CS 591: Formal Methods in Security and Privacy
Semantics of programs

Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu
Programming Language

\[
c ::= \text{abort} \\
| \text{skip} \\
| x := e \\
| c ; c \\
| \text{if } e \text{ then } c \text{ else } c \\
| \text{while } e \text{ do } c
\]

\[x, y, z, \ldots\] program variables

\[e_1, e_2, \ldots\] expressions

\[c_1, c_2, \ldots\] commands
Expressions

We want to be able to write complex programs with our language.

\[ e ::= x \]
\[ \quad | \quad f(e_1, \ldots, e_n) \]

Where \( f \) can be any arbitrary operator.

Some expression examples

\[ x+5 \quad x \mod k \quad x[i] \quad (x[i+1] \mod 4)+5 \]
Semantics of Expressions

This is defined on the structure of expressions:

\[ \{ x \}_m = m(x) \]

\[ \{ f(e_1, \ldots, e_n) \}_m = \{ f \}(\{ e_1 \}_m, \ldots, \{ e_n \}_m) \]

where \( \{ f \} \) is the semantics associated with the basic operation we are considering.
Semantics of Commands

What is the meaning of the following command?

\[ k := 2; \quad z := x \mod k; \quad \text{if } z = 0 \text{ then } r := 1 \text{ else } r := 2 \]

We can give the semantics as a relation between command, memories and memories or failure.

\[ \text{Exp} \times \text{Mem} \rightarrow (\text{Mem} + \perp) \]

We will denote this relation as:

\[ \{ c \}_m = m' \quad \text{or} \quad \{ c \}_m = \perp \]

This is commonly typeset as:

\[ \llbracket c \rrbracket_m = m' \]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \bot \\
\{\text{skip}\}_m &= m \\
\{x:=e\}_m &= m[x\leftarrow\{e\}_m] \\
\{c;c'\}_m &= \{c'\}_m', \quad \text{if} \quad \{c\}_m = m' \\
\{c;c'\}_m &= \bot, \quad \text{if} \quad \{c\}_m = \bot \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_t\}_m, \quad \text{if} \quad \{e\}_m=\text{true} \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_f\}_m, \quad \text{if} \quad \{e\}_m=\text{false}
\end{align*}
\]
Semantics of While

If \( \{e\}_m = \text{false} \) Then

\( \{\text{while } e \text{ do } c\}_m = m \)

What about when \( \{e\}_m = \text{true} \)?
Semantics of While

If \( \{e\}_m = \text{true} \) Then we would like to have:

\[
\{\text{while } e \text{ do } c\}_m = \{c;\text{while } e \text{ do } c\}_m
\]

Is this well defined?
Today: more on semantics of While
Well defined?

What is the semantics of the following program:

```
n:=2;
r:=1;
while n ≥ 1 do
  r := n * r;
n := n-1;
```

\{while e do c\}_m = \{c;\text{while e do c}\}_m
Approximating While

We could define the following syntactic approximations of a While statement:

```
while^n e do c
```
Approximating While

We could define the following syntactic approximations of a While statement:

\[
\text{while}^n \ e \ \text{do} \ c
\]

This can be defined inductively on \( n \) as:

\[
\text{while}^0 \ e \ \text{do} \ c = \text{skip}
\]

\[
\text{while}^{n+1} \ e \ \text{do} \ c = \text{if} \ e \ \text{then} \ (c;\text{while}^n \ e \ \text{do} \ c) \ \text{else} \ \text{skip}
\]
Semantics of While

We could go back and try to define the semantics using the approximations:

\[
\{\text{while } e \text{ do } c\}_m = \{\text{while}^n e \text{ do } c\}_m
\]
Semantics of While

We could go back and try to define the semantics using the approximations:

\[
\{\text{while } e \text{ do } c\}_m = \{\text{while}^n e \text{ do } c\}_m
\]

How do we find the n?
Information order

An idea that has been developed to solve this problem is the idea of information order.

This corresponds to the idea of order different possible denotations in terms of the information they provide.

In our case, we can use the following order on possible outputs:

\[ m_1 \geq m_2 \geq m_3 \geq \ldots \geq m_n \ldots \]
Semantics of While

Using fixpoint theorems on lattices we can try now to define the semantics using the approximations and a sup operation:

\[ \{ \text{while } e \text{ do } c \}_m = \sup_{n \in \text{Nat}} \{ \text{while}^n e \text{ do } c \}_m \]
Semantics of While

Using fixpoint theorems on lattices we can try now to define the semantics using the approximations and a sup operation:

\[
\{\text{while } e \text{ do } c\}_m = \sup_{n \in \text{Nat}} \{\text{while } e \text{ do } c\}_n
\]

Will this work?
Semantics of While

Using fixpoint theorems on lattices we can try now to define the semantics using the approximations and a sup operation:

$$\{\text{while } e \text{ do } c\}_m = \operatorname{sup}_{n \in \text{Nat}} \{\text{while } e \text{ do } c\}_n$$

Will this work?

We are missing the base case.
Approximating While Revisited

We could define the following lower iteration of a While statement:

\[ \text{while}_n \ e \ \text{do} \ c \]

This can be defined using the approximations as:

\[ \text{while}_n \ e \ \text{do} \ c = \]
\[ \text{while}_n \ e \ \text{do} \ c; \text{if} \ e \ \text{then} \ \text{abort} \ \text{else} \ \text{skip} \]
Semantics of While

We now have all the components to define the semantics of while:

\[
\{\text{while } e \text{ do } c\}_m = \sup_{n \in \mathbb{N}} \{\text{while}_n e \text{ do } c\}_m
\]
Semantics of Commands

This is defined on the structure of commands:

\[\{\text{abort}\}_m = \bot\]
\[\{\text{skip}\}_m = m\]
\[\{x:=e\}_m = m[x\leftarrow\{e\}_m]\]
\[\{c;c'\}_m = \{c'\}_m', \quad \text{if} \quad \{c\}_m = m'\]
\[\{c;c'\}_m = \bot, \quad \text{if} \quad \{c\}_m = \bot\]
\[\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_t\}_m, \quad \text{if} \quad \{e\}_m = \text{true}\]
\[\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_f\}_m, \quad \text{if} \quad \{e\}_m = \text{false}\]
\[\{\text{while } e \text{ do } c\}_m = \sup_{n\in\text{Nat}}\{\text{while}_n e \text{ do } c\}_m\]

where

\[\text{while}_n e \text{ do } c = \text{while}_n e \text{ do } c;\text{if } e \text{ then } \text{abort} \text{ else skip}\]

and
Semantics of Commands
This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \bot \\
\{\text{skip}\}_m &= m \\
\{x := e\}_m &= m[x \leftarrow \{e\}_m] \\
\{c;c'\}_m &= \{c'\}_m, \quad \text{if} \quad \{c\}_m = m' \\
\{c;c'\}_m &= \bot, \quad \text{if} \quad \{c\}_m = \bot \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_t\}_m, \quad \text{if } \{e\}_m = \text{true} \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_f\}_m, \quad \text{if } \{e\}_m = \text{false} \\
\{\text{while } e \text{ do } c\}_m &= \sup_{n \in \mathbb{N}} \{\text{while}_n e \text{ do } c\}_m \\
\end{align*}
\]

where

\[
\begin{align*}
\text{while}_n e \text{ do } c &= \text{while}_n e \text{ do } c; \text{if } e \text{ then abort } \text{ else } \text{skip} \\
\text{while}_0 e \text{ do } c &= \text{skip} \\
\text{while}_{n+1} e \text{ do } c &= \text{if } e \text{ then } (c; \text{while}_n e \text{ do } c) \text{ else } \text{skip}
\end{align*}
\]
Example

What is the semantics of the following program:

\begin{verbatim}
n:=2;
r:=1;
while n ≥ 1 do
  r := n * r;
n := n-1;
\end{verbatim}
Example

What is the semantics of the following program:

```
Fact(n: Nat) : Nat
  r := 1;
  while n > 1 do
    r := n * r;
    n := n - 1;
```
Hoare Triples
Hoare triple

$c : P \Rightarrow Q$

- **Precondition** (a logical formula)
- **Program**
- **Postcondition** (a logical formula)
Some examples

\[ x = z + 1 : \{ z > 0 \} \implies \{ x > 1 \} \]
Some examples

\[ x = z + 1 : \{ z > 0 \} \Rightarrow \{ x > 1 \} \]

Is it valid?
Some examples

\[ x = z + 1 : \{ z > 0 \} \Rightarrow \{ x > 0 \} \]
Some examples

\[ x = z + 1 : \{ z > 0 \} \Rightarrow \{ x > 0 \} \]

Is it valid?
Some examples

\[ x = z + 1 : \{ z < 0 \} \Rightarrow \{ x < 0 \} \]
Some examples

\[ x = z + 1 : \{ z < 0 \} \Rightarrow \{ x < 0 \} \]

Is it valid?
Some examples

\[ x = z + 1 : \{ z = n \} \Rightarrow \{ x = n + 1 \} \]
Some examples

\[ x = z + 1 : \{ z = n \} \Rightarrow \{ x = n + 1 \} \]

Is it valid?
Some examples

```
while x>0
    z=x*2+y
    x=x/2
    z=x*2-y
```

\[
\{ y > x \} \implies \{ z < 0 \}
\]
Some examples

\[ \text{while } x > 0 \]
\[ z = x \times 2 + y \]
\[ x = x / 2 \]
\[ z = x \times 2 - y \]

: \{ y > x \} \Rightarrow \{ z < 0 \}

Is it valid?
Some examples

while $x > 0$
  $z = x \times 2 + y$
  $x = x / 2$
  $z = x \times 2 - y$

: $\{y > x\} \implies \{z < 0\}$
Some examples

\[
\begin{align*}
\text{while } x > 0 & \\
\quad z &= x \times 2 + y \\
\quad x &= x / 2 \\
\quad z &= x \times 2 - y
\end{align*}
\]

\[
\{ y > x \} \implies \{ z < 0 \}
\]

Is it valid?
Some examples

\[ z = x \cdot 2 + y \]
\[ x = \frac{x}{2} \]
\[ z = x \cdot 2 - y \]

: \{\text{even } y \land \text{odd } x\} \Rightarrow \{z < \sqrt{2.5}\}
Some examples

\[
\begin{align*}
z &= x \times 2 + y \\
x &= x / 2 \\
z &= x \times 2 - y
\end{align*}
\]

: \{\text{even } y \wedge \text{ odd } x\} \Rightarrow \{z < \sqrt{2.5}\}

Is it valid?
How do we determine the validity of an Hoare triple?
Validity of Hoare triple

$c : P \Rightarrow Q$

Precondition
(a logical formula)

Postcondition
(a logical formula)

Program
Validity of Hoare triple

We are interested only in inputs that meet $P$ and we want to have outputs satisfying $Q$.

$c : P \Rightarrow Q$

Program

Precondition (a logical formula)

Postcondition (a logical formula)
Validity of Hoare triple

We are interested only in inputs that meets $P$ and we want to have outputs satisfying $Q$.

How shall we formalize this intuition?

$P \Rightarrow Q$
Validity of Hoare triple

We say that the triple $c : P \Rightarrow Q$ is valid if and only if for every memory $m$ such that $P(m)$ and memory $m'$ such that $\{c\}_m = m'$ we have $Q(m')$. 
Validity of Hoare triple

We say that the triple $c : P \Rightarrow Q$ is valid if and only if for every memory $m$ such that $P(m)$ and memory $m'$ such that $\{c\}_m = m'$ we have $Q(m')$.

Is this condition easy to check?
Hoare Logic
Floyd-Hoare reasoning

A verification of an interpretation of a flowchart is a proof that for every command \( c \) of the flowchart, if control should enter the command by an entrance \( a_i \) with \( P_i \) true, then control must leave the command, if at all, by an exit \( b_j \) with \( Q_j \) true. A semantic definition of a particular set of command types, then, is a rule for constructing, for any command \( c \) of one of these types, a verification condition \( V_c(P; Q) \) on the antecedents and consequents of \( c \). This verification condition must be so constructed that a proof that the verification condition is satisfied for the antecedents and consequents of each command in a flowchart is a verification of the interpreted flowchart.
Rules of Hoare Logic

Skip

\[ \vdash \text{skip} : P \Rightarrow P \]
Rules of Hoare Logic

Composition

\[ \vdash c; c' : \quad \text{P} \Rightarrow Q \]
Rules of Hoare Logic Composition

\[ \vdash c ; c' : P \Rightarrow Q \]
Rules of Hoare Logic

Composition

\[ \vdash c ; c' : P \Rightarrow Q \]

\[ \vdash c : P \Rightarrow R \]

\[ \vdash c ; c' : P \Rightarrow Q \]
Rules of Hoare Logic

Composition

\[ \vdash c; c': P \Rightarrow Q \]

\[ \vdash c: P \Rightarrow R \quad \vdash c': R \Rightarrow Q \]

\[ \vdash c; c': P \Rightarrow Q \]
Rules of Hoare Logic

If then else

\[ \frac{\vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q}{\vdash} \]
Rules of Hoare Logic

If then else

\[ \vdash c_1 : P \Rightarrow Q \]

\[ \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q \]
Rules of Hoare Logic
If then else

\[ \frac{\vdash c_1 : P \Rightarrow Q \quad \vdash c_2 : P \Rightarrow Q}{\vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q} \]
Is this correct?
Rules of Hoare Logic

If then else

\[ \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q \]
Rules of Hoare Logic

If then else

\[\Gamma \vdash e \land P \Rightarrow Q\]

\[\vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q\]
Rules of Hoare Logic

If then else

\[ \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q \]

\[ \vdash c_1 : e \land P \Rightarrow Q \]
\[ \vdash c_2 : \neg e \land P \Rightarrow Q \]

\[ \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : P \Rightarrow Q \]
Rules of Hoare Logic

Assignment

\[ \vdash x := e : P \Rightarrow P[e/x] \]
Rules of Hoare Logic

Assignment

$$\Gamma, x : e : P \Rightarrow P[ e / x ]$$

Is this correct?
Rules of Hoare Logic

Assignment

\[ \vdash x := e : P[e/x] \Rightarrow P \]
Rules of Hoare Logic

While

\[ \vdash \text{while } e \text{ do } c : P \Rightarrow P \land \neg e \]
Rules of Hoare Logic

\( \vdash c : e \land P \Rightarrow P \)

\( \vdash \text{while } e \text{ do } c : \neg e \land P \Rightarrow P \land P \land \neg e \)
How do we know that these are the right rules?
Correctness of a rule

\[ \vdash c : P \Rightarrow Q \]
Correctness of a rule

\[ \vdash c : P \Rightarrow Q \]

\[ \vdash c' : R \Rightarrow S \]

\[ \vdash c : P \Rightarrow Q \]
Correctness of a rule

\[ \vdash c' : R \Rightarrow S \]

\[ \vdash c : P \Rightarrow Q \]

We say that a rule is correct if given a valid triple as described by the assumption(s), we can prove the validity of the triple in the conclusion.