Verification Methods

Verification methods may be classified according to the following main criteria:

- ***Proof-based vs. model-based*** - if a soundness and completeness theorem holds, than:
  - proof = valid formula = true in all models;
  - model-based = check satisfiability in one model
- ***Degree of automation*** - fully automated, partially automated, or manual
- ***Full- vs. property-verification*** - a single property vs. full behavior
- ***Domain of application*** - hardware or software; sequential or concurrent; reactive or terminating; etc.
- ***Pre- vs. post-development***

Program Verification — Where it Stands

Used to verify sequential programs with infinite state and complex data.

- Proof based
- Semi-automatic — some steps cannot be carried out algorithmically by a computer.
- Property-oriented
- Aplication domain: Sequential, transformational programs
- Pre/post development: the methods can be used during the development process to create small proofs that can be subsequently combined into proofs of larger program fragments.

Why should we specify and verify code

- A formal specification is less ambiguous.
- Experience has shown that verifying programs w.r.t. formal specifications can significantly cut down the duration of software development and maintainance by eliminating most errors in the planning phase.
- Makes debugging easier
- Software built from formal specifications is easier to reuse.
- Verification of safety-critical software ***guarantees*** safety; testing does not.
- Many examples of software-related catastrophies due to lack of verification.
  - Arianne rocket exploded immediately after launch
  - Lost control of Martian probe
  - Y2K problem

A Framework for Software Verification

As a software developer, you may get an order from a customer, which provides an informal description of your task.

- Convert the informal description D of an application domain into an "equivalent" formula Φ_D of some symbolic logic.
- Write a program P which is meant to realize Φ_D in the programming environment required by the customer.
- Prove that P satisfies Φ_D.

A Core Programming Language

We use a language with simple integer and boolean expressions, and simple commands: assignment, if, and while commands.

$$
\begin{aligned}
E &::= n \mid x \mid (-E) \mid (E+E) \mid (E-E) \mid (E*E) \\
B &::= \texttt{true} \mid \texttt{false} \mid (!B) \mid (B\&B) \mid (B||) \mid E<E \\
C &::= x=E \mid C;C \mid \texttt{if}\ B\ \{C\}\ \texttt{else}\ \{C\} \mid \texttt{while}\ B\ \{C\}
\end{aligned}
$$

Example:

```
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
```


Hoare Triples

We need to be able to express the following statement: "If the execution of a program fragment P starts in a state satisfying Φ, then the execution of P ends in a state satisfying Ψ. We denote this by:

$$(\!|\Phi|\!)\ P\ (\!|\Psi|\!)$$

and we call this construct a ***Hoare triple***. Φ is called the ***precondition***, and Ψ is called the ***postcondition***.

Example: Assume that the specification of a program P is ***"to calculate a number whose square is less that x."*** Then, the following assertion should hold:

$$(\!|x>0|\!)\ P\ (\!|y \cdot y < x|\!)$$

It means: if we start execution in a state where $x>0$, then the execution of P ends with a state where $y^2 < x$.

What happens if the execution starts with $x \leq 0$? We don't know!

Examples

Both these examples realize the specification $(\!|x > 0|\!)\ P\ (\!|y \cdot y < x|\!)$.

$$(\!|x > 0|\!)$$
```
y = 0
```
$$(\!|y \cdot y < x|\!)$$

$$(\!|x > 0|\!)$$
```
y = 0
while (y*y < x) {
  y = y+1
}
y = y-1
```
$$(\!|y \cdot y < x|\!)$$

Partial and Total Correctness

- ***Partial correctness:*** we ***do not require*** the program to terminate.
- ***Total correctness:*** we ***do require*** the program to terminate.

Definition (partial correctness): We say that the triple $(\!|\Phi|\!)\ \P\ (\!|\Psi|\!)$ is satisfied under partial correctness if, for all states which satisfy Φ, the state resulting from P's execution satisfies the postcondition Ψ, provided that P actually terminates. In this case we write

$$\models_{par} (\!|\Phi|\!)\ \P\ (\!|\Psi|\!)$$

Definition (total correctness): We say that the triple $(\!|\Phi|\!)\ \P\ (\!|\Psi|\!)$ is satisfied total partial correctness if, for all states in which P is executed and which satisfy the precondition Φ, P is guaranteed to terminate, and the state resulting from P's execution satisfies the postcondition Ψ. In this case we write

$$\models_{tot} (\!|\Phi|\!)\ \P\ (\!|\Psi|\!)$$

Examples

The following statement

$$\models_{par} (\!|\Phi|\!)\ \texttt{while true}\ \{\ \texttt{x} = 0;\ \}\ (\!|\Psi|\!)$$

holds for all Φ and Ψ. The corresponding total correctness statement does not hold.

Succ:
```
a = x + 1;
if (a - 1 == 0 {
  y = 1;
} else {
  y = a;
}
```

We have:

$$\models_{par} (\!|\top|\!)\ \texttt{Succ}\ (\!|y = x+1|\!)$$

and

$$\models_{tot} (\!|\top|\!)\ \texttt{Succ}\ (\!|y = x+1|\!)$$

Remark: $\models_{tot} (\!|\Phi|\!)\ P\ (\!|\Psi|\!)$ implies $\models_{par} (\!|\Phi|\!)\ P\ (\!|\Psi|\!)$.

Program Variables and Logical Variables

Consider the examples:

Fac2:
```
y = 1;
while (x != 0) {
  y = y * x;
  x = x - 1;
}
```

Sum:
```
z = 0;
while (x > 0) {
  z = z + x;
  x = x - 1;
}
```

The values of y and z are functions of ***the original*** values of x. That value is no longer available as a program variable at the end of the program. We introduce logical variables to handle this situation.

$$\models_{tot} (\!|\texttt{x} = x_0 \wedge \texttt{x} \geq 0|\!)\ \texttt{Fac2}\ (\!|\texttt{y} = x_0!|\!)$$

$$\models_{tot} (\!|\texttt{x} = x_0 \wedge \texttt{x} > 0|\!)\ \texttt{sum}\ \left(\!\left|\texttt{z} = \frac{x_0(x_0+1)}{2}\right|\!\right)$$

Proof Calculus for Partial Correctness

$$\frac{(\!|\phi|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ C_1; C_2\ (\!|\psi|\!)}\ \text{Composition}$$

$$\frac{}{(\!|\psi[E/x]|\!)\ x = E\ (\!|\psi|\!)}\ \text{Assignment}$$

$$\frac{(\!|\phi \wedge B|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi \wedge \neg B|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ \texttt{if}\ B\ \{C_1\}\ \texttt{else}\ \{C_2\}\ (\!|\psi|\!)}\ \text{If-statement}$$

$$\frac{(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)}{(\!|\psi|\!)\ \texttt{while}\ B\ \{C\}\ (\!|\psi \wedge \neg B|\!)}\ \text{Partial-while}$$

$$\frac{\vdash \phi' \to \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash \psi \to \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\ \text{Implied}$$

Proof Trees (1)

$$\dfrac{\dfrac{(\!|1 = 1|\!)\ \texttt{y = 1}\ (\!|y = 1|\!)}{(\!|\top|\!)\ \texttt{y = 1}\ (\!|y = 1|\!)}\,i \qquad \dfrac{(\!|y = 1 \wedge 0 = 0|\!)\ \texttt{z = 0}\ (\!|y = 1 \wedge z = 0|\!)}{(\!|y = 1|\!)\ \texttt{z = 0}\ (\!|y = 1 \wedge z = 0|\!)}\,i}{(\!|\top|\!)\ \texttt{y = 1; z = 0}\ (\!|y = 1 \wedge z = 0|\!)}\,c$$

Proof Trees (2)

$$\dfrac{\dfrac{\dfrac{(\!|y\cdot(z+1)=(z+1)!|\!)\ \texttt{z = z+1}\ (\!|y\cdot z=z!|\!)}{(\!|y=z!\land z\neq x|\!)\ \texttt{z = z+1}\ (\!|y\cdot z=z!|\!)}\,i \qquad (\!|y\cdot z=z!|\!)\ \texttt{y = y*z}\ (\!|y=z!|\!)}{(\!|y=z!\land z\neq x|\!)\ \texttt{z = z+1; y = y*z}\ (\!|y=z!|\!)}\,c}{\dfrac{(\!|y=z!|\!)\ \texttt{while (z != x) \{z = z+1; y = y*z\}}\ (\!|y=z!\land z=x|\!)}{(\!|y=1\land z=0|\!)\ \texttt{while (z != x) \{z = z+1; y = y*z\}}\ (\!|y=x!|\!)}\,i}\,w$$

Proof Trees (3)

Using the rule for composition, we get

$$(\!|\top|\!)\ \texttt{y = 1; z = 0; while (z != x) \{z = z+1; y = y*z\}}\ (\!|y=x!|\!)$$

Proof Tableaux

The rule for sequential composition suggests a more convenient way of presenting proofs in program logic: ***proof tableaux***. We can think of any program of our core programming language as a sequence.

Corresponding tableau:

C_1;
C_2;
⋮
C_n

Each of the transitions

$$(\!|\Phi_i|\!)\ C_{i+1}\ (\!|\Phi_{i+1}|\!)$$

appeals to one of the proof rules

$(\!|\Phi_0|\!)$	
C_1;	
$(\!|\Phi_1|\!)$	justification
C_2;	
$(\!|\Phi_2|\!)$	justification
⋮ $(\!|\Phi_{n-1}|\!)$	justification
C_2;	
$(\!|\Phi_n|\!)$	justification

Examples:Assignment

We show $\vdash_{\mathsf{par}} (\!|y=5|\!)\ \texttt{x = y + 1}\ (\!|x=6|\!)$:

$(\!|y=5|\!)$	
$(\!|y+1=6|\!)$	Implied
`x = y + 1`	
$(\!|x=6|\!)$	Assignment

We prove $\vdash_{\mathsf{par}} (\!|y<3|\!)\ \texttt{y = y + 1}\ (\!|y<4|\!)$:

$(\!|y<3|\!)$	
$(\!|y+1<4|\!)$	Implied
`y = y + 1;`	
$(\!|y<4|\!)$	Assignment

Example: If Statement

$(\!|\top|\!)$	
$(\!|(x+1-1=0\to 1=x+1)\land(\neg(x+1-1=0)\to x+1=x+1)|\!)$	Implied
`a = x + 1;`	
$(\!|(a-1=0\to 1=x+1)\land(\neg(a-1=0)\to a=x+1)|\!)$	Assignment
`if (a - 1 == 0) {`	
$(\!|1=x+1|\!)$	If-Statement
`y = 1;`	
$(\!|y=x+1|\!)$	Assignment
`} else {`	
$(\!|a=x+1|\!)$	If-Statement
`y = a;`	
$(\!|y=x+1|\!)$	Assignment
`}`	
$(\!|y=x+1|\!)$	If-Statement

Invariant

Definition: An ***invariant*** of the while-statement `while` B $\{C\}$ having guard B and body C is a formula η such that $\models_{par} (\!|\eta\land B|\!)\ C\ (\!|\eta|\!)$; i.e., if η and B are true in a state and C is executed and terminates, then η is again true in the resulting state.

Example:

```
y = 1;
z = 0;
while (z != x) {
   z = z + 1;
   y = y * z;
   }
```

iteration	z	y	B
0	0	1	true
1	1	1	true
2	2	2	true
3	3	6	true
4	4	24	true
5	5	120	true
6	6	720	false

Invariant: `y = z!`

Example

```
    (|⊤|)
    (|1 = 0!|)                          Implied
y = 1;
    (|y = 0!|)                          Assignment
z = 0;
    (|y = z!|)                          Assignment
while (z != x) {
        (|y = z! ∧ z ≠ x|)              Invariant Hyp. ∧ guard
        (|y · (z+1) = (z+1)!|)          Implied
    z = z + 1;
        (|y · z = z!|)                  Assignment
    y = y * z;
        (|y = z!|)                      Assignment
}
    (|y = z! ∧ ¬(z ≠ x)|)               Partial-while
    (|y = x!|)                          Implied
```


While Rule for Total Correctness

$$\frac{(\!|\eta \wedge B \wedge 0 \leq E = E_0|\!)\ C\ (\!|\eta \wedge 0 \leq E < E_0|\!)}{(\!|\eta \wedge 0 \leq E|\!)\ \texttt{while}\ B\ \{C\}\ (\!|\eta \wedge \neg B|\!)}\quad \text{Total-while}$$

Example

```
    (|x ≥ 0|)
    (|1 = 0! ∧ 0 ≤ x − 0|)                             Implied
y = 1;
    (|y = 0! ∧ 0 ≤ x − 0|)                             Assignment
z = 0;
    (|y = z! ∧ 0 ≤ x − z|)                             Assignment
while (x != z) {
        (|y = z! ∧ x ≠ z ∧ 0 ≤ x − z = E0|)            Invariant Hyp. ∧ guard
        (|y · (z+1) = (z+1)! ∧ 0 ≤ x − (z+1) < E0|)    Implied
    z = z + 1;
        (|y · z = z! ∧ 0 ≤ x − z < E0|)                Assignment
    y = y * z;
        (|y = z! ∧ 0 ≤ x − z < E0|)                    Assignment
}
    (|y = z! ∧ x = z|)                                 Total-while
    (|y = x!|)                                         Implied
```