Proof Rules For Partial Correctness

Overview

- **The proof rules that follow constitute an** *axiomatic semantics* **of our programming language**

- **They can be used to prove that programs meet partial correctness specifications**

Judgements

- **Three kinds of things that could be true or false have been introduced**
  - **Statements of mathematics, e.g.** $(\mathtt{X}+1)^2 = \mathtt{X}^2+2\times\mathtt{X}+1$
  - **Partial correctness specifications** $\{P\}\ C\ \{Q\}$
  - **Total correctness specifications** $[P]\ C\ [Q]$

- **These three kinds of things are examples of** *judgements*
  - **A logical system provides rules for establishing the truth (i.e.** *proving***) various kinds of judgements**
  - **Floyd-Hoare logic provides rules for proving partial correctness specifications**
  - **The laws of arithmetic, which are assumed known, provide ways of proving statements about integers**

- $\vdash S$ **means statement** S **can be proved**
  - **How to prove predicate calculus statements assumed known (via IB course 'Logic and Proof')**
  - **This course covers axioms and rules for proving** *program correctness statements*

Notation for Axioms and Rules

- **The axioms of Floyd-Hoare logic are specified by *schemas***
  - **these can be *instantiated* to get particular partial correctness specifications**

- **The inference rules of Floyd-Hoare logic will be specified with a notation of the form**

$$\frac{\vdash\ S_1,\ \ldots\ ,\ \vdash\ S_n}{\vdash\ S}$$

  - **This means the *conclusion* $\vdash S$ may be deduced from the *hypotheses* $\vdash S_1$, . . . , $\vdash S_n$**
  - **The hypotheses can either all be theorems of Floyd-Hoare logic**
  - **or a mixture of theorems of Floyd-Hoare logic and theorems of predicate calculus**

Skip

- **Syntax:** `SKIP`

- **Semantics: the state is unchanged**

The Skip Axiom

$$\vdash \ \{P\}\ \texttt{SKIP}\ \{P\}$$

- **It is a simple axiom schema**
  - P **can be instantiated with different values**

- **Instances of the skip axiom are:**
  - $\vdash\ \{\texttt{Y} = 2\}\ \texttt{SKIP}\ \{\texttt{Y} = 2\}$
  - $\vdash\ \{\texttt{T}\}\ \texttt{SKIP}\ \{\texttt{T}\}$
  - $\vdash\ \{\texttt{R=X+(Y} \times \texttt{Q)}\}\ \texttt{SKIP}\ \{\texttt{R=X+(Y} \times \texttt{Q)}\}$

Assignments

- **Syntax:** V `:=` E

- **Semantics: the state is changed by assigning the value of the term E to the variable V**

- **Example:** `X:=X+1`

  - **This adds one to the value of the variable** `X`

Preamble to the Assignment Axiom

- **The assignment axiom says that**
  - **the value of a variable V *after* executing an assignment command V:=E**
  - **equals the value of the expression E in the state *before* executing it**
- **If a statement P is to be true *after* the assignment**
- **Then the statement obtained by substituting E for V in P must be true *before* executing it**
- **Every statement about V in the postcondition, must correspond to a statement about E in the precondition**
  - **In the initial state V has a value which is about to be lost**

Substitution Notation

- **Define $P\texttt{[}E/V\texttt{]}$ to mean the result of replacing all occurrences of V in P by E**
  - **Read $P\texttt{[}E/V\texttt{]}$ as 'P with E for V'**
  - **For example,**

$$(\texttt{X}+1 > \texttt{X})\texttt{[Y+Z/X]} \;=\; ((\texttt{Y}+\texttt{Z})+1 > \texttt{Y}+\texttt{Z})$$

- **The way to remember this notation is to remember the 'cancellation law'**

$$V\texttt{[}E/V\texttt{]} \;=\; E$$

which is analogous to the cancellation property of fractions

$$v \times (e/v) \;=\; e$$

The Assignment Axiom

The Assignment Axiom

$$\vdash \ \{P[E/V]\}\ V := E\ \{P\}$$

Where V is any variable, E is any expression, P is any statement and the notation $P[E/V]$ denotes the result of substituting the term E for all occurrences of the variable V in the statement P.

- **Instances of the assignment axiom are**
  - $\vdash\ \{\mathtt{Y} = 2\}\ \mathtt{X} := 2\ \{\mathtt{Y} = \mathtt{X}\}$
  - $\vdash\ \{\mathtt{X} + 1 = \mathtt{n} + 1\}\ \mathtt{X} := \mathtt{X} + 1\ \{\mathtt{X} = \mathtt{n} + 1\}$
  - $\vdash\ \{E = E\}\ \mathtt{X} := E\ \{\mathtt{X} = E\}$ **(if $\mathtt{X}$ does not occur in E)**

The Backwards Fallacy

- **Many people feel the assignment axiom is 'backwards'**

- **One common erroneous intuition is that it should be**

$$\vdash \ \{P\}\ V\texttt{:=}E\ \{P\texttt{[}V/E\texttt{]}\}$$

  - **Where** $P\texttt{[}V/E\texttt{]}$ **denotes the result of substituting** V **for** E **in** P

  - **This has the false consequence** $\vdash\ \{\texttt{X=0}\}\ \texttt{X:=1}\ \{\texttt{X=0}\}$, **since the** `(X=0)[X/1]` **is equal to** `(X=0)` **as** `1` **doesn't occur in** `(X=0)`

- **Another erroneous intuition is that it should be**

$$\vdash \ \{P\}\ V\texttt{:=}E\ \{P\texttt{[}E/V\texttt{]}\}$$

  - **This has the false consequence** $\vdash\ \{\texttt{X=0}\}\ \texttt{X:=1}\ \{\texttt{1=0}\}$ **which follows by taking** P **to be** `X=0`, V **to be** `X` **and** E **to be** `1`

Validity

- **The fact that it is easy to have wrong intuitions about the assignment axiom shows that it is important to have rigorous means of establishing the validity of axioms and rules**

- **We will not go into this topic here**

  - **It is possible to give a** *formal semantics* **of our little programming language**

  - **and then to** *prove* **that the axioms and rules of inference of Floyd-Hoare logic are sound**

  - **This will only increase our confidence in the axioms and rules to the extent that we believe the correctness of the formal semantics**

- **The simple assignment axiom above is not valid for 'real' programming languages**

  - **G. Ligler has shown that it can fail to hold in six different ways for the language Algol 60**

Expressions with Side-effects

- **The validity of the assignment axiom depends on expressions not having side effects**

- **Suppose that our language were extended so that it contained the 'block expression'**

```
BEGIN Y:=1; 2 END
```

  - **This expression has value `2`, but its evaluation also 'side effects' the variable `Y` by storing `1` in it**

- **If the assignment axiom applied to block expressions, then it could be used to deduce**

$$\vdash \{\texttt{Y=0}\}\ \texttt{X:=BEGIN Y:=1; 2 END}\ \{\texttt{Y=0}\}$$

  - **since `(Y=0)[E/X] = (Y=0)` (because `X` does not occur) in `(Y=0)`**

  - **This is clearly false; after the assignment `Y` will have the value `1`**

Precondition Strengthening

- **Recall that**

$$\frac{\vdash\ S_1,\ \ldots\ ,\ \ \vdash\ S_n}{\vdash\ S}$$

means $\vdash\ S$ **can be deduced from** $\vdash\ S_1, \ldots, \vdash\ S_n$

- **Using this notation, the rule of precondition strengthening is**

Precondition strengthening

$$\frac{\vdash\ P \Rightarrow P', \qquad \vdash\ \{P'\}\ C\ \{Q\}}{\vdash\ \{P\}\ C\ \{Q\}}$$

Example

- **From**
  - $\vdash$ `X=n` $\Rightarrow$ `X+1=n+1`
    - **trivial arithmetical fact**
  - $\vdash \ \{\mathtt{X}+\mathtt{1}=\mathtt{n}+\mathtt{1}\}\ \mathtt{X}:=\mathtt{X}+\mathtt{1}\ \{\mathtt{X}=\mathtt{n}+\mathtt{1}\}$
    - **instance of the assignment axiom**

- **It follows by precondition strengthening that**

$$\vdash \ \{\mathtt{X}=\mathtt{n}\}\ \mathtt{X}:=\mathtt{X}+\mathtt{1}\ \{\mathtt{X}=\mathtt{n}+\mathtt{1}\}$$

  - `n` **is an** *auxiliary* **(or** *ghost***) variable**

Example

- **From**
  - $\vdash \texttt{T} \Rightarrow (E\texttt{=}E)$
  - $\vdash \ \{E = E\} \ \texttt{X} := E \ \{\texttt{X} = E\}$
- **It follows that**

$$\vdash \ \{\texttt{T}\} \ \texttt{X} := E \ \{\texttt{X} = E\}$$

  - **as long as if** $\texttt{X}$ **is not in** E **(why?)**

Postcondition weakening

- **Just as the previous rule allows the precondition of a partial correctness specification to be strengthened, the following one allows us to weaken the postcondition**

Postcondition weakening

$$\frac{\vdash\ \{P\}\ C\ \{Q'\}, \qquad \vdash\ Q' \Rightarrow Q}{\vdash\ \{P\}\ C\ \{Q\}}$$

An Example Formal Proof

- **Here is a little formal proof**

1.	$\vdash$ {`R=X` $\wedge$ `0=0`} `Q:=0` {`R=X` $\wedge$ `Q=0`}	By the assignment axiom
2.	$\vdash$ `R=X` $\Rightarrow$ `R=X` $\wedge$ `0=0`	By pure logic
3.	$\vdash$ {`R=X`} `Q:=0` {`R=X` $\wedge$ `Q=0`}	By precondition strengthening
4.	$\vdash$ `R=X` $\wedge$ `Q=0` $\Rightarrow$ `R=X+(Y` $\times$ `Q)`	By laws of arithmetic
5.	$\vdash$ {`R=X`} `Q:=0` {`R=X+(Y` $\times$ `Q)`}	By postcondition weakening

- **The rules precondition strengthening and post-condition weakening are sometimes called the** *rules of consequence*

Sequences

- **Syntax:** $C_1; \cdots ;C_n$

- **Semantics: the commands $C_1, \cdots, C_n$ are executed in that order**

- **Example:** `R:=X; X:=Y; Y:=R`

  - **The values of `X` and `Y` are swapped using `R` as a temporary variable**

  - **This command has the *side effect* of changing the value of variable `R` to the old value of variable `X`**

The sequencing rule

- **The next rule enables a partial correctness specification for a sequence $C_1;C_2$ to be derived from specifications for C_1 and C_2**

The sequencing rule

$$\frac{\vdash\ \{P\}\ C_1\ \{Q\}, \qquad \vdash\ \{Q\}\ C_2\ \{R\}}{\vdash\ \{P\}\ C_1;C_2\ \{R\}}$$

Example Proof

Example: By the assignment axiom:

(i) $\vdash$ {`X=x`$\wedge$`Y=y`} `R:=X` {`R=x`$\wedge$`Y=y`}

(ii) $\vdash$ {`R=x`$\wedge$`Y=y`} `X:=Y` {`R=x`$\wedge$`X=y`}

(iii) $\vdash$ {`R=x`$\wedge$`X=y`} `Y:=R` {`Y=x`$\wedge$`X=y`}

Hence by (i), (ii) and the sequencing rule

(iv) $\vdash$ {`X=x`$\wedge$`Y=y`} `R:=X; X:=Y` {`R=x`$\wedge$`X=y`}

Hence by (iv) and (iii) and the sequencing rule

(v) $\vdash$ {`X=x`$\wedge$`Y=y`} `R:=X; X:=Y; Y:=R` {`Y=x`$\wedge$`X=y`}

Blocks

- **Syntax:** `BEGIN VAR` V_1`;` $\cdots$ `VAR` V_n`;` C `END`

- **Semantics: the command C is executed, and then the values of $V_1, \cdots, V_n$ are restored to the values they had before the block was entered**

  - **The initial values of $V_1, \cdots, V_n$ inside the block are unspecified**

- **Example:** `BEGIN VAR R; R:=X; X:=Y; Y:=R END`

  - **The values of `X` and `Y` are swapped using `R` as a temporary variable**

  - **This command does *not* have a side effect on the variable `R`**

The Block Rule

- **The block rule takes care of local variables**

The block rule

$$\frac{\vdash\ \{P\}\ C\ \{Q\}}{\vdash\ \{P\}\ \texttt{BEGIN VAR}\ V_1\texttt{;}\ \ldots;\ \texttt{VAR}\ V_n\texttt{;}\ C\ \texttt{END}\ \{Q\}}$$

where none of the variables $V_1, \ldots, V_n$ occur in P or Q.

- **Note that the block rule is regarded as including the case when there are no local variables (the '$n = 0$' case)**

The Side Condition

- **The syntactic condition that none of the variables $V_1, \ldots, V_n$ occur in P or Q is an example of a *side condition***
  - **Without this condition the rule is invalid; this is illustrated in the example below**

- **From**
  - $\vdash$ {`X=x` $\wedge$ `Y=y`} `R:=X; X:=Y; Y:=R` {`Y=x` $\wedge$ `X=y`}
  - **it follows by the block rule that**

    ```
    ⊢ {X=x ∧ Y=y}
        BEGIN VAR R; R:=X; X:=Y; Y:=R END
       {Y=x ∧ X=y}
    ```

  - **since** `R` **does not occur in** `X=x` $\wedge$ `Y=y` **or** `X=y` $\wedge$ `Y=x`

- **However from**
  - $\vdash$ {`X=x` $\wedge$ `Y=y`} `R:=X; X:=Y` {`R=x` $\wedge$ `X=y`}
  - **one *cannot* deduce**

    ```
    ⊢ {X=x ∧ Y=y}
         BEGIN VAR R; R:=X; X:=Y END
       {R=x ∧ X=y}
    ```

  - **since** `R` **occurs in** `R=x` $\wedge$ `X=y`

An Exercise

Consider the specification

```
{X=x} BEGIN VAR X; X:=1 END {X=x}
```

Can this be deduced from the rules given so far?

(i) If so, give a proof of it

(ii) If not, explain why not and suggest additional rules and/or axioms to enable it to be deduced

Exercises

- **Is the following true?**

$$\vdash \{\texttt{X=x} \wedge \texttt{Y=y}\}\ \texttt{X:=X+Y; Y:=X-Y; X:=X-Y}\ \{\texttt{Y=x} \wedge \texttt{X=y}\}$$

  - **If so prove it**
  - **If not, give the circumstances when it fails**

- **Show**

$$\vdash \{\texttt{X=R+(Y}\times\texttt{Q)}\}\ \texttt{BEGIN R:=R-Y; Q:=Q+1 END}\ \{\texttt{X=R+(Y}\times\texttt{Q)}\}$$

Two-armed conditionals

- **Syntax:** `IF` S `THEN` C_1 `ELSE` C_2

- **Semantics:**

  - **If the statement S is true in the current state, then C_1 is executed**

  - **If S is false, then C_2 is executed**

- **Example:** `IF X<Y THEN MAX:=Y ELSE MAX:=X`

  - **The value of the variable `MAX` it set to the maximum of the values of `X` and `Y`**

- **One-armed conditional is defined by:**

$$\texttt{IF}\ S\ \texttt{THEN}\ C \quad \equiv \quad \texttt{IF}\ S\ \texttt{THEN}\ C\ \texttt{ELSE}\ \texttt{SKIP}$$

The Conditional Rule

The conditional rule

$$\frac{\vdash\ \{P \wedge S\}\ C_1\ \{Q\}, \qquad \vdash\ \{P \wedge \neg S\}\ C_2\ \{Q\}}{\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

- **Suppose we are given**

$\vdash$ {T ∧ X≥Y} MAX:=X {MAX=max(X,Y)}

- **and**

$\vdash$ {T ∧ ¬(X≥Y)} MAX:=Y {MAX=max(X,Y)}

- **Then by the conditional rule it follows that**

$\vdash$ {T} IF X≥Y THEN MAX:=X ELSE MAX:=Y {MAX=max(X,Y)}

- **A one-armed conditional rule is derived later**

`WHILE`-commands

- **Syntax:** `WHILE` S `DO` C
- **Semantics:**
  - If the statement S is true in the current state, then C is executed and the `WHILE`-command is repeated
  - If S is false, then nothing is done
  - Thus C is repeatedly executed until the value of S becomes false
  - If S never becomes false, then the execution of the command never terminates
- **Example:** `WHILE ¬(X=0) DO X:= X-2`
  - If the value of `X` is non-zero, then its value is decreased by 2 and then process is repeated
- **This `WHILE`-command will terminate (with `X` having value `0`) if the value of `X` is an even non-negative number**
  - In all other states it will not terminate

Invariants

- **Suppose** $\vdash \{P \wedge S\}\ C\ \{P\}$
- **then** P **is an** *invariant* **of** C **whenever** S **holds**
- **The `WHILE`-rule says that**
  - **if** P **is an invariant of the body of a `WHILE`-command whenever the test condition holds**
  - **then** P **is an invariant of the whole `WHILE`-command**
- **In other words**
  - **if executing** C *once* **preserves the truth of** P
  - **then executing** C *any number of times* **also preserves the truth of** P
- **The `WHILE`-rule also expresses the fact that after a `WHILE`-command has terminated, the test must be false**
  - **Otherwise, it wouldn't have terminated**

The WHILE-Rule

The WHILE-rule

$$\frac{\vdash\ \{P \wedge S\}\ C\ \{P\}}{\vdash\ \{P\}\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ \{P \wedge \neg S\}}$$

- **It is easy to show**

$\vdash\ \{\texttt{X=R+(Y}\times\texttt{Q)}\wedge\texttt{Y}\leq\texttt{R}\}\ \texttt{BEGIN R:=R-Y; Q:=Q+1 END}\ \{\texttt{X=R+(Y}\times\texttt{Q)}\}$

- **Hence by the WHILE-rule with** P = '$\texttt{X=R+(Y}\times\texttt{Q)}$'

$$\begin{array}{l}
\vdash\ \{\texttt{X=R+(Y}\times\texttt{Q)}\}\\
\quad\texttt{WHILE Y}\leq\texttt{R DO}\\
\qquad\texttt{BEGIN R:=R-Y; Q:=Q+1 END}\\
\ \{\texttt{X=R+(Y}\times\texttt{Q)}\ \wedge\ \neg(\texttt{Y}\leq\texttt{R})\}
\end{array}$$

Example

- **From the previous slide**

$$\vdash \{\texttt{X=R+(Y}\times\texttt{Q)}\}$$

```
  WHILE Y≤R DO
    BEGIN R:=R-Y; Q:=Q+1 END
```

$$\{\texttt{X=R+(Y}\times\texttt{Q)} \wedge \neg(\texttt{Y}\leq\texttt{R})\}$$

- **It is easy to deduce that**

$$\vdash \{\texttt{T}\}\ \texttt{R:=X; Q:=0}\ \{\texttt{X=R+(Y}\times\texttt{Q)}\}$$

- **Hence by the sequencing rule and postcondition weakening**

$$\vdash \{\texttt{T}\}$$

```
  R:=X;
  Q:=0;
  WHILE Y≤R DO
    BEGIN R:=R-Y; Q:=Q+1 END
```

$$\{\texttt{R<Y} \wedge \texttt{X=R+(Y}\times\texttt{Q)}\}$$

How do you find an invariant?

- **Look at the facts:**
  - **It must hold initially**
  - **With the negated test it must establish the result**
  - **The body must leave it unchanged**

- **Think about how the loop works**
  - **The invariant says that what** *has been done so far* **together with what** *remains to be done* **gives** *the desired result*

Example

- **Consider a factorial program**

```
{X=n ∧ Y = 1}
 WHILE X≠0 DO
   BEGIN Y:=Y×X; X:=X-1 END
{X=0 ∧ Y=n!}
```

- **Look at the Facts**
  - **Finally** `X=0` **and** `Y=n!`
  - **Initially** `X=n` **and** `Y=1`
  - **On each loop** `Y` **is increased and,** `X` **is decreased**

- **Think how the loop works**
  - `Y` **holds the result so far**
  - `X!` **is what remains to be computed**
  - `n!` **is the desired result**

- **The invariant is** `X!×Y=n!`

Conjunction and Disjunction

Specification conjunction

$$\frac{\vdash\ \{P_1\}\ C\ \{Q_1\}, \qquad \vdash\ \{P_2\}\ C\ \{Q_2\}}{\vdash\ \{P_1 \wedge P_2\}\ C\ \{Q_1 \wedge Q_2\}}$$

Specification disjunction

$$\frac{\vdash\ \{P_1\}\ C\ \{Q_1\}, \qquad \vdash\ \{P_2\}\ C\ \{Q_2\}}{\vdash\ \{P_1 \vee P_2\}\ C\ \{Q_1 \vee Q_2\}}$$

- **These rules are useful for splitting a proof into independent bits**
  - **They enable** $\vdash \{\mathtt{P}\}\ \mathtt{C}\ \{\mathtt{Q}_1 \wedge \mathtt{Q}_2\}$ **to be proved by proving separately that both** $\vdash \{\mathtt{P}\}\ \mathtt{C}\ \{\mathtt{Q}_1\}$ **and also that** $\vdash \{\mathtt{P}\}\ \mathtt{C}\ \{\mathtt{Q}_2\}$

Summary

- **We have shown how rules can be devised that allow us to make judgements about partial correctness statements**

- **It can be hard to get the rules right in the first place**

- **We can use the rules to prove that programs meet their specifications**

- **The rules reduce the proof to symbol pushing**
  - **With practice this is routine**
  - **The hard part is in formulating invariants**

- **Applying the rules can be fiddly**
  - **Next we will look at ways to make proofs easier to perform**