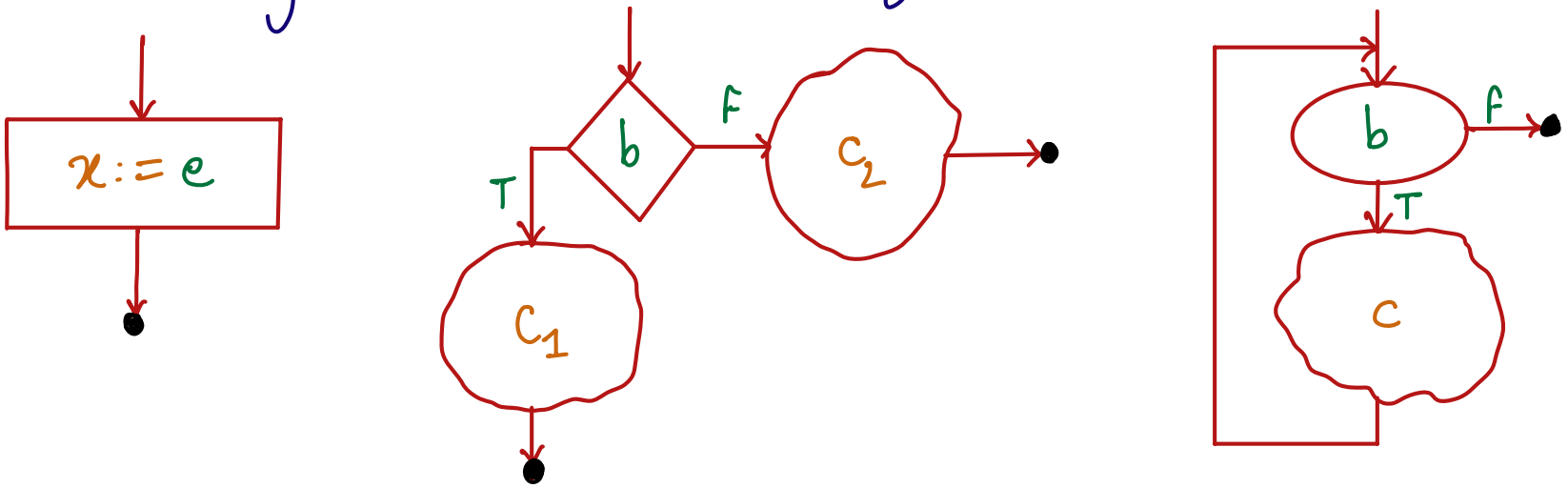
Hoare Logic

Recall our syntax for commands in the imperative language:

$c ::= \text{skip} \mid x := e \mid c_1 ; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \text{ end}$

Can think of these as blocks in a flowchart.



Interpretation of a flowchart: Map from edges to propositions

We can combine blocks to form larger flowcharts

Can we also compose the propositions along the edges to analyze the resultant complex program?

Recall the following program:

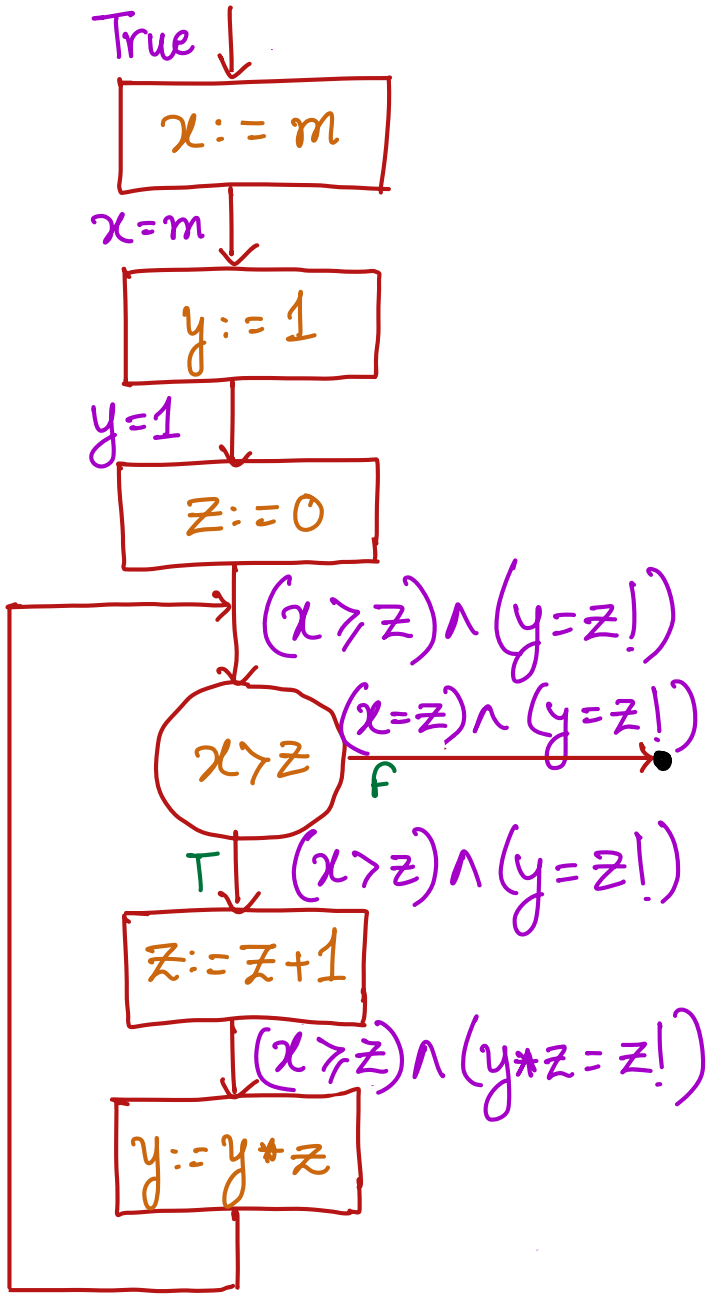
c_1 : $x := m ; \; y := 1 ; \; z := 0 ;$

w :
```
while (x > z) do
    z := z + 1;   y := y * z;
end
```

We proved, by analyzing the various state transformations effected by this program, that it computed the factorial of x and stored it in y.

This needed a non-trivial induction.

Can we simplify such analysis?



A **Hoare triple** is of the form $\{\alpha\}\, c\, \{\beta\}$, where c is a command, and α and β are **assertions**. — First-order logic formulae

α: Precondition

β: Postcondition

Terms $t_1, t_2 := x \mid v \mid n \mid t_1 + t_2 \mid t_1 * t_2$

x: first-order variable

v: Program identifier

Assertions $\alpha, \beta := eq(t_1, t_2) \mid gt(t_1, t_2) \mid \alpha \wedge \beta \mid \alpha \vee \beta \mid \neg\alpha \mid \alpha \to \beta \mid \exists v.\, \alpha \mid \text{True} \mid \text{False}$

Informally, the triple $\{\alpha\}\, c\, \{\beta\}$ means that whenever one runs the command c starting in a state which satisfies the formula α, **if the execution terminates**, one ends up in a state which satisfies the formula β.

One can reason about these triples directly, based on program structure, using the following rules.

$$\frac{}{\{\alpha\}\,\mathrm{skip}\,\{\alpha\}}\ \text{Hskip}$$

$$\frac{}{\{\alpha(e)\}\,(x := e)\,\{\alpha(x)\}}\ \text{Hasgn}$$

$\alpha(e)$ obtained by replacing all occurrences (if any) of x in α by e.

$$\frac{\{\alpha\}\,c_1\,\{\beta'\} \qquad \{\beta'\}\,c_2\,\{\beta\}}{\{\alpha\}\,(c_1; c_2)\,\{\beta\}}\ \text{Hseq}$$

$$\frac{\models \alpha' \to \alpha \qquad \{\alpha\}\,c\,\{\beta\} \qquad \models \beta \to \beta'}{\{\alpha'\}\,c\,\{\beta'\}}\ \text{Con}$$

$\models \varphi$ means φ is a **valid** assertion, i.e. $s \models \varphi$ for all states s

$$\frac{\{\alpha \wedge b\}\,c\,\{\beta\} \qquad \{\alpha \wedge \neg b\}\,c'\,\{\beta\}}{\{\alpha\}\,(\text{if } b \text{ then } c \text{ else } c')\,\{\beta\}}\ \text{If}$$

$$\frac{\{b \wedge i\}\,c\,\{i\}}{\{i\}\,(\text{while } b \text{ do } c \text{ end})\,\{i \wedge \neg b\}}\ \text{While}$$

i is a **loop invariant**
If i is true before the loop, it is true after an iteration too

Back to our earlier example.

$x := m \; ; \; y := 1 \; ; \; z := 0$

We start with a state where the only assertion that holds is True.

We would like to prove $\{True\}(x := m)\{eq(x, m)\}$.

The only rule we have for handling an assignment command is Hasgn.

But Hasgn works with precondition $\alpha(e)$, command $(x := e)$, and postcondition $\alpha(x)$.

Suppose $\alpha(x)$ is $eq(x, m)$. Then, since the command is $(x := m)$, $\alpha(e)$ must be $eq(m, m)$.

So, we can prove $\{eq(m, m)\}\,(x := m)\,\{eq(x, m)\}$.

But to prove $\{True\}(x := m)\{eq(x, m)\}$, we can use the Con rule, since $True \rightarrow eq(m, m)$.

Exercise: Prove that we can derive, using these rules, the following Hoare triple about our factorial program.

$$\{\text{True}\}\ c_1; w\ \{x = m; y = m!; z = m\}$$

$$\frac{}{\{\alpha(e)\}(x := e)\{\alpha(x)\}}\ \text{Hasgn}$$

This is a "backward assignment" rule.

Start from a state s' where $\alpha(x)$ holds, and work backwards.

We need to reach s' by executing $x := e$.

What is the easiest way to ensure that assigning e to x gets us to a state where $\alpha(x)$ holds?

Make sure that α held of the expression e to begin with!

⇓

Start in a state where $\alpha(e)$ holds.

Can we come up with some reasonable "forward assignment" rule?

$$\frac{}{\{\top\}\ (x := e)\ \{eq(x, e)\}}\ \text{HasgnF}_1$$

Is this a valid rule?

We are implicitly quantifying over all expressions e.

To get a valid forward assignment rule,
one has to somehow "remember" the value of the program identifier x
in the original state, evaluate the expression in the assign command
using this value, and then assign this resultant value to x.

Exercise:
Show that the following is a valid assignment rule.

$$\{\alpha\}\,(x := e)\,\{\exists x_0 : \alpha\left({}^{x_0}/_{x}\right) \wedge x = e\left({}^{x_0}/_{x}\right)\}$$

We say that a Hoare triple is valid (denoted $\models \{\alpha\}\, c\, \{\beta\}$) if for all states s, s', if $s \models \alpha$ and $s -[c]\rightarrow s'$, then $s' \models \beta$.

Thm (Soundness of Hoare logic): If $\vdash \{\alpha\}\, c\, \{\beta\}$, then $\models \{\alpha\}\, c\, \{\beta\}$.

Proof by induction on the structure of the proof of $\vdash \{\alpha\}\, c\, \{\beta\}$.

Let π be a proof of $\vdash \{\alpha\}\, c\, \{\beta\}$ ending in a rule r.

The following cases arise.

- r = Hskip : α is β, and c is skip. For any states s and s', if $s -[\text{skip}]\rightarrow s'$, then s is s'. Thus, if $s \models \alpha$, then $s' \models \beta$.
- r = Hasgn : Consider s, s' s.t. $s \models \alpha(e)$ and $s -[x := e]\rightarrow s'$. Then, $s' = s[x \mapsto a]$, where $s \vdash e \Rightarrow a$. In general in logic, $s \models \alpha(e/x)$ iff $s[x \mapsto a] \models \alpha$, so done.

- r = If: Consider s, s' s.t. $s \models \alpha$ and $s -[\text{if } b \text{ then } c_1 \text{ else } c_2]\to s'$.

  (i) $s \models b$ and $s -[c_1]\to s'$:

  $s \models \alpha$ and $s \models b \Rightarrow s \models (\alpha \wedge b)$

  By IH, $\models \{\alpha \wedge b\}\, c_1\, \{b\}$. So, $s' \models b$.

  (ii) $s \not\models b$ and $s -[c_2]\to s'$:

  $s \models \alpha$ and $s \not\models b \Rightarrow s \models \alpha$ and $s \models \neg b \Rightarrow s \models (\alpha \wedge \neg b)$

  By IH, $\models \{\alpha \wedge \neg b\}\, c_2\, \{b\}$. So, $s' \models b$.

- r = While: We show that for any $n \in \mathbb{N}$, and for any s, s' s.t. $s \models \iota$ and there is a proof of $s -[\text{while } b \text{ do } c \text{ end}]\to s'$ of size $\leq n$, we have $s' \models \iota \wedge \neg b$.

- r = While : We show that for any $n \in \mathbb{N}$, and for any s, s' s.t. $s \models \iota$ and there is a proof of $s -[\text{while } b \text{ do } c \text{ end}]\to s'$ of size $\leq n$, we have $s' \models \iota \wedge \neg b$.

Consider a proof π of $s -[\text{while } b \text{ do } c \text{ end}]\to s'$ of size n, and assume the above statement holds for all $m < n$ (IH_1).

Two cases arise:

(i) $s \not\models b$ and $s = s'$.

$s \models \iota$ and $s \not\models b \Rightarrow s \models \iota \wedge \neg b$, done

(ii) $s \models b$ and there is some s'' s.t. $s -[c]\to s''$ and there is a proof of $s'' -[\text{while } b \text{ do } c \text{ end}]\to s'$ of size strictly less than n.

By IH, $\models \{\iota \wedge b\}\, c\, \{\iota\}$. By the definition of validity, $s'' \models \iota$.

By IH_1, $s' \models \iota \wedge \neg b$.