
Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu
From the previous class
Does the program comply with the specification?

**Precondition:** \( x \geq 0 \) and \( y \geq 0 \)

**Function** Add\((x: \text{ int}, y: \text{ int}) : \text{ int} \) 

```
{ 
    r = 0;
    n = y;
    while n != 0 
        { 
            r = r + 1;
            n = n - 1;
        } 
    return r
}
```

**Postcondition:** \( r = x + y \)
Does the program comply with the specification?

Precondition: $x \geq 0$ and $y \geq 0$

Function Add($x$: int, $y$: int) : int
{
    $r = 0$;
    $n = y$;
    while $n != 0$
    {
        $r = r + 1$;
        $n = n - 1$;
    }
    return $r$
}

Postcondition: $r = x + y$
Precondition: \( x \geq 0 \) and \( y \geq 0 \)

Function \( \text{Add}(x: \text{int}, y: \text{int}) : \text{int} \)

\[
\begin{aligned}
    & \text{r} = x; \\
    & n = y; \\
    & \text{while } n \neq 0 \\
    & \quad \{ \\
    & \quad \quad r = r + 1; \\
    & \quad \quad n = n - 1; \\
    & \quad \} \\
    & \text{return } r \\
\end{aligned}
\]

Postcondition: \( r = x + y \)
How about this one?

Precondition: $x \geq 0$ and $y \geq 0$

Function Add($x$: int, $y$: int) : int
{
    $r = x$;
    $n = y$;
    while $n \neq 0$
    {
        $r = r + 1$;
        $n = n - 1$;
    }
    return $r$
}

Postcondition: $r = x + y$
How can we make this reasoning mathematically precise?
We need to assign a formal meaning to the different components:
Formal Semantics

We need to assign a formal meaning to the different components:

- Precondition
- Program
- Postcondition

formal semantics of programs
Formal Semantics

We need to assign a formal meaning to the different components:

- **Precondition**
- **Program**
- **Postcondition**

- formal semantics of specification conditions
- formal semantics of programs
- formal semantics of specification conditions
We need to assign a formal meaning to the different components:

- Precondition
- Program
- Postcondition

We also need to describe the rules which combine program and specifications.

Formal semantics of programs
Formal semantics of specification conditions
Formal semantics of specification conditions
Goal for today

• Formalize the semantics of a simple imperative programming language.
A first example

**FastExponentiation**\((n, k : \text{Nat}) : \text{Nat}\)

\[
n' \:= n; \quad k' \:= k; \quad r \:= 1; \\
\text{if } k' > 0 \text{ then} \\
\quad \text{while } k' > 1 \text{ do} \\
\quad \quad \text{if even}(k') \text{ then} \\
\quad \quad \quad n' \:= n' \times n'; \\
\quad \quad \quad k' \:= k'/2; \\
\quad \quad \text{else} \\
\quad \quad \quad r \:= n' \times r; \\
\quad \quad \quad n' \:= n' \times n'; \\
\quad \quad \quad k' \:= (k' - 1)/2; \\
\quad r \:= n' \times r; \\
(* \text{result is } r *)
\]
Programming Language

c ::= abort
   │ skip
   │ x := e
   │ c ; c
   │ if e then c else c
   │ while e do c

x, y, z, ... program variables

e₁, e₂, ... expressions

c₁, c₂, ... commands
How would you describe the meaning of a program in a mathematically precise way?
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

\[ x + 5 \quad x \mod k \quad x[i] \quad (x[i+1] \mod 4) + 5 \]
Types

In expressions we want to be able to use “arbitrary” data types.

\[
t ::= b \\
   \mid T(t_1, ..., t_n)
\]
Types

In expressions we want to be able to use “arbitrary” data types.

\[
t ::= b \quad | \quad T(t_1, \ldots, t_n)
\]

We assume a collection of base types \(b\) including

- Bool
- Int
- Nat
- String

We also assume a set of type constructors \(T\) that we can use to build more complex types, such as:

- Bool list
- Int*Bool
- Int*String \(\rightarrow\) Bool
Types

We also use types to guarantee that commands are well-formed.

For example, in the commands

```
while e do c           if e then c_1 else c_2
```

We require that $e$ is of type $\text{Bool}$. 
Types

We also use types to guarantee that commands are well-formed.

For example, in the commands

```
while e do c  
if e then c₁ else c₂
```

We require that `e` is of type `Bool`.

We omit the details of the type system here but you can find them in the notes by Gilles Barthe.
Values

Values are atomic expressions whose semantics is self-evident and which do not need a further analysis.

For example, we have the following values:

- true
- 5
- [1, 2, 3, 4]
- “Hello”

The following are not values:

- not true
- x+5
- [x, x+1]
- x[1]
Values

Values are atomic expressions whose semantics is self-evident and which do not need a further analysis.

For example, we have the following values

```
true 5 [1,2,3,4] “Hello”
```

The following are not values:

```
not true x+5 [x,x+1] x[1]
```

We could define a grammar for values, but we prefer to leave this at the intuitive level for now.
How can we give semantics to expressions and commands?

**FastExponentiation** ($n, k : \text{Nat}) : \text{Nat}$

```
n' := n; k' := k; r := 1;
if k' > 0 then
  while k' > 1 do
    if even(k') then
      n' := n' * n';
      k' := k'/2;
    else
      r := n' * r;
      n' := n' * n';
      k' := (k' - 1)/2;
  r := n' * r;
(* result is r *)
```
Memories

We can formalize a memory as a map $m$ from variables to values.

$$m = [x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$$

We consider only maps that respect types.
Memories

We can formalize a memory as a map $m$ from variables to values.

$$m = [x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$$

We consider only maps that respect types.

We want to read the value associated to a particular variable:

$$m(x)$$

We want to update the value associated to a particular variable:

$$m[x\leftarrow v]$$

This is defined as

$$m[x\leftarrow v](y) = \begin{cases} v & \text{if } x=y \\ m(y) & \text{Otherwise} \end{cases}$$
Semantics of Expressions

What is the meaning of the following expressions?

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

What is the meaning of the following expressions?
\[ x + 5 \quad \text{x mod } k \quad x[i] \quad (x[i+1] \mod 4) + 5 \]

We can give the semantics as a relation between expressions, memories and values.

\[
\text{Exp} \ast \text{Mem} \ast \text{Val}
\]

We will denote this relation as:

\[
\{ e \}_m = v
\]
Semantics of Expressions

What is the meaning of the following expressions?

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

We can give the semantics as a relation between expressions, memories and values.

\[ \text{Exp} \times \text{Mem} \times \text{Val} \]

We will denote this relation as:

\[ \{ e \}_m = v \]

This is commonly typeset as:

\[ [e]_m = v \]
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.
Suppose we have a memory

\[ m = [i \mapsto 1, x \mapsto [1, 2, 3], y \mapsto 2] \]

That \{\text{mod}\} is the modulo operation and \{+\} is addition, we can derive the meaning of the following expression:
Semantics of Expressions

Suppose we have a memory

\[ m = [i \mapsto 1, x \mapsto [1, 2, 3], y \mapsto 2] \]

That \{\text{mod}\} is the modulo operation and \{\text{+}\} is addition, we can derive the meaning of the following expression:

\[ \{(x[i+1] \mod y) + 5\}_m \]
Semantics of Expressions

Suppose we have a memory

\[ m = [i \mapsto 1, x \mapsto [1, 2, 3], y \mapsto 2] \]

That \{\text{mod}\} is the modulo operation and \{+\} is addition, we can derive the meaning of the following expression:

\[ \{(x[i+1] \mod y) + 5\}_m = \{(x[i+1] \mod y)\}_m + \{5\}_m \]
Semantics of Expressions

Suppose we have a memory

$$m = [i \mapsto 1, x \mapsto [1, 2, 3], y \mapsto 2]$$

That \{mod\} is the modulo operation and \{+\} is addition, we can derive the meaning of the following expression:

$$\{ (x[i+1] \mod y) + 5 \}_m$$

$$= \{ (x[i+1] \mod y) \}_m \{+\} \{5\}_m$$

$$= (\{x[i+1]\}_m \{\text{mod}\} \{y\}_m) \{+\} \{5\}_m$$
Semantics of Expressions

Suppose we have a memory

\[ m = [i \mapsto 1, x \mapsto [1, 2, 3], y \mapsto 2] \]

That \{\text{mod}\} is the modulo operation and \{+\} is addition, we can derive the meaning of the following expression:

\[
\{(x[i+1] \mod y) + 5\}_m
\]

\[
= \{(x[i+1] \mod y)\}_m\{+\}\{5\}_m
\]

\[
= (\{x[i+1]\}_m \{\text{mod}\} \{y\}_m)\{+\}\{5\}_m
\]

\[
= (\{x\}_m[\{i\}_m\{+\}\{1\}_m] \{\text{mod}\} \{y\}_m)\{+\}\{5\}_m
\]
Semantics of Expressions

Suppose we have a memory

\[ m = [i \rightarrow 1, x \rightarrow [1, 2, 3], y \rightarrow 2] \]

That \{mod\} is the modulo operation and \{+\} is addition, we can derive the meaning of the following expression:

\[ \{(x[i+1] \mod y) + 5\}_m \]

\[ = \{(x[i+1] \mod y)\}_m \{+\} \{5\}_m \]

\[ = (\{x[i+1]\}_m \{\mod\} \{y\}_m) \{+\} \{5\}_m \]

\[ = (\{x\}_m[\{i\}_m\{+\} \{1\}_m] \{\mod\} \{y\}_m) \{+\} \{5\}_m \]

\[ = (\{x\}_m[1\{+\}1] \{\mod\} 2) \{+\} 5 \]
Semantics of Expressions

Suppose we have a memory

\[
m = \{ i \mapsto 1, x \mapsto [1, 2, 3], y \mapsto 2 \}\]

That \{\mod\} is the modulo operation and \{+\} is addition, we can derive the meaning of the following expression:

\[
\{(x[i+1] \mod y) + 5\}_m
\]

\[
= \{(x[i+1] \mod y)\}_m \{+\} \{5\}_m
\]

\[
= (\{x[i+1]\}_m \{\mod\} \{y\}_m) \{+\} \{5\}_m
\]

\[
= (\{x\}_m[\{i\}_m\{+\}\{1\}_m] \{\mod\} \{y\}_m) \{+\} \{5\}_m
\]

\[
= (\{x\}_m[1\{+\}1] \{\mod\} 2) \{+\} 5
\]

\[
= (\{x\}_m[2] \{\mod\} 2) \{+\} 5
\]
Semantics of Expressions

Suppose we have a memory

\[ m = \{ i \mapsto 1, x \mapsto [1, 2, 3], y \mapsto 2 \} \]

That \{ \text{mod} \} is the modulo operation and \{ + \} is addition, we can derive the meaning of the following expression:

\[
\{(x[i+1] \mod y) + 5\}_m
\]

\[
= \{(x[i+1] \mod y)\}_m\{+\}\{5\}_m
\]

\[
= (\{x[i+1]\}_m \{\text{mod}\} \{y\}_m)\{+\}\{5\}_m
\]

\[
= (\{x\}_m[\{i\}_m\{+\}\{1\}_m] \{\text{mod}\} \{y\}_m)\{+\}\{5\}_m
\]

\[
= (\{x\}_m[1\{+\}1] \{\text{mod}\} 2)\{+\}5
\]

\[
= (\{x\}_m[2] \{\text{mod}\} 2)\{+\}5
\]

\[
= (2 \{\text{mod}\} 2)\{+\}5 = 0 \{+\} 5 = 5
\]
Operational vs Denotational Semantics

The style of semantics we are using is denotational, in the sense that we describe the meaning of an expression by means of the value it denotes.

A different approach, more operational in nature, would be to describe the meaning of an expression by means of the value that the expression evaluates to in an abstract machine.
Semantics of Commands

What is the meaning of the following command?

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

What is the meaning of the following command?

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

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

\[ \text{Exp} \ast \text{Mem} \ast \text{Mem} \]
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.

\[ \text{Exp} \times \text{Mem} \times \text{Mem} \]

Would this work?
Semantics of Commands

What is the meaning of the following command?

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

\[ \text{Exp} \ast \text{Mem} \ast (\text{Mem} \lor \bot) \]

We will denote this relation as:

\[ \{ c \}_{m} = m' \quad \text{Or} \quad \{ c \}_{m} = \bot \]
Semantics of Commands

What is the meaning of the following command?

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

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

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

We will denote this relation as:

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

This is commonly typeset as:

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

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

This is defined on the structure of commands:

$$\{\text{abort}\}_m = \bot$$
Semantics of Commands

This is defined on the structure of commands:

\[ \{\text{abort}\}_m = \perp \]
\[ \{\text{skip}\}_m = m \]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \bot \\
\{\text{skip}\}_m &= m \\
\{x:=e\}_m &= m[x \leftarrow \{e\}_m]
\end{align*}
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \bot \\
\{\text{skip}\}_m &= m \\
\{x:=e\}_m &= m[x\leftarrow\{e\}_m] \\
\{c; c'\}_m &= \{c'\}_m', \quad \text{if} \quad \{c\}_m = m'
\end{align*}
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \bot \\
\{\text{skip}\}_m &= m \\
\{x := e\}_m &= m[x \leftarrow \{e\}_m] \\
\{c; c'\}_m &= \{c'\}_{m'} \quad \text{if} \quad \{c\}_m = m' \\
\{c; c'\}_m &= \bot \quad \text{if} \quad \{c\}_m = \bot
\end{align*}
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\{\text{abort}\}_m = \bot \\
\{\text{skip}\}_m = m \\
\{x := e\}_m = m[x \leftarrow \{e\}_m] \\
\{c; c'\}_m = \{c'\}_{m'} \quad \text{if} \quad \{c\}_m = m' \\
\{c; c'\}_m = \bot \quad \text{if} \quad \{c\}_m = \bot \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_t\}_m \quad \text{if} \quad \{e\}_m = \text{true}
\]
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

What about while
Semantics of While

What about while

\{\text{while } e \text{ do } c\}_m = ???
Semantics of While

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

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

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

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

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

\[ m_1 \quad m_2 \quad m_3 \quad \cdots \quad m_n \quad \cdots \]

\[ \geq \quad \geq \quad \geq \quad \cdots \quad \geq \]

Dana Scott
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:

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

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:

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

Will this work?

We are missing the base case.
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:

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

where

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

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \bot \\
\{\text{skip}\}_m &= m \\
\{x:=e\}_m &= m[x\leftarrow\{e\}_m] \\
\{c; c'\}_m &= \{c'\}_m', \quad \text{if} \quad \{c\}_m = m' \\
\{c; c'\}_m &= \bot, \quad \text{if} \quad \{c\}_m = \bot \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_t\}_m, \quad \text{if} \quad \{e\}_m = \text{true} \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_f\}_m, \quad \text{if} \quad \{e\}_m = \text{false} \\
\{\text{while } e \text{ do } c\}_m &= \sup_{n \in \mathbb{N}} \{\text{while}_n e \text{ do } c\}_m \\
\text{where} \\
\text{while}_n e \text{ do } c &= \text{while}_n e \text{ do } c; \text{if } e \text{ then } \text{abort} \text{ else } \text{skip} \\
\text{and} \\
\text{while}_0 e \text{ do } c &= \text{skip} \\
\text{while}_{n+1} e \text{ do } c &= \text{if } e \text{ then } (c; \text{while}_n e \text{ do } c) \text{ else } \text{skip}
\end{align*}
\]
Example

What is the semantics of the following program:

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


What is the semantics of the following program:

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

Example