CS 520
Theory of Programming Language

03/31 – 04/07, 2021

1. Reminder.

② denot. semantics.

① Hoare logic. Specification using Hoare triples $\{p\}c\{q\}$.

Proof rules for showing the satisfaction of specs.

② Example. (Cdiv3). .... loop invariant.

↑ ① exercises.

2. Exercises.

① ✓ $\dfrac{}{\{q/x\to e\}\ x:=e\ \{q\}}$ ✓ $\dfrac{}{\{p\}\ x:=e\ \{\exists x_0.\ (x=(e/x\to x_0)) \wedge (p/x\to x_0)\}}$

↑ initial values of a, b. ↑

② $\{\underbrace{a\geq 1 \wedge b\geq 1 \wedge a=a_0 \wedge b=b_0}_{P}\}$ ✓

while $\neg(a=b)$ do

  if $(a>b)$ then $a:=a-b$.

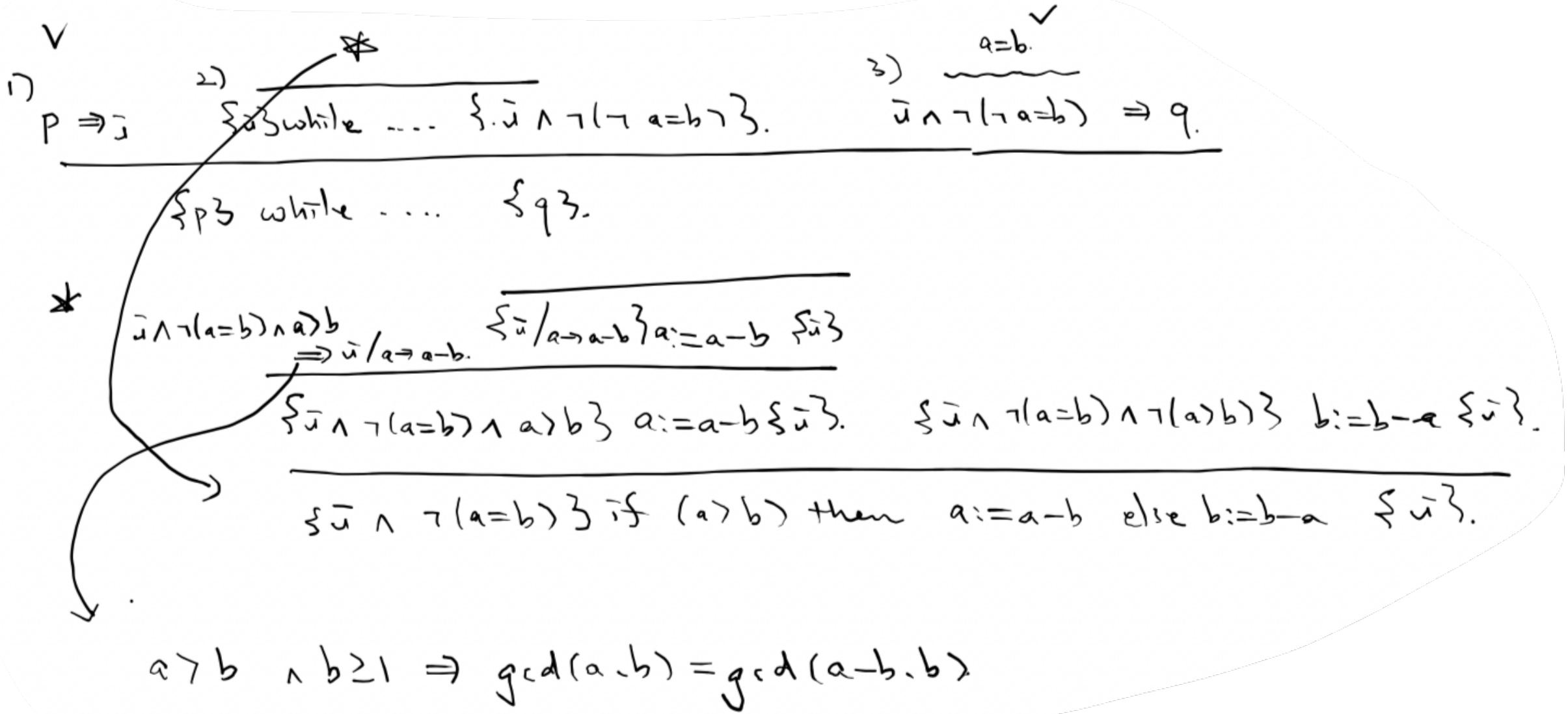
  else $b:=b-a$.

$\dfrac{\{\bar{i}\wedge b\}\ c\ \{\bar{i}\}}{\{\bar{i}\}\ \text{while } b \text{ do } c\ \{\bar{i}\wedge\neg b\}}$ ($\bar{i}$ = loop invariant)

$\{\underbrace{a=\gcd(a_0,b_0)}_{q}\}$ ✓

✓ $\bar{i} \equiv \left(\underbrace{\gcd(a_0,b_0)=\gcd(a,b)}\ \wedge\ a\geq 1\ \wedge\ b\geq 1.\right)$

Note. $a=b \Rightarrow \gcd(a,b)=a.$


1) p ⇒ J
2) {J} while ... {J ∧ ¬(¬ a=b)}.
3) a=b.
J ∧ ¬(¬ a=b) ⇒ q.
{p} while ... {q}.
J ∧ ¬(a=b) ∧ a>b ⇒ J/a→a-b.
{J/a→a-b} a:=a-b {J}
{J ∧ ¬(a=b) ∧ a>b} a:=a-b {J}.
{J ∧ ¬(a=b) ∧ ¬(a>b)} b:=b-a {J}.
{J ∧ ¬(a=b)} if (a>b) then a:=a-b else b:=b-a {J}.
a > b ∧ b ≥ 1 ⇒ gcd(a,b) = gcd(a-b, b)

3. Soundness of proof rules.

Thm. If we prove $\{p\}c\{q\}$ in Hoare logic, then $\{p\}c\{q\}$ is valid.

$$\llbracket \{p\}c\{q\} \rrbracket = tt.$$

Proof. "All the proof rules are correct."

$$\left[\begin{array}{l} \text{If } \dfrac{\varphi_1 \quad \varphi_2 \quad \cdots \quad \varphi_n}{\varphi} \text{ is a proof rule in Hoare logic,} \\ \text{then } \llbracket \varphi_1 \rrbracket = tt \wedge \llbracket \varphi_2 \rrbracket = tt \wedge \cdots \wedge \llbracket \varphi_n \rrbracket = tt \Longrightarrow \llbracket \varphi \rrbracket = tt \end{array}\right.$$

weakest precondition.

$$\frac{}{\{q[x \to e]\}\ x := e\ \{q\}}.$$

$$\frac{\{i \wedge b\}\ c\ \{i\}}{\{i\}\ \text{while } b \text{ do } c\ \{i \wedge \neg b\}}.$$

ex. Show the correctness of these rules.

① $\{q/x \to e\}\ x := e\ \{q\} = tt$

$\forall \sigma \in \Sigma$. if $[\![q/x \to e]\!]\sigma = tt$, then $[\![q]\!]_\perp([\![x := e]\!]\sigma) \sqsubseteq tt$.

$[\![q/x \to e]\!]\sigma = tt$ $\Rightarrow$ $[\![q]\!][\sigma \mid x : [\![e]\!]\sigma] = tt$.

$[\![q/x \to e]\!]\sigma = [\![q]\!][\sigma \mid x : [\![e]\!]\sigma]$ //.

$[\![q]\!][\sigma \mid x : [\![e]\!]\sigma] = tt$

$[\![q]\!]_\perp([\![x := e]\!]\sigma) = [\sigma \mid x : [\![e]\!]\sigma]$

$\Sigma_\perp \to \mathbb{B}_\perp$

$\{tt, \perp, ff\}$.

" Reminder of subst. lemma: $[\![q/x \to e]\!]\sigma = [\![q]\!][\sigma \mid x : [\![e]\!]\sigma]$ //

(2) $[\![\{i\}\ \text{while } b \text{ do } c\ \{i \wedge \neg b\}]\!] = tt$.

Sufficient $\bigwedge$ $\forall \sigma \in \Sigma$. , if $[\![i]\!]\sigma = tt$, then

to show that.

$[\![i \wedge \neg b]\!]_\perp\big([\![\text{while } b \text{ do } c]\!]\sigma\big) \sqsupseteq tt$

We may assume that. $\forall \sigma \in \Sigma$ if $[\![i \wedge b]\!]\sigma = tt$, then $[\![i]\!]_\perp([\![c]\!]\sigma) \sqsupseteq tt$.

Γ F where $F : [\Sigma \to_c \Sigma_\perp] \to_c [\Sigma \to_c \Sigma_\perp]$

$F(f)(\sigma') = \text{if } [\![b]\!]\sigma' = tt \text{ then } f_\perp([\![c]\!]\sigma') \text{ else } \sigma'$.

$(\Sigma \to \Sigma_\perp)$

$= \bigsqcup_{n=0}^{\infty} F^n(\perp)$

$\lambda\sigma'.\perp \sqsubseteq \Sigma$.

$[\Sigma \to_c \Sigma]$

$[\![i \wedge \neg b]\!]_\perp\Big(\Big(\bigsqcup_{n=0}^{\infty} F^n(\perp)\Big)(\sigma)\Big) \sqsupseteq tt$.

$= \bigsqcup_{n=0}^{\infty} \big(F^n(\perp)(\sigma)\big)$

$= \bigsqcup_{n=0}^{\infty} \big([\![i \wedge \neg b]\!]_\perp(F^n(\perp)(\sigma))\big) \sqsupseteq tt$.

$\Leftarrow$ $[\![i \wedge \neg b]\!]_\perp(F^n(\perp)(\sigma)) \sqsupseteq tt$ for all n.

By ind.

What we prove: for all $n \geq 0$,

for all $b \in \Sigma$, if $[\![\bar{i}]\!] b = tt$, then $[\![\bar{i} \wedge \neg b]\!]_1(F^n(\perp)(b)) \sqsubseteq tt$.

Inductive case:

Pick $b \in \Sigma$ s.t. $[\![\bar{i}]\!] b = tt$.

Case analysis on b.

(1) Case: $[\![b]\!] b = tt$ and $[\![c]\!] b \neq \perp$. (Body of the loop.)

To show: $\llbracket \neg a \vee \neg b \rrbracket_{\top}(F^{n}(\top)(q)) \sqsupseteq tt$.

$= \llbracket \neg a \vee \neg b \rrbracket_{\top}(F(F^{n-1}(\top))(q))$

$= \llbracket \neg a \vee \neg b \rrbracket_{\top}$ (if $\llbracket q \rrbracket = tt$ then $(F^{n-1}(\top))(\llbracket c \rrbracket)$ else q)

$= \llbracket \neg a \vee \neg b \rrbracket_{\top}(F^{n-1}(\top)(\llbracket c \rrbracket))$ $\sqsupseteq tt$. (By I.H.)

$F^{n-1}(\top)(\llbracket c \rrbracket) = tt$? Yes $\to$ I.H. No. $\Rightarrow$?

$\llbracket \neg a \rrbracket(\llbracket c \rrbracket) = tt$.

$\llbracket b \rrbracket(\llbracket c \rrbracket) = tt$.

$\llbracket \neg a \vee b \rrbracket(\llbracket c \rrbracket) = tt$.

By the assm. $\neg \llbracket \neg a \rrbracket_{\top}(\llbracket c \rrbracket) \sqsupseteq tt$.

$\neq \top$.

$\llbracket \neg a \rrbracket(\llbracket c \rrbracket) = tt$.

□

Failures, Input-Output, and Continuation (chap5)

1. Motivation / High-level message.

(1) P.L.s. lang with more realistic features. ... input, output, failures.

(2) Recursively defined domains. ..... deeper understanding of domains / predomains.

(elements of predomains represent computations in abstract form)

$$\Sigma \to \tilde{\Sigma}\,\underline{\Sigma}.$$

(3) Continuation.

(4) $\Pi \subset \Pi : \Sigma \to \Sigma_{\perp}.$
$\Pi \in \Pi : \Sigma \to \mathbb{Z}.$

2. Syntax of the language.

① <intexp> ::= ...

<boolexp> ::= ...

<comm> ::= .... | newvar <var> := <intexp> in <comm>.

| fail | !<intexp>. | ?<var>.

② e.g. (1) x := 124 ;
while true do
(?y ; if (y=0) then fail else (x := x/y ; !x))

2 input. 4 input. 0 input.
62. 25

(2) x := 124 ; y := 0 ;
(new bar x := 3 in fail) ;
* y := 2. y := 3

not executed.

○ ... x=124, y=0.
X ... x=3, y=0. ... → renaming of local variables won't preserve the meaning
X ... x=124, y=3.