
  - Manual development with automated regression
  - Most practical approach now
- Proofs
  - Any program property
  - Verify program for all executions
  - Manual development with automated proof checkers
  - Practical for small programs, may scale up in the future

- So why study proofs if they aren't (yet) practical?
  - Proofs tell us how to ***think*** about program correctness
    - Important for development, inspection, dynamic assertions
  - Foundation for static analysis tools
    - These are just simple, automated theorem provers
    - Many are practical today!

How would you argue that this program is correct?

```
/*@ requires len >= 0 && array.length == len
  @
  @ ensures \result ==
  @             (\sum int j;  0 <= j && j < len;  array[j])
  @*/
float sum(int array[], int len) {
	float sum = 0.0;
	int i = 0;
	while (i < len) {
		sum = sum + array[i];
		i = i + 1;
	}
	return sum;
}
```

Notation from the Java Modeling Language (JML)

Hoare Triples

- Formal reasoning about program correctness using pre- and postconditions

- Syntax: {P} S {Q}
  - P and Q are predicates
  - S is a program

- Semantics
  - If we start in a state where P is true and execute S, then S will terminate in a state where Q is true

Hoare Triple Examples

- { true } x := 5 { }
- { } x := x + 3 { x = y + 3 }
- { } x := x * 2 + 3 { x > 1 }
- { x=a } if (x < 0) then x := -x { }
- { false } x := 3 { }
- { x < 0 } while (x!=0) x := x-1 { }

Hoare Triple Examples

- { true } x := 5 { x=5 }
- { x = y } x := x + 3 { x = y + 3 }
- { x > -1 } x := x * 2 + 3 { x > 1 }
- { x=a } if (x < 0) then x := -x { x=|a| }
- { false } x := 3 { x = 8 }
- { x < 0 } while (x!=0) x := x-1 { }
  - no such triple!

Strongest Postconditions

- Here are a number of valid Hoare Triples:
  - {x = 5} x := x * 2 { true }
  - {x = 5} x := x * 2 { x > 0 }
  - {x = 5} x := x * 2 { x = 10 || x = 5 }
  - {x = 5} x := x * 2 { x = 10 }

Strongest Postconditions

- Here are a number of valid Hoare Triples:
  - {x = 5} x := x * 2 { true }
  - {x = 5} x := x * 2 { x > 0 }
  - {x = 5} x := x * 2 { x = 10 || x = 5 }
  - {x = 5} x := x * 2 { x = 10 }
    - All are true, but this one is the most *useful*
    - x=10 is the *strongest postcondition*

- If {P} S {Q} and for all Q' such that {P} S {Q'}, Q $\Rightarrow$ Q', then Q is the strongest postcondition of S with respect to P
  - check: x = 10 $\Rightarrow$ true
  - check: x = 10 $\Rightarrow$ x > 0
  - check: x = 10 $\Rightarrow$ x = 10 || x = 5
  - check: x = 10 $\Rightarrow$ x = 10

Assertion Strength

- A ***model*** is an assignment of variables to values
  - E.g. [x = 5]

- A logical ***assertion*** is a formula over variables that is true or false depending on a ***model***
  - x > 0 is true in the model [x=5]
  - x > 0 is false in the model [x=0]

- An assertion A is ***stronger*** than B if B is true in ***all*** models where A holds
  - Equivalently, A is stronger than B if A implies B
  - Example: x > 1 is stronger than x > 0

  - What is a model where x > 0 is true but x > 1 is false?

Assertion Strength

- A ***model*** is an assignment of variables to values
  - E.g. [x = 5]

- A logical ***assertion*** is a formula over variables that is true or false depending on a ***model***
  - x > 0 is true in the model [x=5]
  - x > 0 is false in the model [x=0]

- An assertion A is ***stronger*** than B
  if B is true in ***all*** models where A holds
  - Equivalently, A is stronger than B if A implies B
  - Example: x > 1 is stronger than x > 0

  - What is a model where x > 0 is true but x > 1 is false?
  - Answer: [x=1]

Hoare Triples, Revisited

- Syntax: {P} S {Q}
  - P and Q are predicates
  - S is a program

- Semantics
  - If we start in a state where P is true and execute S, then S will terminate in a state where Q is true

  - Note “state” just means a model in the sense above

Weakest Preconditions

- Here are a number of valid Hoare Triples:
  - {x = 5 && y = 10} z := x / y { z < 1 }
  - {x < y && y > 0} z := x / y { z < 1 }
  - {y ≠ 0 && x / y < 1} z := x / y { z < 1 }

Weakest Preconditions

- Here are a number of valid Hoare Triples:
  - {x = 5 && y = 10} z := x / y { z < 1 }
  - {x < y && y > 0} z := x / y { z < 1 }
  - {y ≠ 0 && x / y < 1} z := x / y { z < 1 }
    - All are true, but this one is the most *useful* because it allows us to invoke the program in the most general condition
    - y ≠ 0 && x / y < 1 is the *weakest precondition*

- If {P} S {Q} and for all P' such that {P'} S {Q}, P' $\Rightarrow$ P, then P is the weakest precondition *wp*(S,Q) of S with respect to Q

Hoare Triples and Weakest Preconditions

- {P} S {Q} holds if and only if P $\Rightarrow$ *wp*(S,Q)
  - In other words, a Hoare Triple is still valid if the precondition is stronger than necessary, but not if it is too weak

- Question: Could we state a similar theorem for a strongest postcondition function?
  - e.g. {P} S {Q} holds if and only if *sp*(S,P) $\Rightarrow$ Q

Hoare Triples and Weakest Preconditions

- {P} S {Q} holds if and only if P $\Rightarrow$ *wp*(S,Q)
  - In other words, a Hoare Triple is still valid if the precondition is stronger than necessary, but not if it is too weak

- Question: Could we state a similar theorem for a strongest postcondition function?
  - e.g. {P} S {Q} holds if and only if *sp*(S,P) $\Rightarrow$ Q
  - A: Yes, but it's harder to compute

Quick Quiz

Consider the following Hoare triples:

A) { z = y + 1 } x := z * 2 { x = 4 }

B) { y = 7 } x := y + 3 { x > 5 }

C) { false } x := 2 / y { true }

D) { y < 16 } x := 2 / y { x < 8 }

- Which of the Hoare triples above are invalid? What model witnesses the invalidity?

- Considering the valid Hoare triples, for which ones can you write a stronger postcondition? (Leave the precondition unchanged, and ensure the resulting triple is still valid)

- Considering the valid Hoare triples, for which ones can you write a weaker precondition? (Leave the postcondition unchanged, and ensure the resulting triple is still valid)

Quick Quiz

Consider the following Hoare triples:

A) { z = y + 1 } x := z * 2 { x = 4 }

Invalid. A witness is [z=1, y=0]

B) { y = 7 } x := y + 3 { x > 5 }

Valid. A weaker precondition is { y > 2 }.
A stronger postcondition is { x == 10 }

C) { false } x := 2 / y { true }

Valid (any Hoare triple with a false precondition is valid)
A weaker precondition is { y != 0 }
We can choose any postcondition; the strongest is { false }

D) { y < 16 } x := 2 / y { x < 8 }

Invalid. A witness is [y=0]

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?
    - What is most general value of y such that 3 + y > 0?
    - y > -3

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := E, P) = [E/x] P
    - Resulting triple: { [E/x] P } x := E { P }

Hoare Logic Rules

- Assignment
  - { P } x := 3 { x+y > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := E, P) = [E/x] P
    - Resulting triple: { [E/x] P } x := E { P }
  - [3 / x] (x + y > 0)
  - = (3) + y > 0
  - = y > -3

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := E, P) = [E/x] P

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := E, P) = [E/x] P
  - [3*y+z / x] (x * y – z > 0)

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := E, P) = [E/x] P
  - [3*y+z / x] (x * y – z > 0)
  - = (3*y+z) * y - z > 0

Hoare Logic Rules

- Assignment
  - { P } x := 3*y + z { x * y - z > 0 }
  - What is the weakest precondition P?
- Assignment rule
  - *wp*(x := E, P) = [E/x] P
  - [3*y+z / x] (x * y – z > 0)
  - = (3*y+z) * y - z > 0
  - = $3*y^2$ + z*y - z > 0

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)
  - = *wp*(x:=x+1, *wp*(y:=x+y, y>5))

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)
  - = *wp*(x:=x+1, *wp*(y:=x+y, y>5))
  - = *wp*(x:=x+1, x+y>5)

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
  - *wp*(x:=x+1; y:=x+y, y>5)
  - = *wp*(x:=x+1, *wp*(y:=x+y, y>5))
  - = *wp*(x:=x+1, x+y>5)
  - = x+1+y>5

Hoare Logic Rules

- Sequence
  - { P } x := x + 1; y := x + y { y > 5 }
  - What is the weakest precondition P?
- Sequence rule
  - $wp(S;T, Q) = wp(S, wp(T, Q))$
  - $wp(x{:=}x+1;\ y{:=}x+y,\ y>5)$
  - $= wp(x{:=}x+1,\ wp(y{:=}x+y,\ y>5))$
  - $= wp(x{:=}x+1,\ x+y>5)$
  - $= x+1+y>5$
  - $= x+y>4$

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - $wp(\text{if B then S else T, Q})$

    $= B \Rightarrow wp(S,Q)\ \&\&\ \neg B \Rightarrow wp(T,Q)$
  - $wp(\text{if x>0 then y:=z else y:=-z, y>5})$

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - *wp*(if B then S else T, Q)

    $= B \Rightarrow wp(S,Q)\ \&\&\ \neg B \Rightarrow wp(T,Q)$
  - *wp*(if x>0 then y:=z else y:=-z, y>5)
  - $= x>0 \Rightarrow wp(y{:=}z, y>5)\ \&\&\ x\leq 0 \Rightarrow wp(y{:=}{-z}, y>5)$

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - *wp*(if B then S else T, Q)
    $= B \Rightarrow wp(S,Q)$ && $\neg B \Rightarrow wp(T,Q)$
  - *wp*(if x>0 then y:=z else y:=-z, y>5)
  - = $x>0 \Rightarrow wp(y{:=}z, y>5)$ && $x \le 0 \Rightarrow wp(y{:=}{-z}, y>5)$
  - = $x>0 \Rightarrow z > 5$ && $x \le 0 \Rightarrow -z > 5$

Hoare Logic Rules

- Conditional
  - { P } if x > 0 then y := z else y := -z { y > 5 }
  - What is the weakest precondition P?
- Conditional rule
  - *wp*(if B then S else T, Q)

    $= B \Rightarrow wp(S,Q)\ \&\&\ \neg B \Rightarrow wp(T,Q)$
  - *wp*(if x>0 then y:=z else y:=-z, y>5)
  - $= x>0 \Rightarrow wp(y:=z, y>5)\ \&\&\ x\leq 0 \Rightarrow wp(y:=-z, y>5)$
  - $= x>0 \Rightarrow z > 5\ \&\&\ x\leq 0 \Rightarrow -z > 5$
  - $= x>0 \Rightarrow z > 5\ \&\&\ x\leq 0 \Rightarrow z < -5$

Reference: Hoare Logic Rules

- Assignment rule
  - *wp*(x := E, P) = [E/x] P
- Sequence rule
  - *wp*(S;T, Q) = *wp*(S, *wp*(T, Q))
- Conditional rule
  - *wp*(if B then S else T, Q)

    $= B \Rightarrow$ *wp*(S,Q) && $\neg B \Rightarrow$ *wp*(T,Q)

Quick Quiz

Compute the weakest precondition in each case.

(A) { } x = y * 2; { x == y * 2 }

(B) { } x = x + 3; { x == z }

(C) { } x = x + 1; y = y * x; { y == 2 * z }

(D) { } x = 0; { x == 1 }

(E) { } x = 0; { true }

(F) { } if (x > 0) then { y = x; } else { y = 0; } { y > 0 }

Quick Quiz

Compute the weakest precondition in each case.

(A) { true } x = y * 2; { x == y * 2 }

(B) { x + 3 = z } x = x + 3; { x == z }

(C) { y * (x + 1) == 2 * z } x = x + 1; y = y * x; { y == 2 * z }

(D) { false } x = 0; { x == 1 }

(E) { true } x = 0; { true }

(F) { x > 0 } if (x > 0) then { y = x; } else { y = 0; } { y > 0 }