Computer Science 203
Programming Languages

Axiomatic Semantics

Cormac Flanagan

University of California, Santa Cruz

Review of Operational Semantics

- Simple (relatively)
- Good for
  - language definition (eg IMP)
  - verifying language properties (eg IMP is deterministic)
  - verifying correctness of tools that manipulate programs
    - eg. interpreters, compilers, type checkers, etc

- Awkward for verifying even simple programs
- Not compositional

Axiomatic Semantics

- Tackles question: Is my program correct?
  - Does it satisfy its specification
- Two parts:
  - A language for writing program specifications
  - Rules for establishing that a program satisfies its specification
- Example specifications
  - This program terminates.
  - All array accesses are within array bounds, no null dereferences, and no unexpected exceptions
  - The method returns a sorted array
  - The variables x and y are always identical whenever z is 0
- Example specification languages
  - First-order logic.
  - Other logics (e.g., temporal logic).

History

- Program verification is almost as old as programming (e.g., “Checking a Large Routine”, Turing 1949)
- In the late ’60s, Floyd had rules for flowcharts and Hoare for a language similar to IMP.
- Since then, there have been axiomatic semantics for substantial languages, and many applications.

Dijkstra Said

- Program testing can be used to show the presence of bugs, but never to show their absence!

Hoare Said

- Thus the practice of proving programs would seem to lead to solution of three of the most pressing problems in software and programming, namely, reliability, documentation, and compatibility. However, program proving, certainly at present, will be difficult even for programmers of high caliber; and may be applicable only to quite simple program designs.

C.A.R Hoare,
"An Axiomatic Basis for
Computer Programming",
1969

Hoare Also Said

- It has been found a serious problem to define these languages [ALGOL, FORTRAN, COBOL] with sufficient rigour to ensure compatibility among all implementors. Since the purpose of compatibility is to facilitate interchange of programs expressed in the language, one way to achieve this would be to insist that all implementations of the language shall "satisfy" the axioms and rules of inference which underlie proofs of the properties of programs expressed in the language, so that all predictions based on these proofs will be fulfilled, except in the event of hardware failure. In effect, this is equivalent to accepting the axioms and rules of inference as the ultimately definitive specification of the meaning of the language.

  Another of the objectives of formal language definition is to assist in the design of better programming languages.

Other Applications of Axiomatic Semantics

- The project of defining and proving everything formally has not succeeded (at least not yet).
- Proving has not replaced testing (and praying).

- Some applications of axiomatic semantics:
  - Documentation of programs and interfaces.
  - Guidance in design and coding.
  - Proving the correctness of algorithms (or finding bugs).
  - Proving the correctness of hardware descriptions (or finding bugs).
  - "Extended static checking" (e.g., checking array bounds).
  - Proof-carrying code.

Assertions for IMP

- *Partial* correctness assertion: {A} c {B }

  If A holds in state σ

  and there exists σ' such that $\langle c, \sigma \rangle \Downarrow \sigma'$

  then B holds in σ'.

- *Total* correctness assertion: [A] c [B]

  If A holds in state σ

  then there exists σ' such that $\langle c, \sigma \rangle \Downarrow \sigma'$

  and B holds in state σ'.

- These are called Hoare triples.
- A is called precondition and B is called postcondition.
- Example: $\{ y \cdot x \}$ z := x; z := z +1 $\{ y < z \}$

The Assertion Language

- We re-use IMP boolean expressions as our assertion language

$A, B ::= b$ (IMP boolean expressions)

We say σ satisfies A (written $\sigma \vDash A$) if $\langle A, \sigma \rangle \Downarrow$ true

Semantics of Assertions

- Now we can define formally the meaning of a partial correctness assertion.

$\models \{ A \}\ c\ \{ B \}$ holds if and only if

$\forall\sigma\in\Sigma.\ \sigma \models A \Rightarrow [\forall\sigma'\in\Sigma.\ \langle c,\sigma\rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]$

- ... and the meaning of a total correctness assertion.

$\models [A]\ c\ [B]$ holds if and only if

$\forall\sigma\in\Sigma.\ \sigma \models A \Rightarrow [\exists\sigma'\in\Sigma.\ \langle c,\sigma\rangle \Downarrow \sigma' \wedge \sigma' \models B]$

- ... which simplifies to

$\forall\sigma\in\Sigma.\ \sigma \models A \Rightarrow [\forall\sigma'\in\Sigma.\ \langle c,\sigma\rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]$
$\wedge \forall\sigma\in\Sigma.\ \sigma \models A \Rightarrow \exists\sigma'\in\Sigma.\ \langle c,\sigma\rangle \Downarrow \sigma'$

Deriving Assertions

- Formal definition of $\models \{ A \}\ c\ \{ B \}$ is difficult to use

$\models \{ A \}\ c\ \{ B \}$ holds if and only if

$$\forall \sigma \in \Sigma.\ \sigma \models A \Rightarrow [\forall \sigma' \in \Sigma.\ \langle c, \sigma \rangle \Downarrow \sigma' \Rightarrow \sigma' \models B]$$

  - defined in terms of the operational semantics
  - $\forall \sigma$

- So we define a symbolic technique (ie, a logic)
  for deriving valid triples $\vdash \{A\}\ c\ \{ B\}$ from other valid triples.

Derivation Rules for Hoare Triples

- Similarly we write $\vdash \{A\}\ c\ \{ B\}$ when we can derive the triple using derivation rules.
- There is one derivation rule for each command in the language.
- Plus, the rule of consequence:

$$\frac{A' \Rightarrow A \qquad \vdash \{A\}\ c\ \{B\} \qquad B \Rightarrow B'}{\vdash \{A'\}\ c\ \{B'\}}$$

Derivation Rules for Hoare Logic $\vdash \{A\}\ c\ \{\ B\}$

- One rule for each language construct (plus one more)

$$\frac{}{\vdash \{A\}\ \text{skip}\ \{A\}} \qquad \frac{}{\vdash \{[e/x]A\}\ x := e\ \{A\}}$$

$$\frac{\vdash \{A\}\ c_1\ \{B\} \qquad \vdash \{B\}\ c_2\ \{C\}}{\vdash \{A\}\ c_1;\ c_2\ \{C\}} \qquad \frac{\vdash \{A \wedge b\}\ c_1\ \{B\} \qquad \vdash \{A \wedge \neg\, b\}\ c_2\ \{B\}}{\vdash \{A\}\ \text{if } b \text{ then } c_1 \text{ else } c_2\ \{B\}}$$

$$\frac{\vdash \{A \text{ Æ } b\}\ c\ \{A\}}{\vdash \{A\}\ \text{while } b \text{ do } c\ \{A \text{ Æ } \neg\, b\}}$$

Rule of Consequence

$$\frac{A' \Rightarrow A \qquad \vdash \{A\}\ c\ \{B\} \qquad B \Rightarrow B'}{\vdash \{A'\}\ c\ \{B'\}}$$

Hoare Rules

- For some constructs multiple rules are possible:

$$\frac{}{\vdash \{A\}\ x := e\ \{\exists x_0.[x_0/x]A \wedge x = [x_0/x]e\}}$$

(This was the “forward” axiom for assignment.)

$$\frac{\vdash A \wedge b \Rightarrow I \quad \vdash \{I\}\ c\ \{A\} \quad \vdash A \wedge \neg\, b \Rightarrow B}{\vdash \{A\}\ \text{while } b \text{ do } c\ \{B\}}$$

- Exercise: these rules can be derived from the previous ones using the rule of consequence.

Example: Assignment

- Assume that x does not appear in e

  Prove that {true} x := e { x = e }
- But

$$\frac{}{\vdash \{e = e\}\ x := e\ \{x = e\}}$$

because $[e/x](x = e) \equiv e = [e/x]e \equiv e = e$

- Assignment + consequence:

$$\frac{\text{true} \Rightarrow e = e \qquad \dfrac{}{\vdash \{e = e\}\ x := e\ \{x = e\}}}{\vdash \{\text{true}\}\ x := e\ \{x = e\}}$$

The Assignment Axiom (Cont.)

- Hoare said: "Assignment is undoubtedly the most characteristic feature of programming a digital computer, and one that most clearly distinguishes it from other branches of mathematics. It is surprising therefore that the axiom governing our reasoning about assignment is quite as simple as any to be found in elementary logic."

- Caveats are needed for languages with aliasing:
  - If x and y are aliased then

    { true } x := 5 { x + y = 10}

    is true

Example: Conditional

$$\frac{\begin{array}{c} D_1 :: \vdash \{\text{true} \wedge y >= 0\}\ x := 1\ \{x > 0\} \\ D_2 :: \vdash \{\text{true} \wedge y > 0\}\ x := y\ \{x > 0\} \end{array}}{\vdash \{\text{true}\}\ \text{if } y \cdot 0 \text{ then } x := 1 \text{ else } x := y\ \{x > 0\}}$$

- D_1 is obtained by consequence and assignment

$$\frac{\text{true} \wedge y >= 0 \Rightarrow 1 > 0 \qquad \vdash \{1 > 0\}\ x := 1\ \{x > 0\}}{\vdash \{\text{true} \wedge y >= 0\}\ x := 1\ \{x > 0\}}$$

- D_2 is also obtained by consequence and assignment

$$\frac{\text{true} \wedge y > 0 \Rightarrow y > 0 \qquad \vdash \{y > 0\}\ x := y\ \{x > 0\}}{\vdash \{\text{true} \wedge y > 0\}\ x := y\ \{x > 0\}}$$

Example: Loop

- We want to derive that

$$\vdash \{x \cdot 0\}\ \text{while}\ x \cdot 5\ \text{do}\ x := x + 1\ \{\ x = 6\}$$

- We use the rule for while with invariant $x \cdot 6$:

$$\dfrac{\dfrac{x \cdot 6 \wedge x \cdot 5 \Rightarrow x + 1 \cdot 6 \qquad \dfrac{}{\vdash \{x + 1 \cdot 6\}\ x := x + 1\ \{\ x \cdot 6\ \}}}{\vdash \{x \cdot 6\ \text{Æ}\ x \cdot 5\ \}\ x := x + 1\ \{x \cdot 6\}}}{\vdash \{x \cdot 6\}\ \text{while}\ x \cdot 5\ \text{do}\ x := x + 1\ \{\ x \cdot 6\ \text{Æ}\ x > 5\}}$$

- We finish off with consequence:

$$\dfrac{\begin{array}{c} x \cdot 0 \Rightarrow x \cdot 6 \\ x \cdot 6\ \text{Æ}\ x > 5 \Rightarrow x\ = 6 \end{array} \qquad \vdash \{x \cdot 6\}\ \text{while} \ldots \{\ x \cdot 6\ \text{Æ}\ x > 5\}}{\vdash \{x \cdot 0\}\ \text{while} \ldots \{x = 6\}}$$

Another Example

- Verify that

  $$\vdash \{A\}\ \text{while true do}\ c\ \{\ B\}$$

  holds for any A, B, and c.

- We must construct a derivation tree

$$\frac{\begin{array}{c} A \Rightarrow \text{true} \\ \text{true Æ false} \Rightarrow B \end{array} \qquad \dfrac{\vdash \{\text{true Æ true}\}\ c\ \{\ \text{true}\ \}}{\{\text{true}\}\ \text{while true do}\ c\ \{\text{true Æ false}\}}}{\vdash \{A\}\ \text{while true do}\ c\ \{\ B\}}$$

- We need an additional lemma:

  $$\forall A. \forall c. \vdash \{\ A\ \}\ c\ \{\text{true}\}$$

Notes on Using Hoare Rules

- Hoare rules are mostly syntax directed.

- There are three wrinkles:
  - When to apply the rule of consequence ?
  - What invariant to use for while ?
  - How do you prove the implications involved in consequence ?

- The last one involves theorem proving:
  - This turns out to be doable.
  - The loop invariants turn out to be the hardest problem ! (Should the programmer give them?)

Where Do We Stand?

- We have a language for asserting properties of programs.
- We know when an assertion is true.
- We also have a symbolic method for deriving assertions.

$$\begin{tikzcd}
\vdash \{A\}\ c\ \{B\} \arrow[r, bend left, "\text{soundness}"] & \models \{A\}\ c\ \{B\} \arrow[l, bend left, "\text{completeness}"]
\end{tikzcd}$$

Soundness of Axiomatic Semantics

- Formal statement of soundness:

  If $\vdash$ {A} c {B} then $\models$ {A} c {B}.

  or, equivalently

  For all σ, if $\sigma \models$ A and D :: <c, σ> $\Downarrow$ σ'

  and H :: $\vdash$ {A} c {B} then $\sigma' \models$ B.

  - Proof: simultaneous induction on the structure of D and H.

Completeness of Axiomatic Semantics

- Formal statement of completeness:

  If $\models \{A\}\ c\ \{B\}$ then $\vdash \{A\}\ c\ \{B\}$.

  or, equivalently

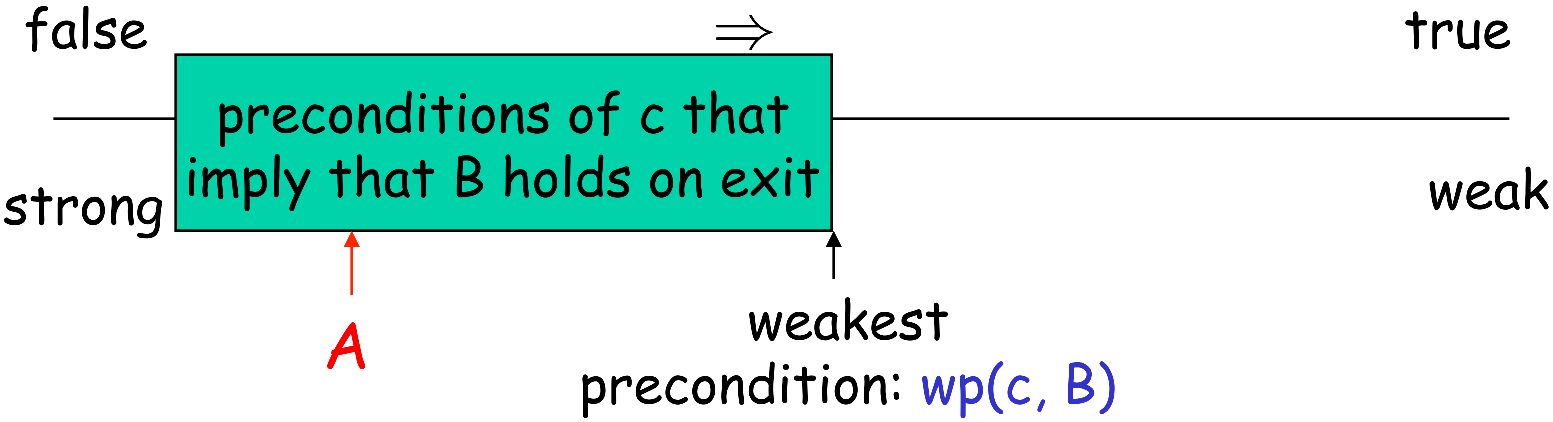
  Suppose that, for all σ, D, σ',
  if $\sigma \models A$ and $D :: \langle c, \sigma\rangle \Downarrow \sigma'$ then $\sigma' \models B$.
  Then there exists H such that $H :: \vdash \{A\}\ c\ \{B\}$.

  - Proof: harder, and requires an assumption that says that loop invariants can be expressed as logical formulas.

(See slides on-line and Winskel's book for more.)

Weakest Preconditions (Dijkstra)

- Assertions can be ordered:


false
⇒
true
preconditions of c that imply that B holds on exit
strong
weak
A
weakest precondition: wp(c, B)

- Thus: to verify {A} c {B }, we may compute wp(c, B) and prove $A \Rightarrow$ wp(c, B).

Weakest Preconditions

- Define wp(c, B) inductively on c, following Hoare rules:

$$\frac{\{A\}\ c_1\ \{C\} \qquad \{C\}\ c_2\ \{B\}}{\{\ A\ \}\ c_1;\ c_2\ \{B\}}$$

$$wp(c_1;\ c_2, B) = wp(c_1, wp(c_2, B))$$

$$\frac{}{\{\ [e/x]B\ \}\ x := E\ \{B\}}$$

$$wp(x := e, B) = [e/x]B$$

$$\frac{\{A\}\ c_1\ \{B\} \qquad \{A'\}\ c_2\ \{B\}}{\{\ E \Rightarrow A\ \text{Æ}\ \neg\ E \Rightarrow A'\}\ \text{if } E \text{ then } s_1 \text{ else } s_2\ \{B\}}$$

$$wp(\text{if } E \text{ then } c_1 \text{ else } c_2, B) = E \Rightarrow wp(c_1, B)\ \text{Æ}\ \neg\ E \Rightarrow wp(c_2, B)$$

Weakest Preconditions for Loops

- We start from the equivalence

  while b do c = if b then c; while b do c else skip

- Let W = wp(while b do c, B)

- We have that

  $$W = (b \Rightarrow wp(c, W) \text{ Æ } \neg\, b \Rightarrow B)$$

- But this is a recursive equation !
  - We know how to solve these… in a complete partial order.
  - Least solutions may or may not exist over formulas.

Extra material on axiomatic semantics: soundness and completeness

(covered in the lectures
only as time permits)

Hoare Rules: Assignment and References

- When is the following Hoare triple valid?

  { A } *x = 5 { *x + *y = 10 }

- A ought to be "*y = 5 or x = y"

- The Hoare rule for assignment would give us:

  [5/*x](*x + *y = 10)

  = 5 + *y = 10

  = *y = 5 (we lost one case)

- How come the rule does not work?

Hoare Rules: Assignment and References (Cont.)

- To model writes correctly we use memory expressions.
  - A memory write changes the value of memory

$$\frac{}{\{\ B[\mathrm{upd}(M, E_1, E_2)/M]\ \}\ {*}E_1 := E_2\ \{B\}}$$

- Important technique:
  - Treat memory as a whole.
  - And reason about memory expressions with rules such as McCarthy's:

$$\mathrm{sel}(\mathrm{upd}(M, E_1, E_2), E_3) = \begin{cases} E_2 & \text{if } E_1 = E_3 \\ \mathrm{sel}(M, E_3) & \text{if } E_1 \neq E_3 \end{cases}$$

Memory Aliasing

- Consider again: $\{ A \}\ {*x} := 5\ \{ {*x} + {*y} = 10 \}$
- We obtain:

$$\begin{aligned}
A &= [\mathrm{upd}(M, x, 5)/M]\ ({*x} + {*y} = 10) \\
&= [\mathrm{upd}(M, x, 5)/M]\ (\mathrm{sel}(M, x) + \mathrm{sel}(M, y) = 10) \\
&= \mathrm{sel}(\mathrm{upd}(M, x, 5), x) + \mathrm{sel}(\mathrm{upd}(M, x, 5), y) = 10 \\
&= 5 + \mathrm{sel}(\mathrm{upd}(M, x, 5), y) = 10 \\
&= \text{if } x = y \text{ then } 5 + 5 = 10 \text{ else } 5 + \mathrm{sel}(M, y) = 10 \\
&= x = y \text{ or } {*y} = 5
\end{aligned}$$

Mutable Records

- Let r : RECORD f1 : T1; f2 : T2 END
- One method for handling records:
  - One "memory" for each field
  - The record address is the index
  - r.f1 is sel(f1,r) and r.f1 := E is f1 := upd(f1,r,E)

Soundness of Axiomatic Semantics

- Formal statement

  If $\vdash$ { A } c { B} then $\models$ { A} c { B}.

  or, equivalently

  For all σ, if $\sigma \models$ A and D :: <c, σ> $\Downarrow$ σ'

  and H :: $\vdash$ { A } c { B} then $\sigma' \models$ B.

- How can we prove this?
  - By induction on the structure of c?
    - No, problems with while and rule of consequence.
  - By induction on the structure of D?
    - No, problems with rule of consequence.
  - By induction on the structure of H?
    - Not quite, problems with while.
  - By simultaneous induction on the structure of D and H.

Simultaneous Induction

- Consider two structures D and H.
  - Assume that $x < y$ iff x is a substructure of y.
- Define the lexicographic ordering

  $(d, h) < (d', h')$ iff $d < d'$ or $d = d'$ and $h < h'$
- This is a well-founded order and serves for a simultaneous induction.

- If $d < d'$ then h can actually be larger than h'.
- It can even be unrelated to h'.

Soundness of the Consequence Rule

- Case: last rule used in H :: $\vdash$ { A} c { B} is the consequence rule:

$$\frac{\vdash A \Rightarrow A' \qquad H_1 :: \vdash \{A'\}\ c\ \{B'\} \qquad \vdash B' \Rightarrow B}{\vdash \{A\}\ c\ \{B\}}$$

- From soundness of the first-order logic derivations we have $\sigma \vDash A \Rightarrow A'$, hence $\sigma \vDash A'$

- From IH with H_1 and **D** we get that $\sigma' \vDash B'$.

- From soundness of the first-order logic derivations we have that $\sigma' \vDash B' \Rightarrow B$, hence $\sigma' \vDash B$, q.e.d.

Soundness of the Assignment Axiom

- Case: the last rule used in H :: ⊢ { A } c { B} is the assignment rule

$$\frac{}{\vdash \{[e/x]B\}\ x := e\ \{B\}}$$

- The last rule used in D :: <x := e, σ> ⇓ σ′ must be

$$\frac{D_1 :: \langle e, \sigma \rangle \Downarrow n}{\langle x := e, \sigma \rangle \Downarrow \sigma[x := n]}$$

- We must prove the substitution lemma:

  If σ ⊨ [e/x]B and <σ, e> ⇓ n then σ ⊨ [n/x]B

Soundness of the While Rule

- Case: last rule used in H : $\vdash$ { A } c { B} is the while rule:

$$\frac{H_1 :: \vdash \{A \text{ Æ } b\}\ c\ \{A\}}{\vdash \{A\}\ \text{while } b \text{ do } c\ \{A \text{ Æ } \neg\, b\}}$$

- There are two possible rules at the root of D.
  - We do only the complicated case:

$$\frac{D_1 :: \langle b, \sigma\rangle \Downarrow \text{true} \qquad D_2 :: \langle c,\sigma\rangle \Downarrow \sigma' \qquad D_3 :: \langle \text{while } b \text{ do } c, \sigma' \rangle \Downarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma''}$$

Soundness of the While Rule (Cont.)

Assume that σ ⊨ A

To show that σ'' ⊨ A Æ ¬ b

- By property of booleans and D_1 we get σ ⊨ b.
  - Hence σ ⊨ A Æ b.
- By IH on H_1 and D_2 we get σ' ⊨ A.
- By IH on H and D_3 we get σ'' ⊨ A Æ ¬ b, q.e.d.

- Note that in the last use of IH the derivation H did not decrease.
- See Winskel, Chapter 6.5 for a soundness proof with denotational semantics.

Completeness of Axiomatic Semantics

- Is it true that whenever $\models \{A\}\ c\ \{B\}$ we can also derive $\vdash \{A\}\ c\ \{B\}$?
- If it isn't then it means that there are valid properties of programs that we cannot verify with Hoare rules.

- Good news: for our language the Hoare triples are complete.
- Bad news: only if the underlying logic is complete

  (whenever $\models A$ we also have $\vdash A$).
  - This is called relative completeness.
  - The underlying logic must also be expressive enough.

A Partial Order for Assertions

- What is the assertion that contains least information?
  - true – it does not say anything about the state.
- What is an appropriate information ordering ?

$$A \sqsubseteq A' \quad \text{iff} \quad \models A' \Rightarrow A$$

- Is this partial order complete?
  - Take a chain $A_1 \sqsubseteq A_2 \sqsubseteq \ldots$
  - Let ÆA_i be the infinite conjunction of A_i

    $\sigma \models$ ÆA_i iff for all i we have that $\sigma \models A_i$
  - Verify that ÆA_i is the least upper bound.
- Can ÆA_i be expressed in our language of assertions?
  - In many cases yes (see Winskel); we'll assume so.

Weakest Precondition for WHILE

- Use the fixed-point theorem

  $$F(A) = (b \Rightarrow wp(c, A)\ \text{Æ}\ \neg\, b \Rightarrow B)$$

  - Verify that F is both monotone and continuous.

- The least fixed point (i.e., the weakest fixed point) is

  $$wp(\text{while } b \text{ do } c, B) = \text{Æ}F^i(\text{true})$$

- Notice that unlike for the denotational semantics of IMP, we are not working on a flat domain.

Proof Idea for Completeness

- Completeness of axiomatic semantics:

  If $\models \{ A \}\ c\ \{ B \}$ then $\vdash \{ A \}\ c\ \{ B\}$

- Assuming that we can compute wp(c, B) with the following properties:
  1. wp is a precondition (according to the Hoare rules)

     $\vdash \{ wp(c, B) \}\ c\ \{ B\}$
  2. wp is the weakest precondition:

     If $\models \{ A \}\ c\ \{ B\}$ then $\models A \Rightarrow wp(c, B)$.

$$\frac{\vdash A \Rightarrow wp(c, B) \qquad \vdash \{wp(c, B)\}\ c\ \{B\}}{\vdash \{A\}\ c\ \{B\}}$$

- We also need that whenever $\models A$ then $\vdash A$.