**Goal:** Software without bugs.

We have seen:

→ how to reason about mathematical properties in First-Order Logic (FOL)
**Goal:** Software without bugs.

We have seen:

→ how to reason about mathematical properties in First-Order Logic (FOL)

From now on:

→ **how to reason about simple imperative programs**

→ FOL for specifications

→ how to reason about properties (e.g., termination) of recursive functional programs using:
  → FOL natural deduction proofs
  → Well-founded induction
  → Case analysis on data types (natural numbers, lists, etc.)
  → variants (when reasoning about termination)

```plaintext
y := 1;
z := 0;
while (z != x) {
    z := z + 1;
y := y * z;
}
```
The goal is to write specifications such as:

→ “Program $P$ computes a number $y$ whose square is less than the input $x$”
→ “Program $P$ is a program such that at the end of $P$, array of integers $a$ contains numbers in increasing order”
The goal is to write specifications such as:

→ “Program $P$ computes a number $y$ whose square is less than the input $x$”
→ “Program $P$ is a program such that at the end of $P$, array of integers $a$ contains numbers in increasing order”

We will do this by using the following logical formalism combining programs with logical specifications.

$$((F) \ P \ (G))$$

→ $(x > 0) \ P \ (y \ y < x)$
→ $(\top) \ P \ (\forall i.((0 \leq i < |a| - 1) \rightarrow a[i] \leq a[i + 1]))$
Hoare triple: \((F) P (G)\)

- \(F\) and \(G\) are FOL formulas.
  - Terms: integers, booleans, sequences and their operations (we will be more precise when needed).
  - Predicates: all standard arithmetic predicates about integers, booleans, and their operations.
  - \(F\) is called **precondition**
  - \(G\) is called **postcondition**

- \(P\) is a program written in an imperative programming language, which has:
  - a state* which is a function \(l\) mapping any variable name \(x\) to an integer \(l(x)\)
  - a given grammar.

- **state \(l\) satisfies \(F\) (\(l\) is a \(F\)-state), written \(l \models F\)** when
  - \(\mathcal{M} \models_l F\), for some **standard model** \(\mathcal{M}\) (contains all integers and interprets terms and predicates in the “standard way”)

- quantifiers in \(F\) and \(G\) contain variables not occurring in \(P\).

* A state is very similar to the **environment** in FOL semantics
for any state $l$ such that $l(x) = -2$ and $l(y) = 5$ and $l(z) = -1$:

$\rightarrow l \models \neg(x + y < z)$ holds

$\rightarrow l \models y - x \cdot z < z$ does not hold
Arithmetic Expressions:  \[ E ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \mid -1 \mid -2 \mid -3 \mid \ldots \]
\[ \mid x \mid (\neg E) \mid (E + E) \mid (E - E) \mid (E \ast E) \]
\[ \mid \text{if } B \text{ then } E \text{ else } E \mid S[E] \mid |S| \]

Sequence Expressions:  \[ S ::= s \mid S[E] \mid S[E..] \mid S[E..E] \]

Boolean Expressions:  \[ B ::= \text{true} \mid \text{false} \mid (\neg B) \mid (B \& B) \mid (B || B) \]
\[ \mid (E < E) \mid (E > E) \mid (E = E) \]

Commands:  \[ C ::= (x := E) \mid C; C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \{ C \} \]

The book has a simpler language. Here (and in assignments/exams) we will use this richer language.

→ Binding precedence for arithmetic expressions:
  → Negation (\neg E) binds more tightly than
  → multiplication (E_1 \ast E_2) which binds more tightly than
  → subtraction (E_1 \ast E_2) and addition (E_1 + E_2)

→ Binding precedence for boolean expressions:
  → Nation (\neg E) binds more tightly than
  → conjunction (E_1 \& E_2) disjunction (E_1 || E_2).
Arithmetic Expressions:  \( E ::= 0 \mid 1 \mid 2 \mid 3 \mid \ldots \mid -1 \mid -2 \mid -3 \mid \ldots \mid x \mid (-E) \mid (E + E) \mid (E - E) \mid (E \times E) \mid \text{if } B \text{ then } E \text{ else } E \mid S[E] \mid |S| \)

Sequence Expressions:  \( S ::= s \mid S[..E] \mid S[E..] \mid S[E..E] \)

Boolean Expressions:  \( B ::= \text{true} \mid \text{false} \mid (!B) \mid (B \& B) \mid (B \lor B) \mid (E < E) \mid (E > E) \mid (E = E) \)

Commands:  \( C ::= (x := E) \mid C; C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \{C\} \)

- There is a 1-1 correspondence between arithmetic/boolean expressions, and terms/formulas of the FOL we use to write specifications.
  - the program expression \(! (x < y)\) corresponds to the FOL formula \( \neg (x < y) \)
  - the program expression \(! (x < z) \land (y < z)\) corresponds to the FOL formula \( \neg (x < z) \land (y < z) \)
  - what program expression corresponds to the FOL formula \( (x = y)\)?
Proof rules: conditional

\((F) \ C \ (G)\)

High-level meaning of a Hoare triple: if we execute \(C\) in any state \(l\) that satisfies \(F\), \([\ldots]^{(\text{next slides})}\) the final state will satisfy \(G\).

\(l\) satisfies \(F\) : \(l \models F\)

\(l\) maps variables to integers.

Are the following true?

\[\rightarrow \ {x \mapsto 1, \ y \mapsto 2} \models x < y\]

\[\rightarrow \ {x \mapsto 1, \ y \mapsto 2} \models x > y\]

\[\rightarrow \ {x \mapsto 1, \ y \mapsto 2} \models (x + 1) = y\]
⊢ \text{par} (|F|) C (|G|)

Meaning of partial Hoare triple: if we execute $C$ in any state $l$ that satisfies $F$, and if $C$ terminates, then the final state will satisfy $G$.

Equivalently: for all $l$ such that $l \models F$, if $l, C \downarrow l'$ then for the final state $l'$ we have $l' \models G$.

→ **Correctness:** the pre- and post-conditions $F$ and $G$ give a *specification* of the program

→ **Partial:** the above statement does not guarantee that $C$ will terminate (which is a part of its correct operation)

→ An infinite loop satisfies all pairs of pre-/post-conditions.

```
while (true) { x := 0 }
```
\[ \vdash_{\text{tot}} (|F|) \ C \ (|G|) \]

Meaning of total Hoare triple: if we execute \( C \) in any state \( l \) that satisfies \( F \), then \( C \) terminates and the final state will satisfy \( G \).

Equivalently: for all \( l \) such that \( l \models F \) we have \( l, C \downarrow l' \) and for the final state \( l' \) we have \( l' \models G \).

→ **Correctness:** the pre- and post-conditions \( F \) and \( G \) give a specification of the program

→ **Total:** the above statement **does** guarantee that \( C \) will terminate

**Q:** are the following statements correct?

→ If \( \vdash_{\text{tot}} (|F|) \ C \ (|G|) \) holds then \( \vdash_{\text{par}} (|F|) \ C \ (|G|) \) holds.

→ If \( \vdash_{\text{par}} (|F|) \ C \ (|G|) \) holds then \( \vdash_{\text{tot}} (|F|) \ C \ (|G|) \) holds.
$\vdash_{\text{tot}} (\downarrow F) \text{ C } (\downarrow G)$

Meaning of total Hoare triple: if we execute $C$ in any state $l$ that satisfies $F$, then $C$ terminates and the final state will satisfy $G$.

Equivalently: for all $l$ such that $l \models F$ we have $l, C \Downarrow l'$ and for the final state $l'$ we have $l' \models G$.

→ **Correctness**: the pre- and post-conditions $F$ and $G$ give a specification of the program

→ **Total**: the above statement does guarantee that $C$ will terminate

**Q**: are the following statements correct?

→ If $\vdash_{\text{tot}} (\downarrow F) \text{ C } (\downarrow G)$ holds then $\vdash_{\text{par}} (\downarrow F) \text{ C } (\downarrow G)$ holds. **Yes**

→ If $\vdash_{\text{par}} (\downarrow F) \text{ C } (\downarrow G)$ holds then $\vdash_{\text{tot}} (\downarrow F) \text{ C } (\downarrow G)$ holds. **No**
Consider the imperative factorial function:

```c
y := 1;
z := 0;
while (z != x) {
    z := z + 1;
y := y * z;
}
```

Should we be able to prove the following?

\[ \vdash \text{tot}(x \geq 0) \ C (\mathcal{F}) \]
\[ \vdash \text{par}(x \geq 0) \ C (\mathcal{F}) \]
\[ \vdash \text{tot}(\top) \ C (\mathcal{F}) \]
\[ \vdash \text{par}(\top) \ C (\mathcal{F}) \]

where \( \mathcal{F} \) is a formula, not specified here.
Consider the imperative factorial function:

\[
\begin{align*}
y & := 1; \\
z & := 0; \\
\text{while} (z \neq x) \{ \\
& \quad z := z + 1; \\
& \quad y := y \times z; \\
& \}
\end{align*}
\]

Should we be able to prove the following?

\[
\begin{align*}
\to & \quad \vdash \text{tot} (|x \geq 0|) \ C \ (|F|) \\
\to & \quad \vdash \text{par} (|x \geq 0|) \ C \ (|F|) \\
\to & \quad \vdash \text{tot} (|\top|) \ C \ (|F|) \quad \textbf{No:} \text{ does not terminate in starting states with } x < 0 \\
\to & \quad \vdash \text{par} (|\top|) \ C \ (|F|)
\end{align*}
\]

where \( F \) is a formula, not specified here.
Consider the imperative factorial function:

```
y := 1;
z := 0;
while (z != x) {
    z := z + 1;
y := y * z;
}
```

Specification: \( \vdash_{\text{par}} (|x \geq 0|) \ C (|F|) \)

What is the right postcondition \( F \) for the above code?
Consider the imperative factorial function:

```plaintext
y := 1;
z := 0;
while (z != x) {
    z := z + 1;
    y := y * z;
}
```

Specification: \(\vdash_{\text{par}} (|x \geq 0|) \ C (|F|)

What is the right postcondition \(F\) for the above code?

\(\vdash_{\text{par}} (|x \geq 0|) \ C (|y = x!|)\)
What is the right post-condition for this version of factorial?

```plaintext
y := 1;
while (x != 0) {
    y := y * x;
    x := x - 1;
}
```

†Dafny calls them ghost variables
What is the right post-condition for this version of factorial?

\[
y := 1;
\text{while } (x \neq 0) \{
    y := y \times x;
    x := x - 1;
\}
\]

\[\vdash_{\text{par}} ((x = x_0) \land (x \geq 0)) \Rightarrow (y = x_0!)]\]

→ **logical variables**: variable \(x_0\) is used only in formulas to “remember” some value from the starting state.

→ **program variables**: variables used by the program

†Dafny calls them **ghost variables**
\[ \vdash_{\text{AR}} F' \rightarrow F \quad (F) \quad C \quad (G) \quad \vdash_{\text{AR}} G \rightarrow G' \quad \implies
\]

\[ (F') \quad C \quad (G') \]

\[ \vdash_{\text{AR}} F' \rightarrow F \] means that the implication is derivable in FOL with natural numbers, equality, standard predicates etc., when all known properties of arithmetic are in our assumptions.
\[
\text{PROOF RULES: ASSIGNMENT}
\]

\[
(G[E/x]) x = E (G) \\
\text{Asg}
\]
PROOF RULES: CONDITIONAL

\[
\begin{align*}
(F \land B) & \ C \ (G) \quad (F \land \neg B) & \ C_2 \ (G) \\
\hline
\text{if } B \text{ then } C_1 \text{ else } C_2 \ (G) & \quad \text{COND}
\end{align*}
\]
PROOF RULES: SEQUENCE

\[
\frac{(F) \ C_1 \ (\eta) \ \ (\eta) \ C_2 \ (G)}{(F) \ C_1; \ C_2 \ (G)} \quad \text{COMP}
\]
Prove the following Hoare triples:

→ \( (y > 0) \ x = y + 1 \ (x > 0) \)
→ \( (x \geq y) \ x = x - y \ (x \geq 0) \)
→ \( (x \geq y) \ x = x - y; \ y = -x \ (y \leq 0) \)
→ Swap without temp:
   \( (x = x_0) \land (y = y_0) \ x = y - x; \ y = y - x; \ x = x + y \ (x = y_0) \land (y = x_0) \)
→ \( (\top) \ \text{if } x < 2 \text{ then } x = 2 \text{ else } x = x \ (x \geq 2) \)