Verification of Contracts

- Given: Specification of imperative **procedure** by **Precondition** and **Postcondition**
- Goal: Formal proof for
  $\mathbf{Precondition}(State) \Rightarrow \mathbf{Postcondition}(\mathbf{procedure}(State))$
- Method: **Hoare Logic**, *i.e.*, a proof system for **Hoare triples** of the form

$$\{\mathbf{Precondition}\}\ \mathbf{procedure}\ \{\mathbf{Postcondition}\}$$

- named after C.A.R. Hoare, the inventor of Quicksort, CSP, and many other
- here: method bodies, no recursion, no pointers (extensions exist)

Syntax

E, F	$::=$	$c \mid x \mid E + F \mid \dots$	expressions
B, P, Q	$::=$	$\neg B \mid P \wedge Q \mid P \vee Q$	boolean expressions
	$\mid$	$E = F \mid E \leq F \mid \dots$	
C, D	$::=$	`skip`	statements
	$\mid$	x=E	assignment
	$\mid$	$C; D$	sequence
	$\mid$	`if` B `then` C `else` D	conditional
	$\mid$	`while` B `do` C	iteration
$\mathcal{H}$	$::=$	$\{P\}\ C\ \{Q\}$	Hoare triples

- (boolean) expressions are free of side effects

Semantics — Domains and Types

$$\begin{array}{rcl} BValue & = & \texttt{true} \mid \texttt{false} \\ IValue & = & 0 \mid 1 \mid \ldots \\ \sigma \in State & = & Variable \to Value \end{array}$$

$$\begin{array}{rcl} \mathcal{E}[\![]\!] & : & Expression \times State \to IValue \\ \mathcal{B}[\![]\!] & : & BoolExpression \times State \to BValue \\ \mathcal{S}[\![]\!] & : & State_\perp \to State_\perp \end{array}$$

- $State_\perp := State \cup \{\perp\}$
- result $\perp$ indicates non-termination

Semantics — Expressions

$$
\begin{aligned}
\mathcal{E}[\![c]\!]\sigma &= c \\
\mathcal{E}[\![x]\!]\sigma &= \sigma(x) \\
\mathcal{E}[\![E{+}F]\!]\sigma &= \mathcal{E}[\![E]\!]\sigma + \mathcal{E}[\![F]\!]\sigma \\
\ldots \\
\mathcal{B}[\![E{=}F]\!]\sigma &= \mathcal{E}[\![E]\!]\sigma = \mathcal{E}[\![F]\!]\sigma \\
\mathcal{B}[\![\neg B]\!]\sigma &= \neg\mathcal{B}[\![B]\!]\sigma \\
\ldots
\end{aligned}
$$

Semantics — Statements

$$
\begin{array}{rcl}
\mathcal{S}[\![C]\!]\bot & = & \bot \\
\mathcal{S}[\![\texttt{skip}]\!]\sigma & = & \sigma \\
\mathcal{S}[\![x{=}E]\!]\sigma & = & \sigma[x \mapsto \mathcal{E}[\![E]\!]\sigma] \\
\mathcal{S}[\![C;\ D]\!]\sigma & = & \mathcal{S}[\![D]\!](\mathcal{S}[\![C]\!]\sigma) \\
\mathcal{S}[\![\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ D]\!]\sigma & = & \mathcal{B}[\![B]\!]\sigma = \texttt{true} \rightarrow \mathcal{S}[\![C]\!]\sigma\ ,\ \mathcal{S}[\![D]\!]\sigma \\
\mathcal{S}[\![\texttt{while}\ B\ \texttt{do}\ C]\!]\sigma & = & F(\sigma) \\
\textit{where} \quad F(\sigma) & = & \mathcal{B}[\![B]\!]\sigma = \texttt{true} \rightarrow F(\mathcal{S}[\![C]\!]\sigma)\ ,\ \sigma
\end{array}
$$

Proving a Hoare triple

$$\{P\}\ C\ \{Q\}$$

- holds if $(\forall\,\sigma \in \mathit{State})\ P(\sigma) \Rightarrow (Q(\mathcal{S}[\![C]\!]\sigma) \vee \mathcal{S}[\![C]\!]\sigma = \bot)$ (partial correctness)
- alternative reading: $P, Q \subseteq \mathit{State}$
  $\{P\}\ C\ \{Q\} \equiv \mathcal{S}[\![C]\!]P \subseteq Q \cup \bot$
- define
  - strongest postcondition: $\mathit{post}(P) = \mathcal{S}[\![C]\!]P$
  - weakest precondition: $\mathit{wp}(Q) = \mathcal{S}[\![C]\!]^{-1}(Q)$
  - weakest liberal precondition: $\mathit{wlp}(Q) = \mathcal{S}[\![C]\!]^{-1}(Q \cup \{\bot\})$

Proof Rules for Hoare Triples

- Proving that $\{P\}\ C\ \{Q\}$ holds directly from the definition is tedious
- Instead: define axioms and inferences rules
- Construct a derivation to prove the triple
- Choice of axioms and rules guided by structure of C

Skip Axiom

$$\{P\}\ \texttt{skip}\ \{P\}$$

Correctness:

- $(\forall\, \sigma \in P)\ \mathcal{S}[\![\texttt{skip}]\!](\sigma) = \sigma \in P$
- P is wp and P is strongest postcondition
- terminates

Assignment Axiom

$$\{P[x \mapsto E]\}\ x = E\ \{P\}$$

Examples:

- $\{1 == 1\}\ x = 1\ \{x == 1\}$
- $\{odd(1)\}\ x = 1\ \{odd(x)\}$
- $\{x == 2 * y + 1\}\ y = 2 * y\ \{x == y + 1\}$

Correctness:

- Let $\sigma' = \mathcal{S}[\![x{=}E]\!]\sigma = \sigma[x \mapsto \mathcal{E}[\![E]\!]\sigma] \in P$

$\Leftrightarrow$ $\texttt{true} = \mathcal{B}[\![P]\!]\sigma' = \mathcal{B}[\![P]\!](\sigma[x \mapsto \mathcal{E}[\![E]\!]\sigma]) = \mathcal{B}[\![P[x \mapsto E]]\!](\sigma)$

$\Leftrightarrow$ $\sigma \in P[x \mapsto E]$

- terminates

Sequence Rule

$$\frac{\{P\}\ C\ \{R\} \qquad \{R\}\ D\ \{Q\}}{\{P\}\ C;\ D\ \{Q\}}$$

Examples:

$$\frac{\{x == 2*y+1\}\ y = 2*y\ \{x == y+1\} \qquad \{x == y+1\}\ y = y+1\ \{x == y\}}{\{x == 2*y+1\}\ y = 2*y;\ y = y+1\ \{x == y\}}$$

$$\frac{\dots}{\{0 == 0 \wedge 1 == 1 \wedge 1 == 1\}\ i = 0;\ k = 1;\ sum = 1\ \{i == 0 \wedge k == 1 \wedge sum == 1\}}$$

Correctness:

- If $\mathcal{S}[\![C]\!](P) \subseteq R$ and $\mathcal{S}[\![D]\!](R) \subseteq Q$, then $\mathcal{S}[\![D]\!](\mathcal{S}[\![C]\!](P)) \subseteq Q$.

Conditional Rule

$$\frac{\{P \land B\}\ C\ \{Q\} \qquad \{P \land \neg B\}\ D\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ D\ \{Q\}}$$

Correctness:

- Let $\sigma \in P$
- $\mathcal{S}[\![\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ D]\!](\sigma) = \mathcal{B}[\![B]\!]\sigma = \texttt{true} \rightarrow \mathcal{S}[\![C]\!]\sigma\ ,\ \mathcal{S}[\![D]\!]\sigma$
- $\mathcal{B}[\![B]\!]\sigma = \texttt{true} \equiv \sigma \in P \land B$
  Antecedent yields: $\mathcal{S}[\![C]\!]\sigma \in Q$
- $\mathcal{B}[\![B]\!]\sigma = \texttt{false} \equiv \sigma \in P \land \neg B$
  Antecedent yields: $\mathcal{S}[\![D]\!]\sigma \in Q$
- Hence, $\mathcal{B}[\![B]\!]\sigma = \texttt{true} \rightarrow \mathcal{S}[\![C]\!]\sigma\ ,\ \mathcal{S}[\![D]\!]\sigma \in Q$.

Conditional Rule — Issues

Examples:

$$\frac{\{P \wedge x < 0\}\ z = -x\ \{z == \mid x \mid\} \qquad \{P \wedge x \geq 0\}\ z = x\ \{z == \mid x \mid\}}{\{P\}\ \texttt{if}\ x < 0\ \texttt{then}\ z = -x\ \texttt{else}\ z = x\ \{z == \mid x \mid\}}$$

- incomplete!
- precondition for $z = -x$ should be $(z == \mid x \mid)[z \mapsto -x] \equiv -x == \mid x \mid$

$\Rightarrow$ need **logical rules**

Logical Rules

- strengthen precondition

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

- weaken postcondition

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

Correctness obvious

- Example needs strengthening: $P \wedge x < 0 \Rightarrow -x == \mid x \mid$
- holds if $P \equiv \mathbf{true}$!
- similarly: $P \wedge x \geq 0 \Rightarrow x == \mid x \mid$

Completed example:

$$\mathcal{D}_1 = \frac{x < 0 \Rightarrow -x == \mid x \mid \qquad \{-x == \mid x \mid\}\ z = -x\ \{z == \mid x \mid\}}{\{x < 0\}\ z = -x\ \{z == \mid x \mid\}}$$

$$\mathcal{D}_2 = \frac{x \geq 0 \Rightarrow x == \mid x \mid \qquad \{x == \mid x \mid\}\ z = x\ \{z == \mid x \mid\}}{\{x \geq 0\}\ z = x\ \{z == \mid x \mid\}}$$

$$\frac{\dfrac{\mathcal{D}_1}{\{x < 0\}\ z = -x\ \{z == \mid x \mid\}} \qquad \dfrac{\mathcal{D}_2}{\{x \geq 0\}\ z = x\ \{z == \mid x \mid\}}}{\{\mathbf{true}\}\ \mathtt{if}\ x < 0\ \mathtt{then}\ z = -x\ \mathtt{else}\ z = x\ \{z == \mid x \mid\}}$$

While Rule

$$\frac{\{P \wedge B\}\ C\ \{P\}}{\{P\}\ \texttt{while}\ B\ \texttt{do}\ C\ \{P \wedge \neg B\}}$$

- P is **loop invariant**

Example: try to prove

```
{ a>=0 /\ i==0 /\ k==1 /\ sum==1 }
while sum <= a do
  k = k+2;
  i = i+1;
  sum = sum+k
{ i*i <= a /\ a < (i+1)*(i+1) }
```

$\Rightarrow$ while rule not directly applicable . . .

Step 1: Find the loop invariant

```
a>=0 /\ i==0 /\ k==1 /\ sum==1
=>
i*i<=a /\ i>=0 /\ k==2*i+1 /\ sum==(i+1)*(i+1)
```

- $P \equiv a \geq 0 \wedge i \geq 0 \wedge k == 2 * i + 1 \wedge sum == (i + 1) * (i + 1)$ holds on entry to the loop
- To prove that P is an invariant, requires to prove that $\{P \wedge sum \leq a\}\ k = k + 1;\ i = i + 1;\ sum = sum + k\ \{P\}$
- It follows by the sequence rule and weakening:

Proof of loop invariance

```
{ i*i<=a /\ i>=0   /\ k==2*i+1      /\ sum==(i+1)*(i+1) /\ sum<=a }
{          i>=0   /\ k+2==2+2*i+1 /\ sum==(i+1)*(i+1) /\ sum<=a }
k = k+2
{          i>=0   /\ k==2+2*i+1   /\ sum==(i+1)*(i+1) /\ sum<=a }
{          i+1>=1 /\ k==2*(i+1)+1 /\ sum==(i+1)*(i+1) /\ sum<=a }
i = i+1
{          i>=1   /\ k==2*i+1     /\ sum==i*i         /\ sum<=a }
{ i*i<=a /\ i>=1   /\ k==2*i+1     /\ sum+k==i*i+k     /\ sum+k<=a+k }
sum = sum+k
{ i*i<=a /\ i>=1   /\ k==2*i+1     /\ sum==i*i+k       /\ sum<=a+k }
{ i*i<=a /\ i>=1   /\ k==2*i+1     /\ sum==i*i+2*i+1   /\ sum<=a+k }
{ i*i<=a /\ i>=1   /\ k==2*i+1     /\ sum==(i+1)*(i+1) /\ sum<=a+k }
{ i*i<=a /\ i>=0   /\ k==2*i+1     /\ sum==(i+1)*(i+1) }
```

Step 2: Apply the while rule

$$\frac{\{P \wedge sum \leq a\}\ k = k+1;\ i = i+1;\ sum = sum + k\ \{P\}}{\{P\}\ \texttt{while}\ sum \leq a\ \texttt{do}\ k = k+1;\ i = i+1;\ sum = sum + k\ \{P \wedge sum > a\}}$$

Now, $P \wedge sum > a$ is

```
  { i*i<=a /\ i>=0   /\ k==2*i+1      /\ sum==(i+1)*(i+1) /\ sum>a }
implies
  { i*i<=a /\ a<(i+1)*(i+1) }
```

Correctness of the while rule

$$\frac{\{P \wedge B\}\ C\ \{P\}}{\{P\}\ \texttt{while}\ B\ \texttt{do}\ C\ \{P \wedge \neg B\}}$$

- Suppose that $\sigma \in P$ and let $\sigma' = \mathcal{S}[\![\texttt{while}\ B\ \texttt{do}\ C]\!](\sigma) = F(\sigma)$
- where $F(\sigma) = \mathcal{B}[\![B]\!]\sigma = \texttt{true} \rightarrow F(\mathcal{S}[\![C]\!]\sigma)\ ,\ \sigma$
- If $F(\sigma) = \bot$, then the conclusion holds.
- Otherwise, prove by induction on the number n of recursive calls of F that $F(P) \subseteq P \wedge \neg B$
- $n = 0$: it must be that $\mathcal{B}[\![B]\!]\sigma = \texttt{false}$ so that $\sigma \in P \wedge \neg B$.
- $n > 0$: it must be that $\mathcal{B}[\![B]\!]\sigma = \texttt{true}$ so that $\sigma \in P \wedge B$. In this case, $F(\sigma) = F(\mathcal{S}[\![C]\!]\sigma)$ and by assumption $\mathcal{S}[\![C]\!]\sigma \in P$.
- Now, the inductive hypothesis applied to $\sigma' = F(\mathcal{S}[\![C]\!]\sigma)$ yields $\sigma' \in P \wedge \neg B$

Termination

A loop `while` B `do` C terminates if there is a well-founded ordering $(A, \succ)$ and a termination value $t \in A$ such that for all values $t_{\text{before C}}$ it holds that $t_{\text{before C}} \succ t_{\text{after C}}$.

In a **well-founded ordering**, all decreasing chains $t_1 \succ t_2 \succ \ldots$ are finite.

Examples for well-founded orderings:

- $(\mathbb{N}, >)$
- $(\mathbb{N} \times \mathbb{N}, >)$ where $(a, b) > (c, d)$ if $a > c$ or $a = c$ and $b > d$
- lexicographic ordering on fixed-length tuples of well-founded orderings

Counterexamples: orderings that are not well-founded

- $(\mathbb{Z}, >)$
  $0, -1, -2, -3, \ldots$
- $(\mathbb{Q}^+, >)$
  $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$
- lexicographic ordering on $\{a, b\}^*$
  $b, ab, aab, aaab, aaaab, \ldots$

Termination of the root example

- Choose $t = a - i * i$ in $(\mathbb{N}, >)$.
- Recall the loop invariant

```
i*i<=a /\ i>=0 /\ k==2*i+1 /\ sum==(i+1)*(i+1)
```

- Hence $a - i * i \geq 0$, *i.e.*, $\in \mathbb{N}$
- If $t_{\text{before C}} = a - i * i$, then
  $t_{\text{after C}} = a - (i+1) * (i+1) = a - i * i - 2i - 1$

$\Rightarrow$ $t_{\text{before C}} > t_{\text{after C}}$ (since $i \geq 0$ is also an invariant)

Another example: Greatest common divisor

```
{ x1 > 0 /\ x2 > 0 }
y1 = x1; y2 = x2;
{ x1 > 0 /\ x2 > 0 /\ x1 == y1 /\ x2 == y2 }
while y1 <> y2 do
  if y1 < y2 then
     y2 = y2 % y1
  else
     y1 = y1 % y2
{ y1 == gcd(x1, x2) }
```

- Invariant?

Invariant of GCD loop

$$P \equiv gcd(x1, x2) == gcd(y1, y2)$$

- Holds on entry since $x1 == y1$ and $x2 == y2$
- $\{y1 < y2 \wedge gcd(x1, x2) == gcd(y1, y2)\}$
  `y2 = y2 % y1`
  $\{gcd(x1, x2) == gcd(y1, y2)\}$
  holds because the precondition of the assignment is
  $\{gcd(x1, x2) == gcd(y1, y2\%y1)\}$
  and $gcd(y1, y2) == gcd(y1, y2\%y1)$
  For the latter: suppose that $d \mid y1$ and $d \mid y2$ and $r = y2\%y1$, that is $y2 = m \cdot y1 + r$ with $m > 0$ (since $y1 < y2$). Now $d \mid m \cdot y1 + r$ if and only if $d \mid r$.
- analogously for the else branch of the conditional

Greatest common divisor with invariant

```
{ x1 > 0 ∧ x2 > 0 }
y1 = x1; y2 = x2;
{ x1 > 0 ∧ x2 > 0 ∧ x1 == y1 ∧ x2 == y2 }

{ gcd (x1, x2) == gcd (y1, y2) }
while y1 <> y2 do
  { gcd (x1, x2) == gcd (y1, y2) ∧ y1 <> y2 }
  if y1 < y2 then
     y2 = y2 % y1
  else
     y1 = y1 % y2
  { gcd (x1, x2) == gcd (y1, y2) }
{ gcd (x1, x2) == gcd (y1, y2) ∧ y1 == y2 }

{ y1 == gcd(x1, x2) }
```

Termination of gcd

```
while y1 <> y2 do
  if y1 < y2 then
     y2 = y2 % y1
  else
     y1 = y1 % y2
```

- let $t = (y1, y2) \in (\mathbb{N} \times \mathbb{N}, >)$ (lexicographic ordering)
- if $y1 < y2$, then $y2\%y1 < y2$
- if $y1 > y2$, then $y1\%y2 < y1$

$\Rightarrow$ in both cases, t decreases

$\Rightarrow$ loop terminates

$\Rightarrow$ code is totally correct

Properties of Formal Verification

- requires more restrictions on assertions (*e.g.*, use a certain logic) than monitoring
- full compliance of code with specification can be guaranteed
- scalability is a challenging research topic:
  - full automatization
  - manageable for small/medium examples
  - large examples require manual interaction
  - general problem is undecidable