Programming Languages:
Imperative Program Construction
1. Hoare Logic and Weakest Precondition: Non-Looping Constructs

Shin-Cheng Mu

Autumn Term, 2022

1 Hoare Logic

The Guarded Command Language

In this course we will talk about program construction using Dijkstra's calculus. Most of the materials are from Kaldewaij [Kal90].

- A program computing the greatest common divisor:

$$
\begin{array}{l}
\mathbf{con}\ A, B : \mathit{Int}\ \{0 < A \wedge 0 < B\} \\
\mathbf{var}\ \ x, y : \mathit{Int} \\
x, y := A, B \\
\mathbf{do}\ y < x \rightarrow x := x - y \\
\mid \quad x < y \rightarrow y := y - x \\
\mathbf{od} \\
\{x = y = \mathit{gcd}\ (A, B)\}\ \ .
\end{array}
$$

- Assignments denoted by $:=$; **do** denotes loops with guarded bodies.
- Assertions delimited in curly brackets.

The Hoare Triple

- Given a program statement S and predicates P and Q, the *Hoare triple* $\{P\}\,S\,\{Q\}$ is a Boolean value.
- Operationally, $\{P\}\,S\,\{Q\}$ is True iff. the statement S, when executed in a state satisfying P, *terminates* in a state satisfying Q.
- **Note**: in some flavours of theory, $\{P\}\,S\,\{Q\}$ need not imply termination. We will stick with the terminating version in our course.

Examples

- $\{x \geqslant 0 \wedge y \geqslant 0\}\,S\,\{r = x \times y\}$ is True iff. S is a program that, given non-negative x and y, terminates and stores $x \times y$ in r.
  - Nothing is said about values of x and y upon termination.
  - When $x \geqslant 0 \wedge y \geqslant 0$ does not hold, S may do anything — including looping forever.
- $\{z \geqslant 0\}\,S\,\{x \times y = z\}$ is True iff. S, given non-negative z, computes a factorization of z, and terminates.
- $\{x > 0\}\,S\,\{\mathit{True}\}$ is True iff. S is any program that terminates, provided that $x > 0$.

Some Properties

- $\{P\}\,S\,\{Q\}$ and $P_0 \Rightarrow P$ implies $\{P_0\}\,S\,\{Q\}$.
- $\{P\}\,S\,\{Q\}$ and $Q \Rightarrow Q_0$ implies $\{P\}\,S\,\{Q_0\}$.
- $\{P\}\,S\,\{Q\}$ and $\{P\}\,S\,\{R\}$ equivales $\{P\}\,S\,\{Q \wedge R\}$.
- $\{P\}\,S\,\{Q\}$ and $\{R\}\,S\,\{Q\}$ equivales $\{P \vee R\}\,S\,\{Q\}$.
- **Note**: "A equivales B" is another way to say "A if and only if B", also denoted by $A \equiv B$.

The No-Op Statement

- Perhaps the simplest statement: $\{P\}\,\mathit{skip}\,\{Q\}$ iff. $P \Rightarrow Q$.
  - E.g. $\{x > 0 \wedge y > 0\}\,\mathit{skip}\,\{x \geqslant 0\}$.

- Note that the annotations need not be "exact."

- Operationally, $skip$ is a statement that does nothing.
  - Why do we need a program that does nothing?
  - It is like why we need a number 0 that represents "nothing". It can be very useful sometimes.

2 Assignments

Substitution

- $P[x\backslash E]$: substituting *free* occurrences of x in P for E.
- We do so in mathematics all the time. A formal definition of substitution, however, is rather tedious.
- For this lecture we will only appeal to "common sense":
  - E.g. $(x \leqslant 3)[x\backslash x-1] \equiv x-1 \leqslant 3 \equiv x \leqslant 4$.
  - $$\begin{aligned} & (\langle \exists y : y \in \mathbb{N} : x < y\rangle \wedge y < x)[y\backslash y+1] \\ \equiv\ & \langle \exists y : y \in \mathbb{N} : x < y\rangle \wedge y+1 < x. \end{aligned}$$
  - $$\begin{aligned} & \langle \exists y : y \in \mathbb{N} : x < y\rangle[x\backslash y] \\ \equiv\ & \langle \exists z : z \in \mathbb{N} : y < z\rangle. \end{aligned}$$
- The notation $[x\backslash E]$ hints at "divide by x and multiply by E."
  - We have $x[x\backslash E] = E$. Nice!
- Just in case you may see different notations in other papers...
  - Many papers use the notation $[E/x]$. Either way, x is the denominator.
  - Kaldewaij actually wrote $[x := E]$, since substitution is closely related to assignments.
  - Some papers write P^x_E for $P[x\backslash E]$.

Substitution and Assignments

- Which is correct:
  1. $\{P\}\, x := E\, \{P[x\backslash E]\}$, or
  2. $\{P[x\backslash E]\}\, x := E\, \{P\}$?
- Answer: 2! For example:

$$\begin{aligned} & \{(x \leqslant 3)[x\backslash x+1]\}\, x := x+1\, \{x \leqslant 3\} \\ \equiv\ & \{x+1 \leqslant 3\}\, x := x+1\, \{x \leqslant 3\} \\ \equiv\ & \{x \leqslant 2\}\, x := x+1\, \{x \leqslant 3\}. \end{aligned}$$

3 Sequencing

Catenation

- $\{P\}\, S; T\, \{Q\}$ equivals that there exists R such that $\{P\}\, S\, \{R\}$ and $\{R\}\, T\, \{Q\}$.
- Verify:

$$\begin{array}{l} \textbf{var } x, y : \mathit{Int} \\ \{x = A \wedge y = B\} \\ x := x - y \\ \{y = B \wedge x + y = A\} \\ y := x + y \\ \{y - x = B \wedge y = A\} \\ x := y - x \\ \{x = B \wedge y = A\} \end{array}$$

4 Selection

If-Conditionals

- Selection takes the form **if** $B_0 \rightarrow S_0 \mid \ldots \mid Bn \rightarrow Sn$ **fi**.
- Each B_i is called a *guard*; $B_i \rightarrow S_i$ is a *guarded command*.
- If none of the guards $B_0 \ldots B_n$ evaluate to true, the program aborts. Otherwise, one of the command with a true guard is chosen *non-deterministically* and executed.

To annotate an **if** statement:

$$\begin{array}{l} \{P\} \\ \textbf{if } B_0 \rightarrow \{P \wedge B_0\}\, S_0\, \{Q, \mathsf{Pf}_0\} \\ \ \mid\ B_1 \rightarrow \{P \wedge B_1\}\, S_1\, \{Q, \mathsf{Pf}_1\} \\ \textbf{fi} \\ \{Q, \mathsf{Pf}_2\}\ \ , \end{array}$$

where Pf_0, Pf_1, Pf_2 are labels referring to proofs.

- Pf_0 refers to a proof of $\{P \wedge B_0\}\, S_0\, \{Q\}$;
- Pf_1 refers to a proof of $\{P \wedge B_1\}\, S_1\, \{Q\}$;
- Pf_2 refers to a proof of $P \Rightarrow B_0 \vee B_1$.
- The proofs and labels are sometimes omitted if they are trivial.

Binary Maximum

- Goal: to assign $x \uparrow y$ to z. By definition, $z = x \uparrow y \;\equiv\; (z = x \,\vee\, z = y) \,\wedge\, x \leqslant z \,\wedge\, y \leqslant z$.
- Try $z := x$. We reason:

$$\begin{aligned} & ((z = x \,\vee\, z = y) \,\wedge\, x \leqslant z \,\wedge\, y \leqslant z)[z \backslash x] \\ \equiv\; & (x = x \,\vee\, x = y) \,\wedge\, x \leqslant x \,\wedge\, y \leqslant x \\ \equiv\; & y \leqslant x, \end{aligned}$$

  which hinted at using a guarded command: $y \leqslant x \to z := x$.
- Indeed:

$$\begin{array}{l} \{\mathit{True}\} \\ \textbf{if } y \leqslant x \to \{y \leqslant x\}\; z := x \;\{z = x \uparrow y\} \\ \mid\; x \leqslant y \to \{x \leqslant y\}\; z := y \;\{z = x \uparrow y\} \\ \textbf{fi} \\ \{z = x \uparrow y\} \;\; . \end{array}$$

On Understanding Programs

- There are two ways to understand the program below:

$$\begin{array}{l} \textbf{if } B_{00} \to S_{00} \mid B_{01} \to S_{01} \textbf{ fi} \\ \textbf{if } B_{10} \to S_{10} \mid B_{11} \to S_{11} \textbf{ fi} \\ \quad\quad \vdots \\ \textbf{if } B_{n0} \to S_{n0} \mid B_{n1} \to S_{n1} \textbf{ fi}. \end{array}$$

- One takes effort exponential to n; the other is linear.
- Dijkstra: "…if we ever want to be able to compose really large programs reliably, we need a programming discipline such that the intellectual effort needed to understand a program does not grow more rapidly than in proportion to the program length." [Dijnd]

5 Weakest Precondition

State Space and Predicates

More precisely speaking…

- A *predicate* on A is a function having type $A \to \mathit{Bool}$.
  - E.g. $\mathit{even} :: \mathit{Int} \to \mathit{Bool}$ is a predicate on Int.
- The *state space* of a program is the states of all its variables.
  - E.g. state space for the GCD program, which has two variables x and y, is $(\mathit{Int} \times \mathit{Int})$.
- An expression having free variables can be seen as a function.
  - E.g. $x \leqslant y$ is a predicate (a function) with type $(\mathit{Int} \times \mathit{Int}) \to \mathit{Bool}$ that yields True for, e.g. $(x, y) = (3, 4)$ and False for $(x, y) = (4, 3)$.

In a Hoare Triple…

- In $\{P\}\, S\, \{Q\}$, P and Q shall be seen as *predicates* on the state space of the program S.
- E.g. In $\{z \geqslant 0\}\, S\, \{x \times y = z\}$, assuming that the program S uses only three variables x, y, and z.
  - The part $z \geqslant 0$ shall be understood as a predicate that takes x, y, and z, and returns True iff. $z \geqslant 0$.
  - The part $x \times y = z$ shall be understood as a predicate that takes x, y, and z, and returns True iff. $x \times y = z$.
- True in a Hoare triple can be understood as a predicate that returns True for any input; similarly with False.
- Let S be a program having variables x, y, z. That $\{P\}\, S\, \{Q\}$ being True means that if S starts running in a state such that $P\ (x, y, z) = \mathit{True}$, it terminates and yields a state such that $Q\ (x, y, z) = \mathit{True}$.

Stronger? Weaker?

- Given propositions P and Q, if $P \Rightarrow Q$, we say that Q is the *weaker* one, and P is the *stronger* one.
- Precisely speaking, P is *no weaker than* Q and Q is *no stronger than* P. But let's be a bit sloppy to avoid confusion…

Stronger and Weaker Predicates

- The convention extends to predicates. If $P\ x \Rightarrow Q\ x$ for every x, Q is the *weaker* one, while P is the *stronger* one.
- Example: $0 \leqslant x < 4$ is weaker than $0 \leqslant x < 3$, which is in turn weaker than $1 \leqslant x < 3$.

- Intuition: for first-order values, the set of values satisfying a weaker predicate is *larger* than that satisfying a stronger predicate.

- Example: P can be weaker than $P \wedge Q$ (since $(P \wedge Q) \Rightarrow P$); $P \vee Q$ can be weaker than P (since $P \Rightarrow (P \vee Q)$).

- Intuition: a weaker predicate enforces less restriction, is more tolerant, and allows more inputs/states to be True.

Predicate-Set Correspondence

- Functions can be hard to grasp.

- A predicate P is isomorphic to the set of values that satisfy the predicate — at least for first order values. Therefore I tend to equate them.

- E.g. think of $x \leqslant 3$ as the set of values satisfying $x \leqslant 3$.

- False is the empty set, True is the set of all values (of the right type).

- $P \Rightarrow Q$ iff. $P \subseteq Q$.

  - A weaker predicate is a bigger set!

- $P \wedge Q$ corresponds to $P \cap Q$; $P \vee Q$ corresponds to $P \cup Q$.

Weakest Precondition

- Recall that the predicates in a Hoare triple need not be exact.

  - $\{x \leqslant 2\}\, x := x + 1\, \{x \leqslant 3\}$ is a valid triple.
  - So is $\{0 < x \leqslant 2\}\, x := x + 1\, \{x \leqslant 3\}$. Note that $x \leqslant 2$ is weaker than $0 < x \leqslant 2$.
  - $x \leqslant 2$ is in fact the weakest (most tolerating) P such that $\{P\}\, x := x + 1\, \{x \leqslant 3\}$ holds.

- Defining weakest precondition in terms of Hoare triple....

- **Definition**: given a statement S, its *weakest precondition* with respect to Q, denoted $wp\ S\ Q$, is the weakest predicate such that $\{wp\ S\ Q\}\, S\, \{Q\}$ holds.

Predicate Transformer

$wp\ S$ is a function from predicates to predicates.

- Also called a *predicate transformer*.

- I myself find it sometimes easier to think of a predicate transformer as a function from sets to sets.

- E.g. $wp\ S\ Q$ gives you the *largest* set P such that for all $x \in P$, running S starting from initial state x gives you a final state in Q.

Weakest Precondition: Skip and Assignment

- Weakest preconditions for skip and $\mathit{assignment}$:

- $wp\ \mathit{skip}\ P = P$.

- $wp\ (x := E)\ P = P[x \backslash E]$.

Hoare Triple, Revisited

- We can do it the other way round: specify wp for each program construct, and define Hoare triple in terms of wp.

- **Definition**: $\{P\}\, S\, \{Q\}$ if and only if $P \Rightarrow wp\ S\ Q$.

Examples

- $\{x > 0\}\, \mathit{skip}\, \{x \geqslant 0\}$ is valid, because:

$$\begin{array}{ll} & wp\ \mathit{skip}\ (x \geqslant 0) \\ \equiv & \quad \{\text{ definition of } wp\ \} \\ & x \geqslant 0 \\ \Leftarrow & x > 0\ \ . \end{array}$$

- $\{0 < x < 2\}\, x := x + 1\, \{x \leqslant 3\}$ is valid, because

$$\begin{array}{ll} & wp\ (x := x + 1)\ (x \leqslant 3) \\ \equiv & \quad \{\text{ definition of } wp\ \} \\ & (x \leqslant 3)[x \backslash x + 1] \\ \equiv & x + 1 \leqslant 3 \\ \Leftarrow & 0 < x < 2\ \ . \end{array}$$

Sequencing and Branching

- $wp\ (S; T)\ Q = wp\ S\ (wp\ T\ Q)$.

  - Or $wp\ (S; T) = wp\ S \cdot wp\ T$, where $(\cdot)$ denotes function composition.

- $wp\ (\mathbf{if}\ B_0 \rightarrow S_0 \mid B_1 \rightarrow S_1\ \mathbf{fi})\ Q = (B_0 \Rightarrow wp\ S_0\ Q) \wedge (B_1 \Rightarrow wp\ S_1\ Q) \wedge (B_0 \vee B_1)$.

Semantics

What does a program *mean*?

- **Denotational semantics**: what a program *is*. Mapping programs to mathematical objects.
- **Operational semantics**: what a program *does*. How one program term transforms to another.
- **Axiomatic semantics**: what a program *guarantees*.
- *Predicate transformer semantics* can be seen as a kind of denotational semantics, and axiomatic semantics.
- The meaning of a program is a *predicate transformer*: give it a post condition Q, it tells us what precondition is sufficient to guarantee Q.
- It is a "goal oriented" semantics that is more suitable for reasoning about and constructing imperative programs.

Properties of Predicate Transformers

- wp must satisfy certain conditions.
- **Strictness**: $wp\ S\ \mathit{False} = \mathit{False}$.
- **Monotonicity**: $P \Rightarrow Q$ implies $wp\ S\ P \Rightarrow wp\ S\ Q$.
- **Distributivity over Conjunction**: $(wp\ S\ Q_0 \wedge wp\ S\ Q_1) \equiv wp\ S\ (Q_0 \wedge Q_1)$.
- One can prove that $(wp\ S\ Q_0 \vee wp\ S\ Q_1) \Rightarrow wp\ S\ (Q_0 \vee Q_1)$.
- $(wp\ S\ Q_0 \vee wp\ S\ Q_1) \equiv wp\ S\ (Q_0 \vee Q_1)$ holds only for *deterministic* programs.

6 Summary

The weakest-precondition semantics for each of the guarded command language are given below:

- $wp\ \mathit{skip}\ P = P$,
- $wp\ (x := E)\ P = P[x \backslash E]$,
- $wp\ (S; T)\ Q = wp\ S\ (wp\ T\ Q)$,
- $wp\ (\mathbf{if}\ B_0 \rightarrow S_0 \mid B_1 \rightarrow S_1\ \mathbf{fi})\ Q = (B_0 \Rightarrow wp\ S_0\ Q) \wedge (B_1 \Rightarrow wp\ S_1\ Q) \wedge (B_0 \vee B_1)$.

The situation for loops is a bit complicated. Abbreviate $\mathbf{do}\ B \rightarrow S\ \mathbf{od}$ to DO, we have

$$wp\ DO\ Q = (B \vee Q) \wedge (\neg B \vee wp\ S\ (wp\ DO\ Q))\ .$$

Based on the weakest preconditions, we have the following rules for constructs of the guarded command language.

- $\{P\}\ \mathit{skip}\ \{Q\}\ \equiv\ P \Rightarrow Q$.
- $\{P\}\ x := E\ \{Q\}\ \equiv\ P \Rightarrow Q[x \backslash E]$ and P implies that E is defined.
- $\{P\}\ S; T\ \{Q\}\ \equiv\ (\exists R :: \{P\}\ S\ \{R\} \wedge \{R\}\ T\ \{Q\})$.
- $\{P\}\ \mathbf{if}\ B_0 \rightarrow S_0\ \mid\ B_1 \rightarrow S_1\ \mathbf{fi}\ \{R\}$ equivals
  1. $P \Rightarrow B_0 \vee B_1$ and
  2. $\{P \wedge B_0\}\ S_0\ \{Q\}$ and $\{P \wedge B_1\}\ S_1\ \{Q\}$.
- $\{P\}\ \mathbf{do}\ B_0 \rightarrow S_0\ \mid\ B_1 \rightarrow S_1\ \mathbf{od}\ \{Q\}$ follows from
  1. $P \wedge \neg B_0 \wedge \neg B_1\ \Rightarrow\ Q$,
  2. $\{P \wedge B_0\}\ S_0\ \{P\}$ and $\{P \wedge B_1\}\ S_1\ \{P\}$, and
  3. there exists an integer function bnd on the state space such that
     (a) $P \wedge (B_0 \vee B_1)\ \Rightarrow\ \mathit{bnd} \geqslant 0$,
     (b) $\{P \wedge B_0 \wedge \mathit{bnd} = C\}\ S_0\ \{\mathit{bnd} < C\}$, and
     (c) $\{P \wedge B_1 \wedge \mathit{bnd} = C\}\ S_1\ \{\mathit{bnd} < C\}$.

Statements of the guarded command language satisfy the following rules:

- $\{P\}\ S\ \{\mathit{false}\}\ \equiv\ \neg P$,
- $\{P\}\ S\ \{Q\}\ \wedge\ (P_0 \Rightarrow P)\ \ \Rightarrow\ \ \{P_0\}\ S\ \{Q\}$,
- $\{P\}\ S\ \{Q\}\ \wedge\ (Q \Rightarrow Q_0)\ \ \Rightarrow\ \ \{P\}\ S\ \{Q_0\}$,
- $\{P\}\ S\ \{Q\}\ \wedge\ \{P\}\ S\ \{R\}\ \ \equiv\ \ \{P\}\ S\ \{Q \wedge R\}$,
- $\{P\}\ S\ \{Q\}\ \wedge\ \{R\}\ S\ \{Q\}\ \ \equiv\ \ \{P \vee R\}\ S\ \{Q\}$.

References

[Dijnd] E. W. Dijkstra. On understanding programs. EWD 264, circulated privately, n.d.

[Kal90] A. Kaldewaij. *Programming: the Derivation of Algorithms*. Prentice Hall, 1990.