Systeme Hoher Qualität und Sicherheit
Vorlesung 9 vom 16.12.2013: Verification with Floyd-Hoare-Logic

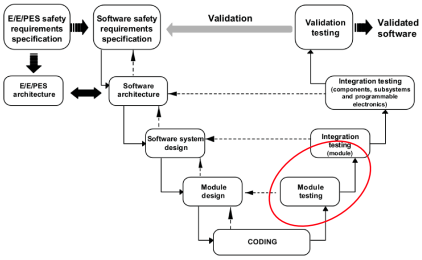
Christoph Lüth & Christian Liguda

Universität Bremen

Wintersemester 2013/14

Where are we?

Floyd-Hoare logic in the Development Process



- The Floyd-Hoare calculus **proves** properties of **sequential** programs.
- Thus, it is at home in the **lower levels** of the **verification branch**, much like the static analysis from last week.
- It is far more powerful than static analysis — and hence, far more **complex to use** (it requires user interaction, and is not **automatic**).

Idea

- What does this compute? $P = N!$
- How can we **prove** this?
- Inuitively, we argue about which value variables have at certain points in the program.
- Thus, to prove properties of imperative programs like this, we need a formalism where we can formalise **assertions** of the program properties at certain points in the exection, and which tells us how these assertions change with **program execution**.

```
{1 ≤ N}
P := 1;
C := 1;
while C ≤ N do {
  P := P × C;
  C := C + 1
}
{P = N!}
```


Floyd-Hoare-Logic

- Floyd-Hoare-Logic consists of a set of **rules** to derive valid assertions about programs. The assertions are denoted in the form of **Floyd-Hoare-Triples**.
- The logical language has both **logical** variables (which do not change), and **program** variables (the value of which changes with program execution).
- Floyd-Hoare-Logic has one basic **principle** and one basic **trick**.
- The **principle** is to **abstract** from the program state into the logical language; in particular, **assignment** is mapped to **substitution**.
- The **trick** is dealing with iteration: iteration corresponds to induction in the logic, and thus is handled with an inductive proof. The trick here is that in most cases we need to **strengthen** our assertion to obtain an **invariant**.

Recall Our Small Language

- Arithmetic Expressions (**AExp**)

$$a ::= \mathbf{N} \mid \mathbf{Loc} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2$$

  with variables **Loc**, numerals **N**

- Boolean Expressions (**BExp**)

$$b ::= \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid a_1 < a_2 \mid \neg b \mid b_1 \wedge b2 \mid b_1 \vee b_2$$

- Statements (**Com**)

$$c ::= \mathbf{skip} \mid \mathbf{Loc} := \mathbf{AExp} \mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \mid \mathbf{while}\ b\ \mathbf{do}\ c \mid c_1; c_2 \mid \{c\}$$

Semantics of our Small Language

- The semantics of an imperative language is **state transition**: the program has an ambient state, and changes it by assigning **values** to certain **locations**
- Concrete example: execution starting with $N = 3$

P	?
C	?
N	3

$\rightsquigarrow$

P	1
C	?
N	3

$\rightsquigarrow$

P	1
C	1
N	3

$\rightsquigarrow$

P	1
C	1
N	3

$\rightsquigarrow \ldots$

P	6
C	4
N	3

Semantics in a nutshell

- Expressions evaluate to **values** **Val**(in our case, integers)
- A program state maps locations to values: $\Sigma = \mathbf{Loc} \rightharpoonup \mathbf{Val}$
- A programs maps an initial state to **possibly** a final state (if it terminates)
- Assertions are predicates over **program states**.

Floyd-Hoare-Triples

Partial Correctness ($\models \{P\}\, c\, \{Q\}$)

c is **partial correct** with **precondition** P and **postcondition** Q if:
for all states σ which satisfy P
if the execution of c on σ terminates in σ'
then σ' satisfies Q

Total Correctness ($\models [P]\, c\, [Q]$)

c is **total correct** with **precondition** P and **postcondition** Q if:
for all states σ which satisfy P
the execution of c on σ terminates in σ'
and σ' satisfies Q

- $\models \{\mathbf{true}\}\ \mathbf{while}\ \mathbf{true}\ \mathbf{do}\ \mathbf{skip}\ \{\mathbf{false}\}$ holds
- $\models [\mathbf{true}]\ \mathbf{while}\ \mathbf{true}\ \mathbf{do}\ \mathbf{skip}\ [\mathbf{false}]$ does **not** hold

Assertion Language

- Extension of **AExp** and **BExp** by
  - **logical** variables **Var** $v := n, m, p, q, k, l, u, v, x, y, z$
  - defined functions and predicates on **Aexp** $n!, \sum_{i=1}^{n}, \ldots$
  - implication, quantification $b_1 \Rightarrow b_2, \forall v.\, b, \exists v.\, b$
- **Aexpv**

$$a ::= \mathbf{N} \mid \mathbf{Loc} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid \mathbf{Var} \mid f(e_1, \ldots, e_n)$$

- **Bexpv**

$$b ::= \mathbf{true} \mid \mathbf{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b2 \mid b_1 \vee b_2 \mid b_1 \Rightarrow b_2 \mid p(e_1, \ldots, e_n) \mid \forall v.\, b \mid \exists v.\, b$$

Rules of Floyd-Hoare-Logic

- The Floyd-Hoare logic allows us to **derive** assertions of the form $\vdash \{P\}\, c\, \{Q\}$
- The **calculus** of Floyd-Hoare logic consists of six rules of the form

$$\frac{\vdash \{P_1\}\, c_1\, \{Q_1\} \ldots \vdash \{P_n\}\, c_n\, \{Q_n\}}{\vdash \{P\}\, c\, \{Q\}}$$

- This means we can derive $\vdash \{P\}\, c\, \{Q\}$ if we can derive $\vdash \{P_i\}\, c_i\, \{Q_i\}$
- There is one rule for each construction of the language.

Rules of Floyd-Hoare Logic: Assignment

$$\frac{}{\vdash \{B[e/X]\}\, X := e\, \{B\}}$$

- An assigment X:=e changes the state such that at location X we now have the value of expression e. Thus, in the state **before** the assignment, instead of X we must refer to e.
- It is quite natural to think that this rule should be the other way around.
- Examples:

X := 10;
$\{0 < 10 \longleftrightarrow (X < 10)[X/0]\}$
X := 0
$\{X < 10\}$

$\{X < 9 \longleftrightarrow X + 1 < 10\}$
X := X+ 1
$\{X < 10\}$

Rules of Floyd-Hoare Logic: Conditional and Sequencing

$$\frac{\vdash \{A \wedge b\}\, c_0\, \{B\} \qquad \vdash \{A \wedge \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1\, \{B\}}$$

- In the precondition of the positive branch, the condition b holds, whereas in the negative branch the negation $\neg b$ holds.
- Both branches must end in the same postcondition.

$$\frac{\vdash \{A\}\, c_0\, \{B\} \qquad \vdash \{B\}\, c_1\, \{C\}}{\vdash \{A\}\, c_0; c_1\, \{C\}}$$

- We need an intermediate state predicate B.

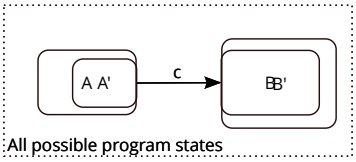
Rules of Floyd-Hoare Logic: Iteration

$$\frac{\vdash \{A \wedge b\}\, c\, \{A\}}{\vdash \{A\}\ \mathbf{while}\ b\ \mathbf{do}\, c\, \{A \wedge \neg b\}}$$

- Iteration corresponds to **induction**. Recall that in (natural) induction we have to show the **same** property P holds for 0, and continues to hold: if it holds for n, then it also holds for $n+1$.
- Analogously, here we need an **invariant** A which has to hold both **before** and **after** the body (but not necessarily in between).
- In the precondition of the body, we can assume the loop condition holds.
- The precondition of the iteration is simply the invariant A, and the postcondition of the iteration is A and the negation of the loop condition.

Rules of Floyd-Hoare Logic: Weakening

$$\frac{A' \longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$



- $\models \{A\}\, c\, \{B\}$ means that whenever we start in a state where A holds, c ends[1] in state where B holds.
- Further, for two sets of states, $P \subseteq Q$ iff $P \longrightarrow Q$.
- We can restrict the set A to A' ($A' \subseteq A$ or $A' \longrightarrow A$) and we can enlarge the set B to B' ($B \subseteq B'$ or $B \longrightarrow B'$), and obtain $\models \{A'\}\, c\, \{B'\}$.

[1] If end it does.

Overview: Rules of Floyd-Hoare-Logic

$$\frac{}{\vdash \{A\}\ \mathbf{skip}\, \{A\}} \qquad \frac{}{\vdash \{B[e/X]\}\, X := e\, \{B\}}$$

$$\frac{\vdash \{A \wedge b\}\, c_0\, \{B\} \qquad \vdash \{A \wedge \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1\, \{B\}}$$

$$\frac{\vdash \{A \wedge b\}\, c\, \{A\}}{\vdash \{A\}\ \mathbf{while}\ b\ \mathbf{do}\, c\, \{A \wedge \neg b\}} \qquad \frac{\vdash \{A\}\, c_0\, \{B\} \qquad \vdash \{B\}\, c_1\, \{C\}}{\vdash \{A\}\, c_0; c_1\, \{C\}}$$

$$\frac{A' \longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$

Properties of Hoare-Logic

Soundness

If $\vdash \{P\}\, c\, \{Q\}$, then $\models \{P\}\, c\, \{Q\}$

- If we derive a correctness assertion, it holds.
- This is shown by defining a formal semantics for the programming language, and showing that all rules are correct wrt. to that semantics.

Relative Completeness

If $\models \{P\}\, c\, \{Q\}$, then $\vdash \{P\}\, c\, \{Q\}$ except for the weakening conditions.

- Failure to derive a correctness assertion is always due to a failure to prove some logical statements (in the weakening).
- First-order logic itself is incomplete, so this result is as good as we can get.

The Need for Verification

Consider the following variations of the faculty example.
Which ones are correct?

$\{1 \leq N\}$
```
P := 1;
C := 1;
while C≤N do {
  C := C+1
  P := P×C;
}
```
$\{P = N!\}$

$\{1 \leq N\}$
```
P := 1;
C := 1;
while C<N do {
  C := C+1
  P := P×C;
}
```
$\{P = N!\}$

$\{1 \leq N \wedge n = N\}$
```
P := 1;
while 0<N do {
  P := P×N;
  N := N–1
}
```
$\{P = n!\}$

A Hatful of Examples

$\{i = Y \wedge Y \geq 0\}$
```
X := 1;
while ¬ (Y = 0) do {
 Y := Y–1;
 X := 2×X
}
```
$\{X = 2^i\}$

$\{A \geq 0 \wedge B \geq 0\}$
```
Q := 0;
R := A–(B×Q);
while B ≤ R do {
 Q := Q+1;
 R := A–(B×Q)
}
```
$\{A = B * Q + R \wedge R < B\}$

$\{0 < A\}$
```
T:= 1;
S:= 1;
I:= 0;
while S ≤ A do {
  T := T+ 2;
  S := S+ T;
  I := I+ 1
  }
```
$\{I * I <= A \wedge A < (I+1) * (I+1)\}$

Summary

- Floyd-Hoare logic in a nutshell:
  - The logic abstracts over the concrete program state by **program assertions**
  - Program assertions are boolean expressions, enriched by **logical** variables (and more)
  - We can prove partial correctness assertions of the form $\models \{P\}\, c\, \{Q\}$ (or total $\models [P]\, c\, [Q]$).
- Validity (correctness wrt a real programming language) depends **very much** on capturing the **exact** semantics formally.
- Floyd-Hoare logic itself is rarely used directly in practice, **verification condition generation** is — see next lecture.