Desarrollo de Software con Verificadores Deductivos Automáticos

Paqui Lucio

Dpto de Lenguajes y Sistemas Informáticos.

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Valencia, 15 de Diciembre de 2016

From Turing to Hoare Logic

- Turing 1949: *Checking a large routine*
  Programs can be mathematically described and analyzed.

From Turing to Hoare Logic

- Turing 1949: *Checking a large routine*
  Programs can be mathematically described and analyzed.
- Naur 1966: *Proof of algorithms by general snapshots*
  The need for a proof arises because we want to convince ourselves that the algorithm we have written is correct.
- Floyd 1967: *Assigning meaning to programs*
  A "formally verified" program is one whose correctness can be proved mathematically.

From Turing to Hoare Logic

- Turing 1949: *Checking a large routine*
  Programs can be mathematically described and analyzed.
- Naur 1966: *Proof of algorithms by general snapshots*
  The need for a proof arises because we want to convince ourselves that the algorithm we have written is correct.
- Floyd 1967: *Assigning meaning to programs*
  A "formally verified" program is one whose correctness can be proved mathematically.
- Hoare 1969: *An axiomatic basis for computer programming*
  A "correctness proof" is made w.r.t. a mathematical description (or specification) of what the program is intended to do (for all possible values of its input).

- Is program P correct? $\equiv$ Does P satisfy its specification S?
  - P is written in a programming language PL
  - S is written in a specification language SL
  - Axiomatic semantics of PL: Set of rules for establishing that P satisfies S

- Is program P correct? $\equiv$ Does P satisfy its specification S?
  - P is written in a programming language PL

    While-Programs
  - S is written in a specification language SL

    First-Order Formulas
  - Axiomatic semantics of PL: Set of rules for establishing that P satisfies S

    Hoare Formal System (a.k.a.Hoare Logic)

While-Programs

- Null: $\texttt{skip}$
- (Simoultaneous) Assignment: $\mathtt{x_1, \ldots, x_n := t_1, \ldots, t_n}$
- Sequential Composition: $P_1; P_2$
- Conditional: $\texttt{if } b \texttt{ then } P_1 \texttt{ else } P_2$
- Iteration: $\texttt{while } b \texttt{ do } P$

- Example:

```
r := 0;
while (r+1)∗(r+1) ≤ x
      do
      r := r+1
```

First-Order Logic (FOL)

- First-order syntax:

$$\varphi ::= p(\bar{t}) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \forall x(\varphi) \mid \exists x(\varphi)$$

$$\forall n(\ (n > 2 \wedge even(n)) \rightarrow \\ \exists x \exists y(prime(x) \wedge prime(y) \wedge n = x + y))$$

First-Order Logic (FOL)

- First-order syntax:

$$\varphi ::= p(\bar{t}) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \forall x(\varphi) \mid \exists x(\varphi)$$

$$\forall n(\ (n > 2 \wedge even(n)) \rightarrow \exists x \exists y(prime(x) \wedge prime(y) \wedge n = x + y))$$

- $\varphi[\bar{t}/\bar{x}]$ stands for the formula that results by (simultaneously) replace every *free* occurrence of each x_i in φ by t_i.

$$\exists z(x * y = 2 * z)[y+1, x-1/x, y] = \exists z((y+1) * (x-1) = 2 * z)$$

First-Order Logic (FOL)

- First-order syntax:

  $$\varphi ::= p(\bar{t}) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \forall x(\varphi) \mid \exists x(\varphi)$$

  $$\forall n(\ (n > 2 \wedge even(n)) \rightarrow \\ \quad \exists x \exists y(prime(x) \wedge prime(y) \wedge n = x + y))$$

- $\varphi[\bar{t}/\bar{x}]$ stands for the formula that results by (simultaneously) replace every *free* occurrence of each x_i in φ by t_i.

  $$\exists z(x * y = 2 * z)[y+1, x-1/x, y] = \exists z((y+1) * (x-1) = 2 * z)$$

- A formula is valid if it is true in any model (i.p. for any value of its variables).

Axiomatic Semantics

- Hoare triple $\{\varphi\}P\{\psi\}$ says that if program P is started in (a state satisfying) φ, then P terminates (if it does) in a state that satisfies ψ.

$$\{m \leq n\}\ \mathtt{j} := (\mathtt{m} + \mathtt{n})/2\ \{m \leq j \leq n\}$$

$$\{False\}\ \mathtt{skip}\ \{\forall n((n > 2 \land even(n)) \rightarrow \exists x \exists y(prime(x) \land prime(y) \land n = x + y))\}$$

Hoare Formal System

$$\frac{}{\{\psi[\overline{t}/\overline{x}]\}\ \overline{x} := \overline{t}\ \{\psi\}} \qquad \frac{\varphi \to \psi}{\{\varphi\}\texttt{skip}\{\psi\}} \qquad \frac{\varphi \to \varphi_1 \quad \{\varphi_1\}P\{\psi\}}{\{\varphi\}P\{\psi\}}$$

$$\frac{\{\varphi\}P_1\{\alpha\} \quad \{\alpha\}P_2\{\psi\}}{\{\varphi\}P_1; P_2\{\psi\}}$$

$$\frac{\{\varphi \land b\}P_1\{\psi\} \quad \{\varphi \land \neg b\}P_2\{\psi\}}{\{\varphi\}\texttt{if}\ b\ \texttt{then}\ P_1\ \texttt{else}\ P_2\ \{\psi\}}$$

$$\frac{\{Inv \land b\}P\{Inv\} \quad (Inv \land \neg b) \to \psi}{\{Inv\}\texttt{while}\ b\ \texttt{do}\ P\{\psi\}}$$

$\{x \geq 0\}$

```
r := 0;
while (r+1)*(r+1) ≤ x
      //invariant ???
      do
      r := r+1
```

$\{r*r \leq x < (r+1)*(r+1)\}$

$\{x \geq 0\}$

```
r := 0;
while (r+1)*(r+1) ≤ x
      //invariant ???
      do
      r := r+1
```

$\{\mathtt{r*r} \leq \mathtt{x} < (\mathtt{r}+1)*(\mathtt{r}+1)\}$

$\mathtt{r} \leq \sqrt{\mathtt{x}} < \mathtt{r}+1$

$\{x \geq 0\}$

```
r := 0;
while (r+1)*(r+1) ≤ x
      //invariant ???
      do
      r := r+1
```

$\{\mathtt{r{*}r} \leq \mathtt{x} < (\mathtt{r}+1) * (\mathtt{r}+1)\}$

$\mathtt{r} \leq \sqrt{\mathtt{x}} < \mathtt{r}+1$

x	r	$(r+1)*(r+1) \leq x$
13	0	$1 * 1 \leq 13$
13	1	$2 * 2 \leq 13$
13	2	$3 * 3 \leq 13$
13	3	$4 * 4 > 13$

$\{x \geq 0\}$

```
r := 0;
while (r+1)*(r+1) ≤ x
      //invariant ???
      do
      r := r+1
```

$\{\mathtt{r*r} \leq \mathtt{x} < (\mathtt{r}+1) * (\mathtt{r}+1)\}$

$\mathtt{r} \leq \sqrt{\mathtt{x}} < \mathtt{r}+1$

x	r	$(r+1)*(r+1) \leq x$
13	0	$1*1 \leq 13$
13	1	$2*2 \leq 13$
13	2	$3*3 \leq 13$
13	3	$4*4 > 13$

invariant $\mathtt{r}*\mathtt{r} \leq \mathtt{x}$

$\{ \mathsf{x} \geq 0 \}$
$\mathsf{r} := 0;$ `while` $(\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x}$ do $\mathsf{r} := \mathsf{r}+1$
$\{ \mathsf{r} * \mathsf{r} \leq \mathsf{x} < (\mathsf{r}+1)*(\mathsf{r}+1) \}$

{ x ≥ 0 }
r := 0; `while` (r+1)∗(r+1) ≤ x do r := r+1
{ r ∗ r ≤ x < (r+1)∗(r+1) }

$$\frac{\{\varphi\}P_1\{\alpha\} \quad \{\alpha\}P_2\{\psi\}}{\{\varphi\}P_1;P_2\{\psi\}}$$

{ (0∗0 =) 0 ≤ x } r := 0; { r∗r ≤ x }

{ r∗r ≤ x }
`while` (r+1)∗(r+1) ≤ x do r := r+1
{ r∗r ≤ x < (r+1)∗(r+1) }

$\{\ x \geq 0\ \}$
$r := 0;\ \texttt{while}\ (r+1)*(r+1) \leq x\ \ \text{do}\ \ r := r+1$
$\{\ r * r \leq x < (r+1)*(r+1)\ \}$

$$\frac{\{\varphi\}P_1\{\alpha\} \quad \{\alpha\}P_2\{\psi\}}{\{\varphi\}P_1;P_2\{\psi\}}$$

$\{\ (0*0 =)\ 0 \leq x\ \}\ r := 0;\ \{\ r*r \leq x\ \}$

$$\frac{}{\{\psi[\overline{t}/\overline{x}]\}\ \overline{x} := \overline{t}\ \{\psi\}}$$

$\{\ r*r \leq x\ \}$
$\texttt{while}\ (r+1)*(r+1) \leq x\ \ \text{do}\ \ r := r+1$
$\{\ r*r \leq x < (r+1)*(r+1)\ \}$

{ x ≥ 0 }
r := 0; while (r+1)∗(r+1) ≤ x do r := r+1
{ r ∗ r ≤ x < (r+1)∗(r+1) }

$$\frac{\{\varphi\}P_1\{\alpha\} \quad \{\alpha\}P_2\{\psi\}}{\{\varphi\}P_1;P_2\{\psi\}}$$

{ (0∗0 =) 0 ≤ x } r := 0; { r∗r ≤ x }

$$\frac{}{\{\psi[\overline{t}/\overline{x}]\}\ \overline{x} := \overline{t}\ \{\psi\}}$$

{ r∗r ≤ x }
while (r+1)∗(r+1) ≤ x do r := r+1
{ r∗r ≤ x < (r+1)∗(r+1) }

$$\frac{\{Inv \wedge b\}P\{Inv\} \quad (Inv \wedge \neg b) \rightarrow \psi}{\{Inv\}\texttt{while}\ b\ \texttt{do}\ P\{\psi\}}$$

{ r∗r ≤ x ∧ (r+1)∗(r+1) ≤ x }
r := r+1;
{ r∗r ≤ x }

(r∗r ≤ x ∧¬((r+1)∗(r+1) ≤ x) →
r ∗ r ≤ x < (r+1)∗(r+1)

$\{ \text{x} \geq 0 \}$
$\text{r} := 0;\ \texttt{while}\ (\text{r}+1)*(\text{r}+1) \leq \text{x}\ \ \text{do}\ \ \text{r} := \text{r}+1$
$\{ \text{r} * \text{r} \leq \text{x} < (\text{r}+1)*(\text{r}+1) \}$

$$\frac{\{\varphi\}P_1\{\alpha\} \quad \{\alpha\}P_2\{\psi\}}{\{\varphi\}P_1;P_2\{\psi\}}$$

$\{ (0*0 =)\ 0 \leq \text{x} \}\ \text{r} := 0;\ \{ \text{r}*\text{r} \leq \text{x} \}$

$$\frac{}{\{\psi[\overline{t}/\overline{x}]\}\ \overline{x} := \overline{t}\ \{\psi\}}$$

$\{ \text{r}*\text{r} \leq \text{x} \}$
$\texttt{while}\ (\text{r}+1)*(\text{r}+1) \leq \text{x}\ \ \text{do}\ \ \text{r} := \text{r}+1$
$\{ \text{r}*\text{r} \leq \text{x} < (\text{r}+1)*(\text{r}+1) \}$

$$\frac{\{Inv \wedge b\}P\{Inv\} \quad (Inv \wedge \neg b) \rightarrow \psi}{\{Inv\}\texttt{while}\ b\ \texttt{do}\ P\{\psi\}}$$

$\{ \text{r}*\text{r} \leq \text{x} \wedge (\text{r}+1)*(\text{r}+1) \leq \text{x} \}$
$\text{r} := \text{r}+1;$
$\{ \text{r}*\text{r} \leq \text{x} \}$

$$\boxed{(\text{r}*\text{r} \leq \text{x} \wedge \neg((\text{r}+1)*(\text{r}+1) \leq \text{x}) \rightarrow \text{r} * \text{r} \leq \text{x} < (\text{r}+1)*(\text{r}+1)}$$

$$\frac{\varphi \rightarrow \varphi_1 \quad \{\varphi_1\}P\{\psi\}}{\{\varphi\}P\{\psi\}}$$

$\{ (\text{r}+1)*(\text{r}+1) \leq \text{x} \}\ \text{r} := \text{r}+1;\ \{ \text{r}*\text{r} \leq \text{x} \}$

$$\boxed{(\text{r}*\text{r} \leq \text{x} \wedge (\text{r}+1)*(\text{r}+1) \leq \text{x}) \rightarrow (\text{r}+1)*(\text{r}+1) \leq \text{x}}$$

$\{\ x \geq 0\ \}$
$r := 0;\ \texttt{while}\ (r+1)*(r+1) \leq x\ \ \texttt{do}\ \ r := r+1$
$\{\ r*r \leq x < (r+1)*(r+1)\ \}$

$$\frac{\{\varphi\}P_1\{\alpha\} \quad \{\alpha\}P_2\{\psi\}}{\{\varphi\}P_1; P_2\{\psi\}}$$

$\{\ (0*0 =)\ 0 \leq x\ \}\ r := 0;\ \{\ r*r \leq x\ \}$

$$\frac{}{\{\psi[\overline{t}/\overline{x}]\}\ \overline{x} := \overline{t}\ \{\psi\}}$$

$\{\ r*r \leq x\ \}$
$\texttt{while}\ (r+1)*(r+1) \leq x\ \ \texttt{do}\ \ r := r+1$
$\{\ r*r \leq x < (r+1)*(r+1)\ \}$

$$\frac{\{Inv \land b\}P\{Inv\} \quad (Inv \land \neg b) \rightarrow \psi}{\{Inv\}\texttt{while}\ b\ \texttt{do}\ P\{\psi\}}$$

$\{\ r*r \leq x \land (r+1)*(r+1) \leq x\ \}$
$r := r+1;$
$\{\ r*r \leq x\ \}$

$$\boxed{(r*r \leq x \land \neg((r+1)*(r+1) \leq x) \rightarrow r*r \leq x < (r+1)*(r+1)}$$

$$\frac{\varphi \rightarrow \varphi_1 \quad \{\varphi_1\}P\{\psi\}}{\{\varphi\}P\{\psi\}}$$

$\{\ (r+1)*(r+1) \leq x\ \}\ r := r+1;\ \{\ r*r \leq x\ \}$

$$\frac{}{\{\psi[\overline{t}/\overline{x}]\}\ \overline{x} := \overline{t}\ \{\psi\}}$$

$$\boxed{(r*r \leq x \land (r+1)*(r+1) \leq x) \rightarrow (r+1)*(r+1) \leq x}$$

From Hoare Logic to VCG

"Thus the practice of proving programs would seem to lead to solution of three of the most pressing problems in software and programming, namely, reliability, documentation, and compatibility. However, program proving, certainly at present, will be difficult even for programmers of high caliber; and may be applicable only to quite simple program designs."

C. Antony R. Hoare
An axiomatic basis for computer programming
1969

Dijkstra Weakest Precondition

The idea firstly appears in the paper *Guarded commands, nondeterminacy and formal derivation of programs*, E.W. Dijkstra, 1975.

- Hoare Logic:
  Given a precondition φ, a code fragment P and a postcondition ψ, is $\{\varphi\}P\{\psi\}$ true?
- Dijkstra Weakest Precondition:
  Given a code fragment P and postcondition ψ, find the unique formula $\mathsf{wp}(P,\psi)$ which is the *weakest* precondition for P and ψ.

> weakest: *$\varphi \rightarrow \mathsf{wp}(P,\psi)$ is valid for any φ such that $\{\varphi\}P\{\psi\}$ is true.*

Dijkstra Weakest Precondition

The idea firstly appears in the paper *Guarded commands, nondeterminacy and formal derivation of programs*, E.W. Dijkstra, 1975.

- Hoare Logic:
  Given a precondition φ, a code fragment P and a postcondition ψ, is $\{\varphi\}P\{\psi\}$ true?
- Dijkstra Weakest Precondition:
  Given a code fragment P and postcondition ψ, find the unique formula $\mathsf{wp}(P, \psi)$ which is the *weakest* precondition for P and ψ.

weakest: *$\varphi \to \mathsf{wp}(P, \psi)$ is valid for any φ such that $\{\varphi\}P\{\psi\}$ is true.*

Each experienced mathematician knows that achievements depend critically on the availability of suitable notations.

E.W. Dijkstra, *My hopes of computing science*, 1979

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$

For example:

$$\mathsf{wp}(\mathtt{x}, \mathtt{y} := \mathtt{y} + \mathtt{1}, \mathtt{x} - \mathtt{1},\ x * y = 0)$$
$$= (y + 1) * (x - 1) = 0$$

- $\texttt{wp}(\overline{\texttt{x}} := \overline{\texttt{t}},\ \psi) = \psi[\overline{\texttt{t}}/\overline{\texttt{x}}]$

For example:

$\texttt{wp}(\texttt{x}, \texttt{y} := \texttt{y} + \texttt{1}, \texttt{x} - \texttt{1},\ x * y = 0)$
$= (y + 1) * (x - 1) = 0$
$= (y = -1) \vee (x = 1)$

- $\mathtt{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$

For example:

$\mathtt{wp}(\mathtt{x}, \mathtt{y} := \mathtt{y} + \mathtt{1}, \mathtt{x} - \mathtt{1},\ x * y = 0)$
$= (y + 1) * (x - 1) = 0$
$= (y = -1) \vee (x = 1)$ is weaker than $x = 1$

- $\mathrm{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$

For example:

$\mathrm{wp}(\mathtt{x}, \mathtt{y} := \mathtt{y} + \mathtt{1}, \mathtt{x} - \mathtt{1},\ x * y = 0)$
$= (y + 1) * (x - 1) = 0$
$= (y = -1) \lor (x = 1)$ is weaker than $x = 1$
is weaker than $(y = -1) \land (x = 1)$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$

For example:

$\mathsf{wp}(\mathtt{x:=y+1;\ \ y:=x-1},\ x * y = 0)$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$

For example:

$$\begin{aligned}
&\ \mathsf{wp}(\mathtt{x{:=}y{+}1;\ \ y{:=}x{-}1},\ x * y = 0) \\
=&\ \mathsf{wp}(\mathtt{x{:=}y{+}1},\ \mathsf{wp}(\mathtt{y{:=}x{-}1},\ x * y = 0))
\end{aligned}$$

- $\text{wp}(\overline{\texttt{x}} := \overline{\texttt{t}},\ \psi) = \psi[\overline{\texttt{t}}/\overline{\texttt{x}}]$
- $\text{wp}(\texttt{P}_1;\texttt{P}_2,\ \psi) = \text{wp}(\texttt{P}_1,\ \text{wp}(\texttt{P}_2,\ \psi))$

For example:

$$
\begin{aligned}
&\text{wp}(\texttt{x:=y+1;\ \ y:=x-1},\ x * y = 0) \\
= {} &\text{wp}(\texttt{x:=y+1},\ \text{wp}(\texttt{y:=x-1},\ x * y = 0)) \\
= {} &\text{wp}(\texttt{x:=y+1},\ x * (x - 1) = 0)
\end{aligned}
$$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P_1};\mathtt{P_2},\ \psi) = \mathsf{wp}(\mathtt{P_1},\ \mathsf{wp}(\mathtt{P_2},\ \psi))$

For example:

$$
\begin{aligned}
& \mathsf{wp}(\mathtt{x:=y+1;\ \ y:=x-1},\ x * y = 0) \\
= \ & \mathsf{wp}(\mathtt{x:=y+1},\ \mathsf{wp}(\mathtt{y:=x-1},\ x * y = 0)) \\
= \ & \mathsf{wp}(\mathtt{x:=y+1},\ x * (x - 1) = 0) \\
= \ & (y + 1) * y = 0
\end{aligned}
$$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$

For example:

$$
\begin{aligned}
& \mathsf{wp}(\mathtt{x:=y+1;\ \ y:=x-1},\ x * y = 0) \\
= {} & \mathsf{wp}(\mathtt{x:=y+1},\ \mathsf{wp}(\mathtt{y:=x-1},\ x * y = 0)) \\
= {} & \mathsf{wp}(\mathtt{x:=y+1},\ x * (x - 1) = 0) \\
= {} & (y + 1) * y = 0 \\
= {} & (y = -1) \vee (y = 0)
\end{aligned}
$$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\, \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P_1};\mathtt{P_2},\, \psi) = \mathsf{wp}(\mathtt{P_1},\, \mathsf{wp}(\mathtt{P_2},\, \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P_1}\ \mathtt{else}\ \mathtt{P_2},\, \psi) = (b \to \mathsf{wp}(\mathtt{P_1},\, \psi)) \wedge (\neg b \to \mathsf{wp}(\mathtt{P_2},\, \psi))$

For example:

$\mathsf{wp}(\mathtt{if\ x} \geq \mathtt{y\ then\ z\!:=\!x\ else\ z\!:=\!y},\, z = max(x, y))$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \to \mathsf{wp}(\mathtt{P}_1,\ \psi)) \wedge (\neg b \to \mathsf{wp}(\mathtt{P}_2,\ \psi))$

For example:

$$\mathsf{wp}(\mathtt{if}\ \mathtt{x} \geq \mathtt{y}\ \mathtt{then}\ \mathtt{z{:=}x}\ \mathtt{else}\ \mathtt{z{:=}y},\ z = max(x,y))$$
$$= (x \geq y \to \mathsf{wp}(\mathtt{z{:=}x},\ z = max(x,y))) \wedge$$
$$(\neg(x \geq y) \to \mathsf{wp}(\mathtt{z{:=}y},\ z = max(x,y)))$$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\, \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\, \psi) = \mathsf{wp}(\mathtt{P}_1,\, \mathsf{wp}(\mathtt{P}_2,\, \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\, \psi) = (b \rightarrow \mathsf{wp}(\mathtt{P}_1,\, \psi)) \wedge (\neg b \rightarrow \mathsf{wp}(\mathtt{P}_2,\, \psi))$

For example:

$$
\begin{aligned}
&\mathsf{wp}(\mathtt{if\ x} \geq \mathtt{y\ then\ z{:=}x\ else\ z{:=}y},\ z = max(x,y)) \\
&= (x \geq y \rightarrow \mathsf{wp}(\mathtt{z{:=}x},\ z = max(x,y))) \wedge \\
&\quad (\neg(x \geq y) \rightarrow \mathsf{wp}(\mathtt{z{:=}y},\ z = max(x,y))) \\
&= (x \geq y \rightarrow x = max(x,y)) \wedge (\neg(x \geq y) \rightarrow y = max(x,y))
\end{aligned}
$$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1; \mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \rightarrow \mathsf{wp}(\mathtt{P}_1,\ \psi)) \wedge (\neg b \rightarrow \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{skip},\ \psi) = \psi$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \to \mathsf{wp}(\mathtt{P}_1,\ \psi)) \wedge (\neg b \to \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{skip},\ \psi) = \psi$
- $\mathsf{wp}(\mathtt{while}\ b\ \mathsf{do}\ \mathtt{P}\ ,\ \psi) = \alpha$ provided that
  - $(\alpha \wedge b) \to \mathsf{wp}(\mathtt{P},\alpha)$
  - $(\alpha \wedge \neg b) \to \psi$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \rightarrow \mathsf{wp}(\mathtt{P}_1,\ \psi)) \wedge (\neg b \rightarrow \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{skip},\ \psi) = \psi$
- $\mathsf{wp}(\mathtt{while}\ b\ \mathsf{do}\ \mathtt{P}\ ,\ \psi) = \alpha$ provided that
  - $(\alpha \wedge b) \rightarrow \mathsf{wp}(\mathtt{P},\alpha)$
  - $(\alpha \wedge \neg b) \rightarrow \psi$

$$\mathsf{wp}(\mathtt{while}\ (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x}\ \mathsf{do}\ \mathsf{r} := \mathsf{r}+1,\ \mathsf{r} * \mathsf{r} \leq \mathsf{x} < (\mathsf{r}+1)*(\mathsf{r}+1)) = \mathsf{r}*\mathsf{r} \leq \mathsf{x} \text{ provided that}$$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \to \mathsf{wp}(\mathtt{P}_1,\ \psi)) \wedge$
  $(\neg b \to \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{skip},\ \psi) = \psi$
- $\mathsf{wp}(\mathtt{while}\ b\ \mathtt{do}\ \mathtt{P}\ ,\ \psi) = \alpha$ provided that
  - $(\alpha \wedge b) \to \mathsf{wp}(\mathtt{P},\alpha)$
  - $(\alpha \wedge \neg b) \to \psi$

$\mathsf{wp}(\mathtt{while}\ (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x}\ \mathsf{do}\ \mathsf{r} := \mathsf{r}+1,$
$\mathsf{r} * \mathsf{r} \leq \mathsf{x} < (\mathsf{r}+1)*(\mathsf{r}+1)) = \mathsf{r}*\mathsf{r} \leq \mathsf{x}$ provided that

- $(\mathsf{r}*\mathsf{r} \leq \mathsf{x}\ \wedge (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x}) \to \mathsf{wp}(\mathsf{r}:=\mathsf{r}+1,\ \mathsf{r}*\mathsf{r} \leq \mathsf{x})$
- $(\ \mathsf{r}*\mathsf{r} \leq \mathsf{x} \wedge\neg(\ (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x})) \to \mathsf{r} * \mathsf{r} \leq \mathsf{x} < (\mathsf{r}+1)*(\mathsf{r}+1)$

- $\mathrm{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathrm{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathrm{wp}(\mathtt{P}_1,\ \mathrm{wp}(\mathtt{P}_2,\ \psi))$
- $\mathrm{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \to \mathrm{wp}(\mathtt{P}_1,\ \psi)) \land (\neg b \to \mathrm{wp}(\mathtt{P}_2,\ \psi))$
- $\mathrm{wp}(\mathtt{skip},\ \psi) = \psi$
- $\mathrm{wp}(\mathtt{while}\ b\ \mathtt{do}\ \mathtt{P}\ ,\ \psi) = \alpha$ provided that
  - $(\alpha \land b) \to \mathrm{wp}(\mathtt{P},\alpha)$
  - $(\alpha \land \neg b) \to \psi$

$\mathrm{wp}(\mathtt{while}\ (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x}\ \mathsf{do}\ \mathsf{r} := \mathsf{r}+1,\ \mathsf{r} * \mathsf{r} \leq \mathsf{x} < (\mathsf{r}+1)*(\mathsf{r}+1)) = \mathsf{r}*\mathsf{r} \leq \mathsf{x}$ provided that

- $(\mathsf{r}*\mathsf{r} \leq \mathsf{x}\ \land (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x}) \to (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x}$
- $(\ \mathsf{r}*\mathsf{r} \leq \mathsf{x} \land\neg(\ (\mathsf{r}+1)*(\mathsf{r}+1) \leq \mathsf{x})) \to \mathsf{r} * \mathsf{r} \leq \mathsf{x} < (\mathsf{r}+1)*(\mathsf{r}+1)$

- $\mathsf{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\mathsf{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathsf{wp}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \to \mathsf{wp}(\mathtt{P}_1,\ \psi)) \wedge (\neg b \to \mathsf{wp}(\mathtt{P}_2,\ \psi))$
- $\mathsf{wp}(\mathtt{skip},\ \psi) = \psi$
- $\mathsf{wp}(\mathtt{while}\ b\ \mathtt{do}\ \mathtt{P}\ ,\ \psi) = \alpha$ provided that
  - $(\alpha \wedge b) \to \mathsf{wp}(\mathtt{P},\alpha)$
  - $(\alpha \wedge \neg b) \to \psi$

- $\text{wp}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \psi[\overline{\mathtt{t}}/\overline{\mathtt{x}}]$
- $\text{wp}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \text{wp}(\mathtt{P}_1,\ \text{wp}(\mathtt{P}_2,\ \psi))$
- $\text{wp}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = (b \to \text{wp}(\mathtt{P}_1,\ \psi)) \wedge (\neg b \to \text{wp}(\mathtt{P}_2,\ \psi))$
- $\text{wp}(\mathtt{skip},\ \psi) = \psi$
- $\text{wp}(\mathtt{while}\ b\ \mathtt{do}\ \mathtt{P}\ ,\ \psi) = \alpha$ provided that
  - $(\alpha \wedge b) \to \text{wp}(\mathtt{P},\alpha)$
  - $(\alpha \wedge \neg b) \to \psi$

$\{\varphi\}P\{\psi\}$ iff

- $\varphi \to \text{wp}(P,\psi)$ and
- all the provisos for calculating $\text{wp}(P,\psi)$

are valid (implications) formulas.

Verification Condition Generation

$$\mathsf{VCG}(\{\varphi\}P\{\psi\}) = \{\ \varphi \rightarrow \mathsf{wp}(P,\psi)\ \} \cup \mathbf{vc}(P,\psi)$$

where

- $\mathbf{vc}(\overline{\mathtt{x}} := \overline{\mathtt{t}},\ \psi) = \mathbf{vc}(\mathtt{skip},\ \psi) = \emptyset$
- $\mathbf{vc}(\mathtt{P}_1;\mathtt{P}_2,\ \psi) = \mathbf{vc}(\mathtt{P}_1,\ \mathsf{wp}(\mathtt{P}_2,\ \psi)) \cup \mathbf{vc}(\mathtt{P}_2, \psi)$
- $\mathbf{vc}(\mathtt{if}\ b\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2,\ \psi) = \mathbf{vc}(\mathtt{P}_1,\ \psi) \cup \mathbf{vc}(\mathtt{P}_2,\ \psi)$
- $\mathbf{vc}(\mathtt{while}\ b\ \mathtt{do}\ \mathtt{P},\ \psi) =$

$$\{\ (Inv \wedge b) \rightarrow \mathsf{wp}(\mathtt{P},Inv),\ (Inv \wedge \neg b) \rightarrow \psi\ \} \cup \mathbf{vc}(P,Inv)$$

Inv is the (inferred/user-defined) invariant of the iteration `while` b `do P`

VCG computes the set of all the FOL-implications of a Hoare proof-tree.

VCG ($\{ x \geq 0 \}$
$r := 0;$
`while` $(r+1)*(r+1) \leq x$ do $r := r+1$
$\{ r * r \leq x < (r+1)*(r+1) \}$)

$=$

$\{ (r*r \leq x \wedge (r+1)*(r+1) \leq x) \rightarrow (r+1)*(r+1) \leq x$,
$(r*r \leq x \wedge \neg((r+1)*(r+1) \leq x)) \rightarrow r * r \leq x < (r+1)*(r+1) \}$

RootApprox.dfy

From VCG to Current Deductive Verification Tools

- Hoare's Verification Grand Challenge:
  - *This contribution ... revives an old challenge: the construction and application of a verifying compiler that guarantees correctness of a program before running it.*
  - *Correctness of computer programs is the fundamental concern of the theory of programming and of its application in large-scale software engineering.*

Tony Hoare
The Verifying Compiler: A Grand Challenge for Computing Research
2003

- Deductive Verification:
  - Logical reasoning (deduction) is used to prove properties.
  - Expressive (at least first-order) logic.
  - Contract-based specifications (standard approach)

- Deductive Verification:
  - Logical reasoning (deduction) is used to prove properties.
  - Expressive (at least first-order) logic.
  - Contract-based specifications (standard approach)
- Arquitectures in deductive verification:
  1. On top of interactive proof assistants

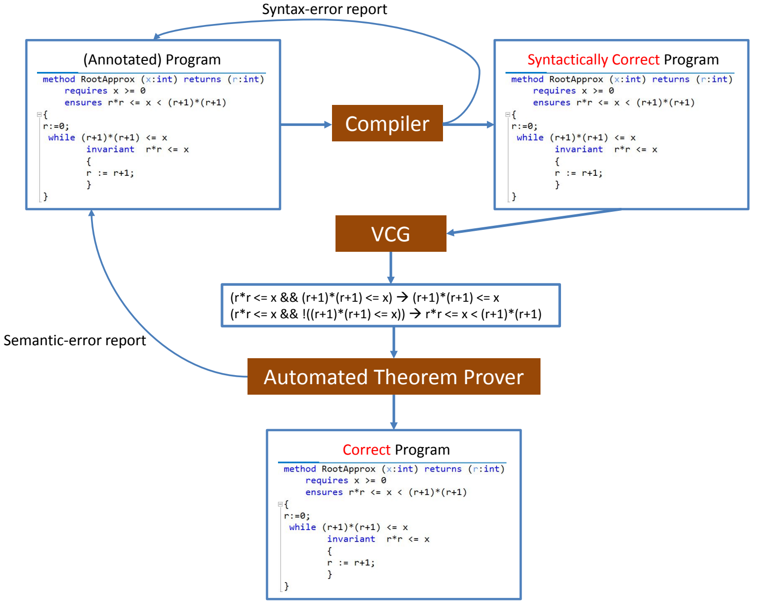
     Isabelle/HOL, Coq, HOL Ligth, PVS, ...
  2. Automatic Program Verifiers

- Deductive Verification:
  - Logical reasoning (deduction) is used to prove properties.
  - Expressive (at least first-order) logic.
  - Contract-based specifications (standard approach)
- Arquitectures in deductive verification:
  1. On top of interactive proof assistants

     Isabelle/HOL, Coq, HOL Ligth, PVS, ...
  2. Automatic Program Verifiers
     1. Program logics for a specific target language

        ACL2, KeY, KIV, VeriFun, ...
     2. VCG + Automatic theorem provers (SMT-solver)

        Spark, Verifast, Dafny, Why, Frama-C, ...


Syntax-error report
(Annotated) Program
method RootApprox (x:int) returns (r:int)
requires x >= 0
ensures r*r <= x < (r+1)*(r+1)
{
r:=0;
while (r+1)*(r+1) <= x
invariant r*r <= x
{
r := r+1;
}
}
Compiler
Syntactically Correct Program
method RootApprox (x:int) returns (r:int)
requires x >= 0
ensures r*r <= x < (r+1)*(r+1)
{
r:=0;
while (r+1)*(r+1) <= x
invariant r*r <= x
{
r := r+1;
}
}
VCG
(r*r <= x && (r+1)*(r+1) <= x) → (r+1)*(r+1) <= x
(r*r <= x && !((r+1)*(r+1) <= x)) → r*r <= x < (r+1)*(r+1)
Semantic-error report
Automated Theorem Prover
Correct Program
method RootApprox (x:int) returns (r:int)
requires x >= 0
ensures r*r <= x < (r+1)*(r+1)
{
r:=0;
while (r+1)*(r+1) <= x
invariant r*r <= x
{
r := r+1;
}
}

Example: A more interesting invariant

Binary Search

"The future looks bright for the collaboration of verification and reasoning. Recent advances in both fields and increasingly tight interaction have already given rise to industrially relevant verification tools. We predict that this is only the beginning, and that within a decade tools based on verification technology will be as useful and widespread for software development as they are today in the hardware domain."

Bernhard Beckert & Reiner Hähnle
Reasoning and Verification: State of the Art and Current Trends[1]
IEEE Intelligent System
2014

[1]This paper provides a representative selection of 27 verification systems.

Timsort is broken by S. de Gouw, F. de Boer, and J. Rot (2015)

Tim Peters developed the Timsort hybrid sorting algorithm in 2002. It is a clever combination of ideas from merge sort and insertion sort, and designed to perform well on real world data. TimSort was first developed for Python, but later ported to Java (where it appears as java.util.Collections.sort and java.util.Arrays.sort) by Joshua Bloch (the designer of Java Collections who also pointed out that most binary search algorithms were broken). TimSort is today used as the default sorting algorithm for Android SDK, Sun's JDK and OpenJDK. Given the popularity of these platforms this means that the number of computers, cloud services and mobile phones that use TimSort for sorting is well into the billions.

Fast forward to 2015. After we had successfully verified Counting and Radix sort implementations in Java (J. Autom. Reasoning 53(2), 129-139) with a formal verification tool called KeY, we were looking for a new challenge. TimSort seemed to fit the bill, as it is rather complex and widely used. Unfortunately, we weren't able to prove its correctness. A closer analysis showed that this was, quite simply, because TimSort was broken and our theoretical considerations finally led us to a path towards finding the bug (interestingly, that bug appears already in the Python implementation). This blog post shows how we did it. Continue reading →

- Dafny is an automatic program verifier (VCG + SMT-solver).
- Dafny is being developed by Microsoft Research.

  Dafny 1.9.8 (August 31, 2016) is the 14th stable release, since Oct 30, 2012.

- Dafny encourages using best-practice programming styles, in particular the design by contract approach.
- Dafny provides
  - Design-time feedback
  - Fluid interaction

  for accessible integrated verification.
- Dafny generates executable (.NET) code, omitting specification (ghost) constructs.

Using **assert** and **assume** in software development

- **assert** φ
  - Dafny first tries to prove φ, and if successful, then φ can be used in the rest of proof.
  - Provides a non-instantiable lemma φ:
    A property that is previously and separately proved and helps to prove other properties.

Using **assert** and **assume** in software development

- **assert** φ
  - Dafny first tries to prove φ, and if successful, then φ can be used in the rest of proof.
  - Provides a non-instantiable lemma φ:
    A property that is previously and separately proved and helps to prove other properties.
- **assume** φ
  - Dafny assumes that φ is true: without proving φ it is enabled to use φ in the current proof.
  - Dafny does not consider verified any file with one **assume**.

Using **assert** and **assume** in software development

- **assert** φ
  - Dafny first tries to prove φ, and if successful, then φ can be used in the rest of proof.
  - Provides a non-instantiable lemma φ:
    A property that is previously and separately proved and helps to prove other properties.
- **assume** φ
  - Dafny assumes that φ is true: without proving φ it is enabled to use φ in the current proof.
  - Dafny does not consider verified any file with one **assume**.
- In verified software development:
  - **assume** φ for checking if φ is the required property;
  - If OK then change from **assume** to **assert**;
  - If "assertion violation" then φ must be proved in a **lemma** or some previous **assert**(s) must be inserted.

Example: The development of a verified quicksort

QuickSort

The beauty of a theorem from mathematics,
the preciseness of an inference rule in logic,
the intrigue of a puzzle,
and the challenge of a game – all are present
in the field of automated reasoning.
(Larry Wos, 1988)