Program Correctness

Literatuur

Verification of Sequential and Concurrent Programs.
Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog.
Series: Texts in Computer Science. Springer.
3rd ed. 2nd Printing.
ISBN: 978-1-84882-744-8.

Te behandelen stof

Course Towards Object-Oriented Program Verification
(zie *Preface*: *Outlines of One-Semester Courses* en slides).
Uit bovenstaand boek behandelen we de hoofdstukken 2, 3, 4, en 5 onderverdeeld in de volgende blokken B1-3:

	Onderwerp	Secties
B1	Partiële Correctheid While Programma's	2.1, 2.2, 2.4, 2.5, 2.7, 3.1, 3.3, 3.4, 3.10, 3.11.
B2	Totale Correctheid While Programma's	3.3 en 3.4.
B3	Partiële Correctheid Recursieve Programma's	4.1, 4.3, 5.1, 5.2, 5.3.

- The Atlantic

Few programmers write even a rough sketch of what their programs will do before they start coding.

The software did exactly what it was told to do. The reason it failed is that it was told to do the wrong thing.

Software engineers don't understand the problem they are trying to solve, and don't care to.

You could do all the testing you wanted and you'd never find all the bugs.

Industriële/Maatschappelijke Relevantie

'Softwarefouten kosten Nederlandse economie jaarlijks 1,6 miljard euro'

Kosten digitalisering rechtspaak: 220 miljoen euro. Volkskrant: Het digitaliseren van de rechtspraak blijkt een veel grotere opgave dan gedacht.

Wat managers denken/verwachten :

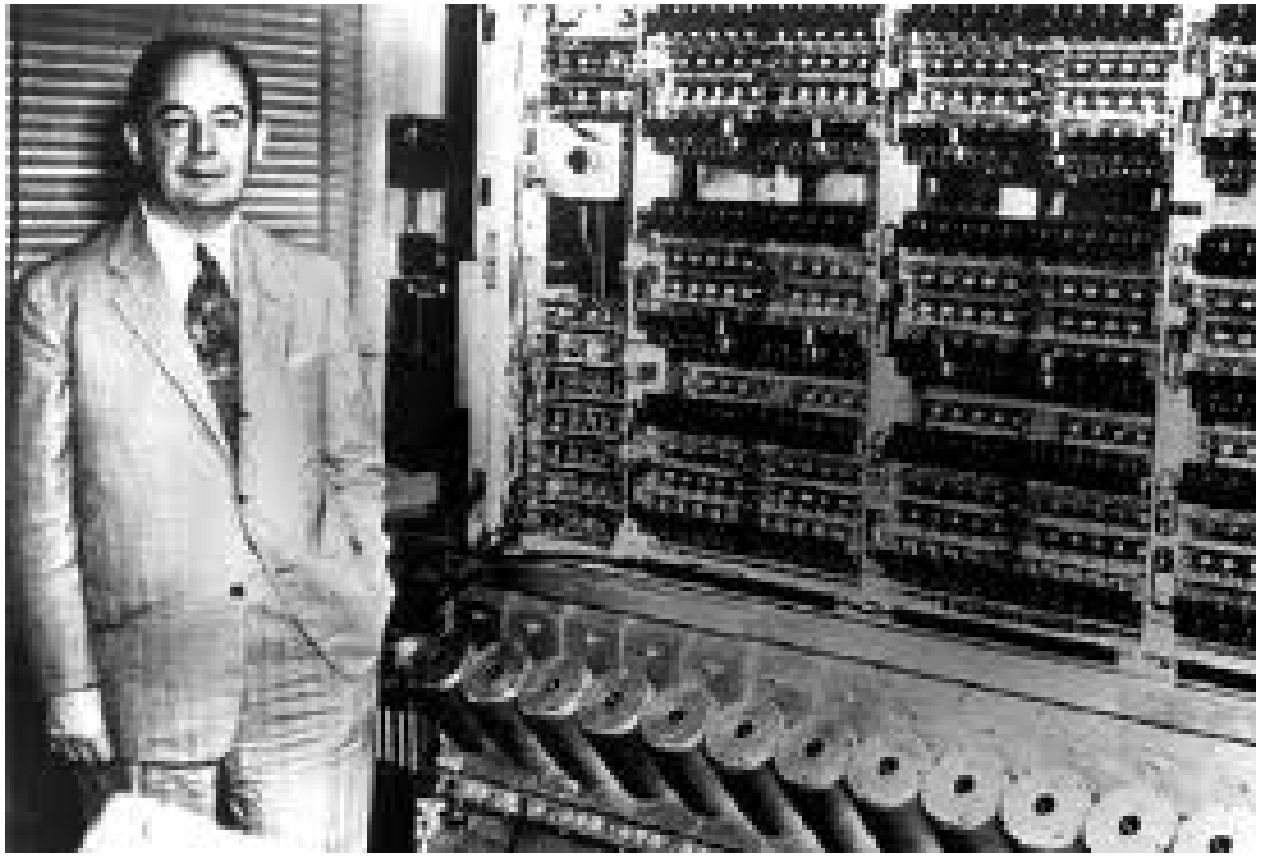
Software development is not an art, and programmers are not artists, despite any claims to the contrary.

Management has come to believe the first and most important misconception: that it is impossible to ship software devoid of errors in a cost-effective way.

Waar Komen Die Bugs Vandaan?

Een imperatief programma beschrijft hoe een probleem door een computer opgelost kan worden.

De Von Neumann Computer Architectuur

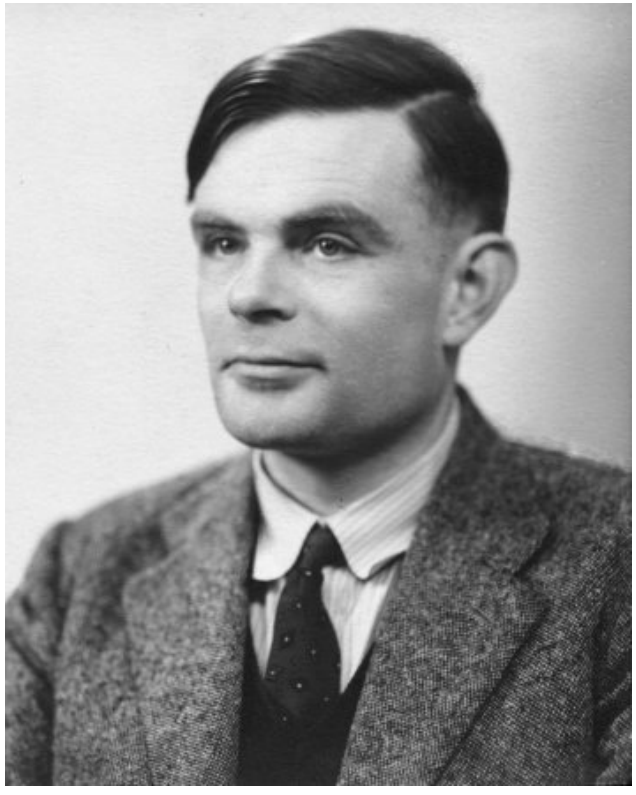


Assembleertalen

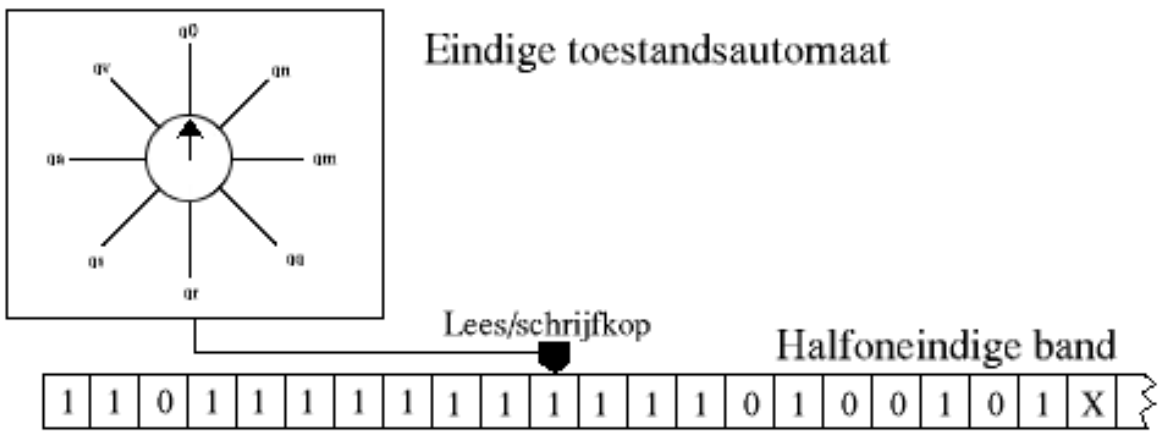
```
; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32   PROC            ; procedure begins here
        CMP   AX,97     ; compare AX to 97
        JL    DONE      ; if less, jump to DONE
        CMP   AX,122    ; compare AX to 122
        JG    DONE      ; if greater, jump to DONE
        SUB   AX,32     ; subtract 32 from AX
DONE:   RET             ; return to main program
SUB32   ENDP            ; procedure ends here
```

FIGURE 17. Assembly language

Founding Father of Computer Science: Alan Turing



De Turing Machiene



Wat Te Doen Met Bugs?

Edsger Dijkstra: Structured Programming

Met "debugging" kun je alleen fouten vinden."

Wat De Hack Wordt Hier Uitgevoerd?

(Initiële waarde van x is groter of gelijk aan 0)

$y := 0; u := 0; v := 1;$
while $u + v \leq x$
do $y := y + 1;$
$u := u + v;$
$v := v + 2$
od

Debugging: Let it Flow

x	y	u	v
13	0	0	1
13	1	1	3
13	2	4	5
13	3	9	7
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Wat is de relatie tuusen de waarden van de variablelen x, y, u and v?

Robert Floyd: Asserties

$$y^2 \leq x < (y+1)^2$$

Specificatie van Programma Correctheid

$$\{p\}S\{q\}$$

where

- S is a (programming) statement
- p and q are assertions
- p is the precondition
- q is the postcondition

Elke terminerende berekening van S
in een begintoestand waarin de preconditie p waar is
resulteert in een eindtoestand die de postcondition q waar maakt.

Specifying Correctness of Assignments

- $\{\textbf{true}\}x := 0\{x = 0\}$
  (Java syntax: $\{\textbf{true}\}x = 0\{x == 0\}$)
- $\{\textbf{true}\}x := y + 1\{x = y + 1\}$
- $\{y = 0\}x := y + 1\{x = 1 \wedge y = 0\}$

Some Exercises

- Does in general $\{\mathbf{true}\}x := e\{x = e\}$ hold, e any side-effect free expression?
- For which precondition p does $\{p\}x := x + 1\{x = y\}$ hold?
- For which precondition p does $\{p\}x := x + 1\{a[x] = 0\}$ hold, where a is an array int[]?
- For which precondition p does $\{p\}x := x + 1\{x = y + 1\}$ hold?
- For which precondition p does $\{p\}a[i] := 0\{a[j] = 0\}$ hold, where a is an array int[]?
- For which precondition p does $\{p\}x := y.val\{x = y.val\}$ hold?
- For which precondition p does $\{p\}x := y \ div \ z\{x = y \ div \ z\}$ hold?
- For which postcondition q does $\{x = \mathbf{null}\}y := x.val\{q\}$ hold?
- For which statements S does $\{\mathbf{true}\}S\{\mathbf{false}\}$ hold?

Specifying Correctness of The Sequential Composition of Statements

- $\{x = y\}x := x + 1; y := y + 1\{x = y\}$
  Question: what holds in between?
- $\{x = q \cdot y + r \wedge r \geq y\}r := r - y; q := q + 1\{x = q \cdot y + r\}$
  Question: what holds in between?

Specifying Correctness of Conditional Statements

$\{\textbf{true}\}$

- **if** $x > y$ **then** $m := x$ **else** $m := y$ **fi**

$\{(m = x \wedge x > y) \vee (m = y \wedge x \leq y)\}$

$\{\textbf{true}\}$

- **if** $y \neq 0$ **then** $z := x \; div \; y$ **else** *skip* **fi**

$\{y \neq 0 \rightarrow z = x \; div \; y\}$

Specifying Correctness of While Statements

- $$\{\textbf{true}\}\textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}\{a[x] = 0\}$$
- $$\{a[n+1] = 0 \wedge \forall i \in [x : n] : \ a[i] \neq 0\}$$
  $$\textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}$$
  $$\{x = n + 1\}$$
- $$\{\forall n : a[n] = b[n]\}\textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}\{\forall n : a[n] = b[n]\}$$

Validating Correctness Formulas

Two methods to validate $\{p\}S\{q\}$:

- Testing: select input values for the program variables which satisfy p, run the S and check upon termination q.
- Verification: Axioms and proof rules.

Syntax of While Programs

S	$::=$	$skip$	skip statement
	$\mid$	$u := t$	assignment
	$\mid$	$S_1; S_2$	sequential composition
	$\mid$	**if** B **then** S_1 **else** S_2 **fi**	choice
	$\mid$	**while** B **do** S_1 **od**	iteration

Example:

$$x := a[i]; a[i] := a[j]; a[j] := x$$

Types

Basic types:

- **integer**,
- **Boolean**.

Higher types:

- $T_1 \times \ldots \times T_n \to T$,
  where
  - $T_1, \ldots, T_n, T$ are basic types.
  - $T_1, \ldots, T_n$ are *argument* types and T is the *value* type.

Variables

We distinguish two sorts of variables:

- *simple* variables (basic type),
- *array* variables or just arrays (higher type).

We denote the set of all simple and array variables by Var.

Constants

- constants of basic type,
- constants of higher type.

Examples:

- $+, -, \cdot, \mathit{min}, \mathit{max}, \mathit{div}, \mathit{mod}$ of type
  $\mathbf{integer} \times \mathbf{integer} \rightarrow \mathbf{integer}$,
- $=, <$ of type
  $\mathbf{integer} \times \mathbf{integer} \rightarrow \mathbf{Boolean}$,
- $\neg$ of type
  $\mathbf{Boolean} \rightarrow \mathbf{Boolean}$,
- $=, \vee, \wedge, \rightarrow, \leftrightarrow$ of type
  $\mathbf{Boolean} \times \mathbf{Boolean} \rightarrow \mathbf{Boolean}$.

Expressions

Expressions are defined by induction as follows:

- a simple variable of type T is an expression of type T,
- a constant of a basic type T is an expression of type T,
- if $s_1, \ldots, s_n$ are expressions of type $T_1, \ldots, T_n$, respectively, and op is a constant of type $T_1 \times \ldots \times T_n \to T$, then $op(s_1, \ldots, s_n)$ is an expression of type T,
- if $s_1, \ldots, s_n$ are expressions of type $T_1, \ldots, T_n$, respectively, and a is an array of type $T_1 \times \ldots \times T_n \to T$, then $a[s_1, \ldots, s_n]$ is an expression of type T,
- if B is a Boolean expression and s_1 and s_2 are expressions of type T, then **if** B **then** s_1 **else** s_2 **fi** is an expression of type T.

Infix notation

$$(s_1 \ op \ s_2)$$

Syntax of Assertions

$$
\begin{array}{lcll}
p & ::= & B & \text{Boolean expression} \\
 & | & (p \wedge q) & \text{conjunction} \\
 & | & \neg p & \text{negation} \\
 & \vdots & & \\
 & | & \exists x : p & \text{quantification}
\end{array}
$$

Example:

$$\forall n : a[n] \leq a[n+1]$$

Axioms for SKIP and Assignment

AXIOM 1: SKIP

$$\{p\}\ \mathit{skip}\ \{p\}$$

AXIOM 2: ASSIGNMENT

$$\{p[u := t]\}\ u := t\ \{p\}$$

Example

$$\{x + 1 = y\}\ x := x + 1\ \{x = y\}$$

Substitution Subscripted Variables

$\{(a[y] = 1)[a[x] := 0]\}\ a[x] := 0\ \{a[y] = 1\}$
Example:

$$
\begin{array}{ll}
(a[y] = 1)[a[x] := 0] & \equiv \\
(a[y])[a[x] := 0] = (1[a[x] := 0]) & \equiv \\
\textbf{if } y[a[x] := 0] = x \textbf{ then } 0 \textbf{ else } a[y[a[x] := 0]] \textbf{ fi} = 1 & \equiv \\
\textbf{if } y = x \textbf{ then } 0 \textbf{ else } a[y] \textbf{ fi} = 1 &
\end{array}
$$

We derive

$$\{\textbf{if } y = x \textbf{ then } 0 \textbf{ else } a[y] \textbf{ fi} = 1\}\ a[x] := 0\ \{a[y] = 1\}$$

Consequence Rule

RULE 6: CONSEQUENCE

$$\frac{p \to p_1, \{p_1\}\ S\ \{q_1\}, q_1 \to q}{\{p\}\ S\ \{q\}}$$

Example: Let $p \equiv$ **if** $y = x$ **then** 0 **else** $a[y]$ **fi** $= 1$.

$$\frac{(y \neq x \wedge a[y] = 1) \to p, \{p\}\ a[x] := 0\ \{a[y] = 1\}, a[y] = 1 \to a[y] = 1}{\{y \neq x \wedge a[y] = 1\}\ a[x] := 0\ \{a[y] = 1\}}$$

Sequential Composition

RULE 3: COMPOSITION

$$\frac{\{p\}\ S_1\ \{r\}, \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$$

Example

$$\frac{\{x+1=y+1\}\ x := x+1\ \{x=y+1\}, \{x=y+1\}\ y := y+1\ \{x=y\}}{\{x+1=y+1\}\ x := x+1;\ y := y+1\ \{x=y\}}$$

Application consequence rule:

$$\{x=y\}\ x := x+1;\ y := y+1\ \{x=y\}$$

since

$$x=y \rightarrow x+1=y+1$$

Correctheid Swap

We willen bewijzen dat

$$\{y = Y \wedge x = X\}x := z; x := y; y := z\{x = Y \wedge y = X\}$$

Bewijs (top-down):
SEQ:

- $\{y = Y \wedge x = X\}z := x; x := y\{x = Y \wedge z = X\}$
- $\{x = Y \wedge z = X\}y := z\{x = Y \wedge y = X\}$

SEQ:

- $\{y = Y \wedge x = X\}z := x\{y = Y \wedge z = X\}$
- $\{y = Y \wedge z = X\}x := y\{x = Y \wedge z = X\}$

Conditional

RULE 4: CONDITIONAL

$$\frac{\{p \wedge B\}\ S_1\ \{q\}, \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$$

Example

$$\frac{\{x \leq y\}\ z := y\ \{z = max(x,y)\}, \{\neg(x \leq y)\}\ z := x\ \{z = max(x,y)\}}{\{\textbf{true}\}\ \textbf{if}\ x \leq y\ \textbf{then}\ z := y\ \textbf{else}\ z := x\ \textbf{fi}\ \{z = max(x,y)\}}$$

Note: in the above premises we have abbreviated $\textbf{true} \wedge x \leq y$ and $\textbf{true} \wedge \neg(x \leq y)$.

Loop

RULE 5: LOOP

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$$

Example

$$\frac{\{x \leq y \wedge x < y\}\ x := x + 1\ \{x \leq y\}}{\{x \leq y\}\ \textbf{while}\ x < y\ \textbf{do}\ x := x + 1\ \textbf{od}\ \{x \leq y \wedge \neg(x < y)\}}$$

Correctheid Integer Wortel

Laat S staan voor

$y := 0; u := 0; v := 1;$
while $u + v \leq x$
do $y := y + 1;$
$\quad u := u + v;$
$\quad v := v + 2$
od

Te bewijzen

$$\{x \geq 0\}\ S\ \{y^2 < x \leq (y+1)^2\}$$

Bewijs

SEQ:

- $\{x \geq 0\}\ y := 0; u := 0; v := 1\ \{x \geq 0 \wedge y = 0 \wedge u = 0 \wedge v = 1\}$
- $\{x \geq 0 \wedge y = 0 \wedge u = 0 \wedge v = 1\}\ W\ \{y^2 \leq x < (y+1)^2\}$

CONS:

- $(x \geq 0 \wedge y = 0 \wedge u = 0 \wedge v = 1) \rightarrow (u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x)$

$\{u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x\}$

- W

$\{u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x \wedge u + v > x\}$

- $(u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x \wedge u + v > x) \rightarrow (y^2 \leq x \leq (y+1)^2)$

LOOP

$\{u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x \wedge u + v \leq x\}$

- $y := y + 1; u := u + v; v := v + 2$

$\{u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x\}$

Straightline Code: Van Achteren Naar Voren

Of, van boven naar beneden:

$$\{u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x\}$$
$$v := v + 2$$
$$\{u = y^2 \wedge v + 2 = 2y + 1 \wedge y^2 \leq x\}$$
$$u := u + v$$
$$\{u + v = y^2 \wedge v + 2 = 2y + 1 \wedge y^2 \leq x\}$$
$$y := y + 1$$
$$\{u + v = (y + 1)^2 \wedge v + 2 = 2(y + 1) + 1 \wedge (y + 1)^2 \leq x\}$$

Tot slot:

$$(u = y^2 \wedge v = 2y + 1 \wedge y^2 \leq x \wedge u + v \leq x)$$
$$\rightarrow$$
$$(u + v = (y + 1)^2 \wedge v + 2 = 2(y + 1) + 1 \wedge (y + 1)^2 \leq x)$$

Correctheid Zero Search

Laat S staan voor **while** $a[x] \neq 0$ **do** $x := x + 1$ **od**.
We willen bewijzen dat

$$\{x = n\}S\{a[x] = 0 \wedge \forall i : n \leq i < x : a[i] \neq 0\}$$

Bewijs (top-down):
CONS ($x = n \rightarrow \forall i : n \leq i < x : a[i] \neq 0$):
$\{\forall n \leq i < x : a[i] \neq 0\}S\{a[x] = 0 \wedge \forall i : n \leq i < x : a[i] \neq 0\}$

LOOP:
$\{\forall n \leq i < x : a[i] \neq 0 \wedge a[x] \neq 0\}x := x + 1\{\forall i : n \leq i < x :$
$a[i] \neq 0\}$

CONS
($\forall n \leq i < x : a[i] \neq 0 \wedge a[x] \neq 0 \rightarrow \forall n \leq i < x + 1 : a[i] \neq 0$):
$\{\forall n \leq i < x + 1 : a[i] \neq 0\}x := x + 1\{\forall i : n \leq i < x : a[i] \neq 0\}$

Correctness Summation Program

$$
\begin{aligned}
SUM \equiv\ & k := 0;\ x := 0; \\
& \textbf{while } k \neq N \textbf{ do} \\
& \quad x := x + a[k]; \\
& \quad k := k + 1 \\
& \textbf{od}.
\end{aligned}
$$

To prove

$$\{N \geq 0\} SUM \{x = \Sigma_{i=0}^{N-1}\ a[i]\}$$

Proof Outline for the Summation Program

$$
\begin{aligned}
SUM \equiv\ & \{N \geq 0\} \\
& \{0 \leq 0 \leq N \wedge 0 = \Sigma_{i=0}^{-1}\ a[i]\} \\
& k := 0;\ x := 0; \\
& \{0 \leq k \leq N \wedge x = \Sigma_{i=0}^{k-1}\ a[i]\} \\
& \textbf{while } k \neq N \textbf{ do} \\
& \{0 \leq k \leq N \wedge k \neq N \wedge x = \Sigma_{i=0}^{k-1}\ a[i]\} \\
& \quad \{0 \leq k < N \wedge x = \Sigma_{i=0}^{k-1}\ a[i]\} \\
& \quad \{0 \leq (k+1) \leq N \wedge x + a[k] = \Sigma_{i=0}^{(k+1)-1}\ a[i]\} \\
& \quad x := x + a[k]; \\
& \quad \{0 \leq (k+1) \leq N \wedge x = \Sigma_{i=0}^{(k+1)-1}\ a[i]\} \\
& \quad k := k + 1 \\
& \quad \{0 \leq k \leq N \wedge x = \Sigma_{i=0}^{k-1}\ a[i]\} \\
& \textbf{od}. \\
& \{0 \leq k \leq N \wedge x = \Sigma_{i=0}^{k-1}\ a[i] \wedge \neg(k \neq N)\} \\
& \{x = \Sigma_{i=0}^{N-1}\ a[i]\}
\end{aligned}
$$

Case Study: Minimum-Sum Section Problem

Let $s_{i,j}$ denote the sum of section $a[i:j]$:

$$s_{i,j} = \Sigma_{k=i}^{j}\ a[k].$$

Design *MINSUM* such that

$$\{N > 0\}MINSUM\{sum = min\ \{s_{i,j} \mid 0 \leq i \leq j < N\}\}$$

For example, the minimum-sum section of

$$a[0:4] = (5, -3, 2, -4, 1)$$

is

$$a[1:3] = (-3, 2, -4)$$

and its sum is -5.

Invariant

Let

$$s_k = min\ \{s_{i,j} \mid 0 \leq i \leq j < k\}.$$

Note that

$$min\ \{s_{i,j} \mid 0 \leq i \leq j < N\} = s_N$$

We *construct* a loop with invariant

$$1 \leq k \leq N \wedge sum = s_k$$

While Body

$$
\begin{array}{ll}
 & s_{k+1} \\
= & \quad \{\text{definition of } s_{k+1}\} \\
 & min(\{s_{i,j} \mid 0 \leq i \leq j < k+1\}) \\
= & \quad \{\text{definition of } s_{i,j}\} \\
 & min(\{s_{i,j} \mid 0 \leq i \leq j < k\} \cup \{s_{i,k} \mid 0 \leq i < k+1\}) \\
= & \quad \{\text{associativity of } min\} \\
 & min(min(\{s_{i,j} \mid 0 \leq i \leq j < k\}), min(\{s_{i,k} \mid 0 \leq i < k+1\})) \\
= & \quad \{\text{definition of } t_{k+1}\} \\
 & min(s_k, t_{k+1})
\end{array}
$$

where

$$t_k \equiv min\ \{s_{i,k-1} \mid 0 \leq i < k\}$$

Synthesis

$\{N > 0\}$
$\{1 \leq 1 \leq N \wedge a[0] = s_1\}$
$k := 1;\ sum := a[0];$
$\{1 \leq k \leq N \wedge sum = s_k\}$
while $k \neq N$ **do** $\{1 \leq k \leq N \wedge sum = s_k \wedge k \neq N\}$
$\{1 \leq k+1 \leq N \wedge min(sum, t_{k+1}) = s_{k+1}\}$
$sum := min(sum, t_{k+1});$
$\{1 \leq k+1 \leq N \wedge sum = s_{k+1}\}$
$k := k+1$
$\{1 \leq k \leq N \wedge sum = s_k\}$
od
$\{1 \leq k \leq N \wedge sum = s_k \wedge \neg(k \neq N)\}$
$\{sum = s_N\}$

Initialization

$$N > 0 \rightarrow (1 \leq k \leq N \wedge sum = s_k)[k, sum := 1, a[0]]$$

Note that

$$(1 \leq k \leq N \wedge sum = s_k)[k, sum := 1, a[0]]$$
$$=$$
$$1 \leq 1 \leq N \wedge a[0] = s_1$$

Boolean Test

$$(1 \leq k \leq N \wedge sum = s_k \wedge k \neq N)$$
$$\rightarrow$$
$$(1 \leq k + 1 \leq N \wedge sum = s_k)$$

Finalization

$$1 \leq k \leq N \wedge sum = s_k \wedge k = N)$$
$$\rightarrow$$
$$sum = s_N$$

Computation of t_{k+1}

$$
\begin{array}{ll}
 & t_{k+1} \\
= & \quad \{\text{definition of } t_k\} \\
 & min\ \{s_{i,k} \mid 0 \leq i < k+1\} \\
= & \quad \{\text{associativity of } min\} \\
 & min(min\ \{s_{i,k} \mid 0 \leq i < k\},\ s_{k,k}) \\
= & \quad \{s_{i,k} = s_{i,k-1} + a[k]\} \\
 & min(min\ \{s_{i,k-1} + a[k] \mid 0 \leq i < k\},\ a[k]) \\
= & \quad \{\text{property of } min\} \\
 & min(min\ \{s_{i,k-1} \mid 0 \leq i < k\} \ +\ a[k],\ a[k]) \\
= & \quad \{\text{definition of } t_k\} \\
 & min(t_k + a[k], a[k])
\end{array}
$$

Correctness by Construction

$\{N > 0\}$
$\{1 \leq 1 \leq N \wedge a[0] = s_1 \wedge a[0] = t_1\}$
$k := 1;\ sum := a[0];\ x := a[0];$
$\{1 \leq k \leq N \wedge sum = s_k \wedge x = t_k\}$
while $k \neq N$
do $\{1 \leq k+1 \leq N \wedge sum = s_k \wedge x = t_k \wedge k \neq N\}$
$\{1 \leq k+1 \leq N \wedge min(sum, min(x + a[k], a[k])) = s_{k+1} \wedge$
$min(x + a[k], a[k]) = t_{k+1}\}$
$x := min(x + a[k], a[k]);$
$\{1 \leq k+1 \leq N \wedge min(sum, x) = s_{k+1} \wedge x = t_{k+1}\}$
$sum := min(sum, x);$
$\{1 \leq k+1 \leq N \wedge sum = s_{k+1} \wedge x = t_{k+1}\}$
$k := k+1$
$\{1 \leq k \leq N \wedge sum = s_k \wedge x = t_k\}$
od
$\{1 \leq k \leq N \wedge sum = s_k \wedge x = t_k \wedge k = N\}$
$\{sum = s_N\}$

Eerste deeltentamen: Voorbeelden

Bewijs de correctheidsbewering

$$\{\mathbf{true}\}\ a[i] := a[j]\ \{a[i] = a[j]\}$$

waar a een array is van type **integer** $\rightarrow$ **integer**.

Uitwerking

Assignment Axiom:

$$\{(a[i] = a[j])[a[i] := a[j]]\}\ a[i] := a[j]\ \{a[i] = a[j]\}$$

We berekenen de preconditie:

$$\begin{array}{ll}
(a[i] = a[j])[a[i] := a[j]] & \equiv \\
a[i][a[i] := a[j]] = a[j][a[i] := a[j]] & \equiv \\
\mathbf{if}\ i = i\ \mathbf{then}\ a[j]\ \mathbf{else}\ a[i]\ \mathbf{fi} = \mathbf{if}\ j = i\ \mathbf{then}\ a[j]\ \mathbf{else}\ a[j]\ \mathbf{fi} & \leftrightarrow \\
a[j] = a[j] & \leftrightarrow \\
\mathbf{true} &
\end{array}$$

Bewijs

$$\{n \geq 0\}$$
$$k := 0; \textbf{while } k \leq n \textbf{ do } a[k] := b[n-k]; k := k+1 \textbf{ od}$$
$$\{\forall i \in [0:n] : a[i] = b[n-i]\}$$

$\{n \geq 0\}$
$\{\forall i \in [0 : -1] : a[i] = b[n - i] \wedge 0 \leq n + 1\}$
$k := 0$
$\{\forall i \in [0 : k - 1] : a[i] = b[n - i] \wedge k \leq n + 1\}$
while $k \leq n$
do $\{\forall i \in [0 : k - 1] : a[i] = b[n - i] \wedge k \leq n \wedge k \leq n + 1\}$
$\{\forall i \in [0 : k - 1] : \textbf{if } i = k \textbf{ then } b[n - k] \textbf{ else } a[i] \textbf{ fi} = b[n - i] \wedge$
$\textbf{if } k = k \textbf{ then } b[n - k] \textbf{ else } a[k] \textbf{ fi} = b[n - k] \wedge k \leq n\}$
$a[k] := b[n - k];$
$\{\forall i \in [0 : k - 1] : a[i] = b[n - i] \wedge a[k] = b[n - k] \wedge k \leq n\}$
$\{\forall i \in [0 : (k + 1) - 1] : a[i] = b[n - i] \wedge k + 1 \leq n + 1\}$
$k := k + 1$
$\{\forall i \in [0 : k - 1] : a[i] = b[n - i] \wedge k \leq n + 1\}$
od
$\{\forall i \in [0 : k - 1] : a[i] = b[n - i] \wedge k \leq n + 1 \wedge \neg(k \leq n)\}$
$\{\forall i \in [0 : n] : a[i] = b[n - i]\}$

Bewijs

$$\{x \geq 0 \wedge y \geq 0\}$$
$$p := 0; c := 0; \textbf{while } c > 0 \textbf{ do } p := p + x; c := c + 1 \textbf{ od}$$
$$\{p = x \times y\}$$

$\{x \geq 0 \wedge y \geq 0\}$
$\{0 = x \times (y - y) \wedge y \geq 0\}$
$p := 0; c := y$
$\{p = x \times (y - c) \wedge c \geq 0\}$
while $c > 0$
do $\{p = x \times (y - c) \wedge c \geq 0 \wedge c > 0\}$
$\{p + x = x \times (y - c) + x \wedge c - 1 \geq 0\}$
$\{p + x = x \times (y - (c - 1)) \wedge c - 1 \geq 0\}$
$p := p + x;$
$\{p = x \times (y - (c - 1)) \wedge c - 1 \geq 0\}$
$c := c - 1$
$\{p = x \times (y - c) \wedge c \geq 0\}$
od
$\{p = x \times (y - c) \wedge c \geq 0 \wedge \neg(c > 0)\}$
$\{p = x \times y\}$

Total Correctness Interpretation

$$\{p\}S\{q\}$$

iff

Every computation of S in a state which satisfies precondition p terminates and does so in a state which satisfies the postcondition q

Examples

- $\{x \geq 0\}$ **while** $x \neq 0$ **do** $x := x - 1$ **od** $\{\textbf{true}\}$
  But not

$$\{\textbf{true}\}\ \textbf{while}\ x \neq 0\ \textbf{do}\ x := x - 1\ \textbf{od}\ \{\textbf{true}\}$$

- For which precondition p we have

$$\{p\}\ \textbf{while}\ a[x] \neq 0\ \textbf{do}\ x := x + 1\ \textbf{od}\ \{\textbf{true}\}$$

  Answer: $\exists n : a[n] = 0 \wedge x \leq n$

Exercise

For which statements S we have

- $\models_{tot} \{\mathbf{true}\} S \{\mathbf{false}\}$
- $\models_{tot} \{\mathbf{true}\} S \{\mathbf{true}\}$
- $\models_{tot} \{\mathbf{false}\} S \{\mathbf{true}\}$

Verification Total Correctness

RULE : LOOP II

$$\frac{\begin{array}{l}\{p \wedge B\}\ S\ \{p\},\\ \{p \wedge B \wedge t = z\}\ S\ \{t < z\},\\ p\ \rightarrow t \geq 0\end{array}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg\ B\}}$$

Example

Prove total correctness of

$$\{x \geq 0\}\ \textbf{while}\ x \neq 0\ \textbf{do}\ x := x - 1\ \textbf{od}\ \{\textbf{true}\}$$

- What's the bound t?
  Answer: take for t variable x.
- What's the invariant p such that p implies $x \geq 0$?
  Answer: take for p the assertion $x \geq 0$ itself.

Proof obligations

1. $\{x \geq 0 \wedge x \neq 0\}\ x := x - 1\ \{x \geq 0\}$
2. $\{x \geq 0 \wedge x \neq 0 \wedge x = z\}\ x := x - 1\ \{x < z\}$
3. $x \geq 0 \rightarrow x \geq 0$

Total Correctness Zero Search

To prove total correctness of

$\{\exists n : a[n] = 0 \wedge x \leq n\}$**while** $a[x] \neq 0$ **do** $x := x + 1$ **od**$\{a[x] = 0\}$

- What is the bound function t?
- What is the invariant p?

Solution: Instantiation

1. Proof total correctness of

   $\{a[n] = 0 \wedge x \leq n\}$**while** $a[x] \neq 0$ **do** $x := x + 1$ **od**$\{a[x] = 0\}$

2. ($\exists$-INTRODUCTION RULE:1)

   $\{\exists n : a[n] = 0 \wedge x \leq n\}$**while** $a[x] \neq 0$ **do** $x := x + 1$ **od**$\{a[x] = 0\}$

Zero Search: Bound and Invariant

Define

- bound t by $n - x$
- invariant p by $a[n] = 0 \wedge x \leq n$

Proof obligations:

- $\{a[n] = 0 \wedge x \leq n \wedge a[x] \neq 0\}\ x := x + 1\ \{a[n] = 0 \wedge x \leq n\}$
- $a[n] = 0 \wedge x \leq n \rightarrow n - x \geq 0$
- $\{a[n] = 0 \wedge x \leq n \wedge a[x] \neq 0 \wedge n - x = z\}\ x := x + 1\ \{n - x < z\}$

Some Auxiliary Rules

∃-INTRODUCTION RULE

$$\frac{\{p\}\ S\ \{q\}}{\{\exists x : p\}\ S\ \{q\}}$$

where x does not occur in S or in $\mathit{free}(q)$.

DISJUNCTION RULE

$$\frac{\{p\}\ S\ \{q\}, \{r\}\ S\ \{q\}}{\{p \vee r\}\ S\ \{q\}}$$

CONJUNCTION RULE

$$\frac{\{p_1\}\ S\ \{q_1\}, \{p_2\}\ S\ \{q_2\}}{\{p_1 \wedge p_2\}\ S\ \{q_1 \wedge q_2\}}$$

SUBSTITUTION RULE

$$\frac{\{p\}\ S\ \{q\}}{\{p[\bar{z} := \bar{t}]\}\ S\ \{q[\bar{z} := \bar{t}]\}}$$

where $(\{\bar{z}\} \cap \mathit{var}(S)) \cup (\mathit{var}(\bar{t}) \cap \mathit{change}(S)) = \emptyset$.

Total Correctness: Decomposition

RULE : DECOMPOSITION

$$\frac{\vdash_p \{p\}\ S\ \{q\},\quad \vdash_t \{p\}\ S\ \{\mathbf{true}\}}{\{p\}\ S\ \{q\}}$$

where the provability signs $\vdash_p$ and $\vdash_t$ refer to proof systems for partial and total correctness for the considered program S, respectively.

Example Decomposition Total Correctness

To prove total correctness of

$$\{x \geq 0 \wedge y > 0\}\ DIV\ \{q \cdot y + r = x \wedge 0 \leq r < y\}$$

where DIV denotes

$$q := 0; r := x; \textbf{while } r \geq y \textbf{ do } r := r - y;\ q := q + 1 \textbf{ od}$$

it suffices to prove its partial correctness and total correctness of

$$\{x \geq 0 \wedge y > 0\}\ DIV\ \{\textbf{true}\}$$

Total Correctness Proof Outlines: An Example

$\{x \geq 0 \wedge y > 0\}$
$q := 0;\ r := x;$
$\{r \geq 0 \wedge y > 0\}\ \{\mathbf{bd} : r\}$
$\{r \geq 0 \wedge y > 0 \wedge r \geq 0\}$
while $r \geq y$ **do**
$\{r \geq 0 \wedge y > 0 \wedge r \geq y\}$
$\{r \geq 0 \wedge y > 0 \wedge r \geq y \wedge r = r\}$
z:=r
$\{r \geq 0 \wedge y > 0 \wedge r \geq y \wedge r = z\}$
$\{r - y \geq 0 \wedge y > 0 \wedge r - y < z\}$
$r := r - y;\ q := q + 1$
$\{r \geq 0 \wedge y > 0 \wedge r < z\}$
od
$\{\mathbf{true}\}$.

Recursive Programs

Given a set of *procedure identifiers* with typical elements

$$P, P_1, \ldots,$$

we extend the syntax of **while** programs by the following clause:

$$S ::= P$$

Note: no parameters.

Procedure declarations

$$P :: S$$

Programs

$$D \mid S$$

where D is a set of procedure declarations and S is the *main* statement.

Factorial Program

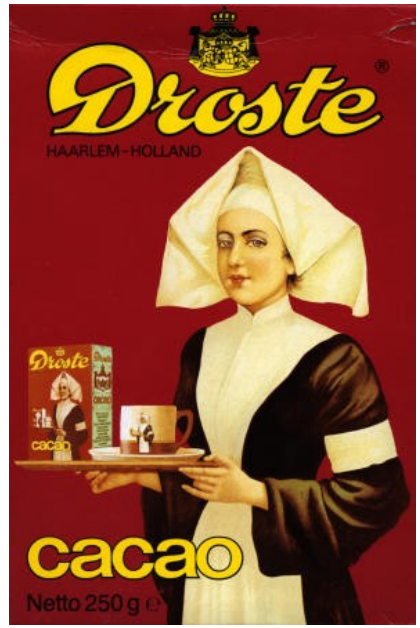
Fac ::
if $x = 0$
then $y := 1$
else $x := x - 1;\ \mathit{Fac};\ x := x + 1;\ y := y \cdot x$
fi
$|\ \mathit{Fac}$

Verification Recursive Programs

The rule

$$\frac{\{p\}\ S\ \{q\}}{\{p\}\ P\ \{q\}}$$

where $P :: S \in D$, gives rise to an infinite regression:



Example: try to prove

$$\{\mathbf{true}\}\ P\ \{\mathbf{true}\}$$

where $P :: P \in D$.

The Simplified Recursion Rule

Assume what you want to prove.

Let $D = P :: S$.

$$\frac{\{p\}\ P\ \{q\} \vdash \{p\}\ S\ \{q\}}{\{p\}\ P\ \{q\}}$$

Correctness Factorial Program

We prove

$$\{x \geq 0\}\ \mathit{Fac}\ \{y = x!\} \vdash \{x \geq 0\}\ S\ \{y = x!\}$$

where S denote the body of Fac.

$\{x \geq 0\}$
if $x = 0$
then $\{x \geq 0 \wedge x = 0\}$
$\{1 = x!\}$
$y := 1$
$\{y = x!\}$
else $\{x \geq 0 \wedge x \neq 0\}$
$\{x - 1 \geq 0\}$
$x := x - 1;$
$\{x \geq 0\}$
$Fac;$
$\{y = x!\}$
$\{y \cdot (x + 1) = (x + 1)!\}$
$x := x + 1;$
$\{y \cdot x = x!\}$
$y := y \cdot x$
$\{y = x!\}$
fi
$\{y = x!\}$

Proving Invariant Properties

We want to prove the correctness formula

$$\{z = x\}\ \mathit{Fac}\ \{z = x\}$$

Try it!

Does Not Work, Hey?!

We need the Substitution Rule:

$$\frac{\{p\}\ S\ \{q\}}{\{p[z := t]\}\ S\ \{q[z := t]\}}$$

where

- z does not appear in the program
- the variables of t are *read-only*

Another Proof-Outline for the Factorial Program

$\{z = x\}$
if $x = 0$
then $\{z = x\}$
$y := 1$
$\{z = x\}$
else $\{z = x\}$
$\{z - 1 = x - 1\}$
$x := x - 1;$
$\{z - 1 = x\}$ ($\equiv (z = x)[z := z - 1]$)
$Fac;$
$\{z - 1 = x\}$ ($\equiv (z = x)[z := z - 1]$)
$\{z = x + 1\}$
$x := x + 1;$
$\{z = x\}$
$y := y \cdot x$
$\{z = x\}$
fi
$\{z = x\}$

Summing Up Recursively

Let P be declared by

$$\textbf{if } n \neq k-1 \textbf{ then } sum := sum + a[k]; k := k+1; P \textbf{ fi}$$

Prove

$$\{k = 1\}\ sum := 0; P\ \{sum = \Sigma_{i=1}^{n} a[i]\}$$

To prove

$$\{k = 1 \wedge sum = 0\}\ P\ \{sum = \Sigma_{i=1}^{n} a[i]\}$$

Generalized assumption:

$$\{sum = \Sigma_{i=1}^{k-1} a[i]\}\ P\ \{sum = \Sigma_{i=1}^{n} a[i]\}$$

Note that

$$(k = 1 \wedge sum = 0) \rightarrow sum = \Sigma_{i=1}^{k-1} a[i]$$

and ...

$\{sum = \Sigma_{i=1}^{k-1} a[i]\}$
if $n \neq k-1$
then $\{sum = \Sigma_{i=1}^{k-1} a[i]\}$
$\{sum + a[k] = \Sigma_{i=1}^{k} a[i]\}$
$sum := sum + a[k];$
$\{sum = \Sigma_{i=1}^{k} a[i]\}$
$k := k + 1;$
$\{sum = \Sigma_{i=1}^{k-1} a[i]\}$
P
$\{sum = \Sigma_{i=1}^{n} a[i]\}$
else $\{n = k-1 \wedge sum = \Sigma_{i=1}^{k-1} a[i]\}$
$\{sum = \Sigma_{i=1}^{n} a[i]\}$
$skip$
$\{sum = \Sigma_{i=1}^{n} a[i]\}$
fi
$\{sum = \Sigma_{i=1}^{n} a[i]\}$

Case Study: Binary Search

$$
\begin{array}{lll}
\mathit{BinSearch} :: & m := (f' + l')\ \mathit{div}\ 2; & \\
& \textbf{if}\ f' \neq l' & \\
& \textbf{then if}\ a[m] < v & \\
& \qquad \textbf{then}\ f' := m + 1;\ \mathit{BinSearch} & \\
& \qquad \textbf{else if}\ a[m] > v & \\
& \qquad\qquad \textbf{then}\ \ l' := m;\ \mathit{BinSearch} & \\
& \qquad\qquad \textbf{fi} & \\
& \qquad \textbf{fi} & \\
& \textbf{fi} &
\end{array}
$$

Correctness Binary Search

$$\{f = f' \wedge l' = l \wedge f' \leq l' \wedge \mathit{sorted}(a)\}$$
$$\mathit{BinSearch}$$
$$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$$

where

- $\mathit{sorted}(a) \equiv \forall n : a[n] \leq a[n+1]$
- $v \in a[f : l] \equiv \exists n \in [f : l] : v = a[n]$

Does Not Fit!

$\{f = f' \wedge l' = l \wedge f' \leq l' \wedge \mathit{sorted}(a)\}$
$m := (f' + l')\ \mathit{div}\ 2;$
if $f' \neq l'$
then if $a[m] < v$
then $f' := m + 1;$
$\{f = f' \wedge l' = l \wedge f' \leq l' \wedge \mathit{sorted}(a)\}$
$\mathit{BinSearch}$
$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$
else if $a[m] > v$
then $l' := m;$
$\{f = f' \wedge l' = l \wedge f' \leq l' \wedge \mathit{sorted}(a)\}$
$\mathit{BinSearch}$
$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$
fi
fi
fi
$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$

So? Let's Weaken The Precondition.

$\{f \leq f' \leq l' \leq l \wedge \mathit{sorted}(a)\}$
$m := (f' + l')\ \mathit{div}\ 2;$
if $f' \neq l'$
then if $a[m] < v$
then $f' := m + 1;$
$\{f \leq f' \leq l' \leq l \wedge \mathit{sorted}(a)\}$
$\mathit{BinSearch}$
$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$
else if $a[m] > v$
then $l' := m;$
$\{f \leq f' \leq l' \leq l \wedge \mathit{sorted}(a)\}$
$\mathit{BinSearch}$
$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$
fi
fi
fi
$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$

Still Does Not Fit, Hey?

What's missing?
Well, this:

$$v \in a[f : l] \rightarrow v \in a[f' : l']$$

Suffices to prove

$$\{f \leq f' \leq l' \leq l \wedge (v \in a[f : l] \rightarrow v \in a[f' : l']) \wedge \mathit{sorted}(a)\}$$
$$\mathit{BinSearch}$$
$$\{f \leq m \leq l \wedge (v \in a[f : l] \rightarrow a[m] = v)\}$$

$\{p\}$
$m := (f' + l')\ div\ 2;$
$\{p \wedge m = (f' + l')\ div\ 2\}$
if $f' \neq l'$ **then**
  if $a[m] < v$ **then**
    $\{p \wedge m = (f' + l')\ div\ 2 \wedge f' \neq l' \wedge a[m] < v\}$
    $\{p[f' := m + 1]\}$
    $f' := m + 1;\ \{p\}\ BinSearch\ \{q\}$
  else if $a[m] > v$ **then**
      $\{p \wedge m = (f' + l')\ div\ 2 \wedge f' \neq l' \wedge a[m] > v\}$
      $\{p[l' := m]\}$
      $l' := m; \{p\}\ BinSearch\ \{q\}$
      else $\{p \wedge m = (f' + l')\ div\ 2 \wedge a[m] = v\}\ \{q\}$
      fi $\{q\}$
  fi $\{q\}$
else $\{p \wedge m = (f' + l')\ div\ 2 \wedge f' = l'\}\ \{q\}$
fi
$\{q\}$

Recursive Procedures with Parameters

Procedure declarations

$$P(u_1, \ldots, u_n) :: S.$$

where $u_1, \ldots, u_n$ are the *formal parameters* of procedure P.

Procedure calls

$$S ::= P(t_1, \ldots, t_n).$$

where $t_1, \ldots, t_n$ are expressions called *actual parameters*.

The Factorial Program (Again)

$Fac(u)$ ::
if $u = 0$
then $y := 1$
else $Fac(u-1);\ y := y \cdot u$
fi
$|\ \ Fac(x)$

Call-By-Value

Inlining (macro expansion of) a procedure call $P(t_1, \ldots, t_n)$ by

$$\textbf{block local } u_1, \ldots, u_n := t_1, \ldots, t_n; S \textbf{ end}$$

given the declaration $P(u_1, \ldots, u_n) :: S$.

Simultaneous assignment

$$u_1, \ldots, u_n := t_1, \ldots, t_n$$

assign the values of $t_1, \ldots, t_n$ to $u_1, \ldots, u_n$.

Partial Correctness Blocks

Examples:

- $\{x = z\}$ **block local** $x := t; S$ **end** $\{x = z\}$
- $\{x = z\}$ **block local** $x := x + 1; y := x$ **end** $\{y = z + 1\}$

Example: Expanding Fac(x)

$$
\begin{array}{l}
\textbf{block local } u := x; \\
\quad \textbf{if } u = 0 \textbf{ then } y := 1 \\
\quad \textbf{else } \textbf{ block local } u := u - 1 \\
\qquad\qquad \textbf{if } u = 0 \textbf{ then } y := 1 \\
\qquad\qquad \textbf{else } \textbf{ block local } u := u - 1 \\
\qquad\qquad\qquad\qquad \textbf{if } u = 0 \textbf{ then } y := 1 \\
\qquad\qquad\qquad\qquad \textbf{else } \textbf{ block local } u := u - 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
\qquad\qquad\qquad\qquad\qquad\quad \textbf{end}; y := y \times u \\
\qquad\qquad\qquad\qquad \textbf{fi} \\
\qquad\qquad\qquad\quad \vdots \\
\qquad\qquad\qquad\quad \textbf{end}; y := y \times u \\
\qquad\qquad \textbf{fi} \\
\qquad\qquad \vdots \\
\quad \textbf{fi} \\
\textbf{end}; y := y \times u
\end{array}
$$

Static Scoping

Let

$$P(x) :: Q \text{ and } Q :: y := x$$

Then static scoping ensures

$$\{x = 0\}\ P(1)\ \{x = 0 \wedge y = 0\}$$

But $P(1)$ expands to

$$\textbf{block local } x := 1; y := x \textbf{ end}$$

Thus

$$\{x = 0\}\ P(1)\ \{x = 0 \wedge y = 1\}$$

Syntactic restriction:

no name clashes between formal parameters and global variables.

Recursion by Macro Expansion and Block Rule

The macro expansion rule

$$\frac{\{p\}\ \textbf{block local}\ \bar{u} := \bar{t}; S\ \textbf{end}\ \{q\}}{\{p\}\ P(\bar{t})\ \{q\}}$$

where $P(\bar{u}) :: S \in D$.

The block rule

$$\frac{\{p\}\ \bar{x} := \bar{t};\ S\ \{q\}}{\{p\}\ \textbf{block local}\ \bar{x} := \bar{t}; S\ \textbf{end}\ \{q\}}$$

where q does not contain free occurrences of the local variables $\bar{x}$.

General Recursion by Macro Expansion and Block Rule

$$\{p_1\}\ P_1(\bar{t}_1)\ \{q_1\}, \ldots, \{p_n\}\ P_n(\bar{t}_n)\ \{q_n\} \vdash \{p\}\ S\ \{q\},$$
$$\{p_1\}\ P_1(\bar{t}_1)\ \{q_1\}, \ldots, \{p_n\}\ P_n(\bar{t}_n)\ \{q_n\} \vdash$$
$$\{p_i\}\ \textbf{block local}\ \bar{u}_i := \bar{t}_i;\ S_i\ \{q_i\},\ i \in \{1, \ldots, n\}$$

where $P_i(\bar{u}_1) :: S_i \in D$ for $i \in \{1, \ldots, n\}$.

Example

Let D contain the following declaration

$$add(x) :: sum := sum + x.$$

Exercise: using the above block rule, prove

$$\{sum = z\}\ \textbf{block local}\ x := 1; sum := sum + x\ \textbf{end}\ \{sum = z + 1\}$$

By applying the above copy rule we then derive

$$\{sum = z\}\ add(1)\ \{sum = z + 1\}$$

The Recursion Rule

$$\frac{\begin{array}{l}\{p_1\}\ P_1(\bar{u}_1)\ \{q_1\}, \ldots, \{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash \{p\}\ S\ \{q\}, \\ \{p_1\}\ P_1(\bar{u}_1)\ \{q_1\}, \ldots, \{p_n\}\ P_n(\bar{u}_n)\ \{q_n\} \vdash \\ \qquad \{p_i\}\ S_i\ \{q_i\},\ i \in \{1, \ldots, n\}\end{array}}{\{p\}\ S\ \{q\}}$$

where $P_i(\bar{u}_1) :: S_i \in D$ and q_i does not contain the free occurrences of the formal parameters $\bar{u}_i$, for $i \in \{1, \ldots, n\}$.

Reasoning About Generic Calls

Instantiation

Generic calls are instantiated by the following rule.

$$\frac{\{p\}\ P(\bar{u})\ \{q\}}{\{p[\bar{u} := \bar{t}]\}\ P(\bar{t})\ \{q\}}$$

where $P(\bar{u}) :: S \in D$, and no formal parameter of $\bar{u}$ appears (free) in q.

We prove

$$\{z = u \wedge u \geq 0\}\ \mathit{Fac}(u)\ \{y = z!\} \vdash \{z = u \wedge u \geq 0\}\ S\ \{y = z!\}$$

where S is the body of Fac.

$\{z = u \wedge u \geq 0\}$
if $u = 0$
then $\{z = u \wedge u = 0\}$
$\{1 = z!\}$
$y := 1$
$\{y = z!\}$
else $\{z = u \wedge u \geq 0 \wedge u \neq 0\}$
$\{z = u \wedge u - 1 \geq 0\}$
$Fac(u - 1);$
$\{z = u \wedge y = z - 1!\}$
$\{y \cdot u = z!\}$
$y := y \cdot u$
$\{y = z!\}$
fi
$\{y = z!\}$

Remains to show that

$$\{z = u \wedge u \geq 0\}\ \mathit{Fac}(u)\ \{y = z!\}$$
$$\vdash$$
$$\{z = u \wedge u - 1 \geq 0\}\ \mathit{Fac}(u-1)\ \{z = u \wedge y = (z-1)!\}$$

From the assumption $\{z = u \wedge u \geq 0\}\ \mathit{Fac}(u)\ \{y = z!\}$ we derive by an application of the instantiation rule that

$$\{z = u - 1 \wedge u - 1 \geq 0\}\ \mathit{Fac}(u-1)\ \{y = z!\}$$

An application of the substitution rule, replacing the freeze variable z by $z - 1$, and the consequence rule, gives

$$\{z = u \wedge u - 1 \geq 0\}\ \mathit{Fac}(u-1)\ \{y = (z-1)!\}$$

The invariance axiom gives us

$$\{z = u\}\ \mathit{Fac}(u-1)\ \{z = u\}$$

We conclude by the conjunction that

$$\{z = u \wedge u - 1 \geq 0\}\ \mathit{Fac}(u-1)\ \{z = u \wedge y = (z-1)!\}$$

Recursively Generating Fibonacci Numbers

Mathematical definition

$$F(0) = 0$$
$$F(1) = 1$$
$$F(n) = F(n-1) + F(n-2)$$

Implementation Let $P(x)$ be declared by

if $x = 0$
then $y := 0$
else if $x = 1$
    then $y := 1$
    else $P(x-1)$;
        block local $u := y$;
        $P(x-2)$;
        $y := u + y$
        end
    fi
fi

Correctness $\{z = x \wedge x \geq 0\}\ P(x)\ \{y = F(z)\}$

$\{z = x \wedge x \geq 0\}$
if $x = 0$
then $\{z = x = 0\} y := 0 \{z = x = y = 0\} \{y = F(z)\}$
else **if** $x = 1$
then $\{z = x = 1\} y := 1 \{z = x = y = 1\} \{y = F(x)\}$
else $\{z = x > 1\} P(x-1); \{y = F(z-1) \wedge z = x > 1\}$
block local $u := y;$
$\{u = F(z-1) \wedge z = x > 1\}$
$P(x-2);$
$\{u = F(z-1) \wedge y = F(z-2)\}$
$y := u + y$
$\{y = F(z-1) + F(z-2)\} \{y = F(z)\}$
end
$\{y = F(z)\}$
fi
$\{y = F(z)\}$
fi
$\{y = F(x)\}$

Adapting The Assumptions

To prove

$$\{z = x \geq 0\}\ P(x)\ \{y = F(z)\}$$
$$\vdash$$
$$\{z = x > 1\}\ P(x-1)\ \{y = F(z-1) \wedge z = x > 1\}$$

1. $\{z = x \geq 0\}\ P(x)\ \{y = F(z)\}$ (assumption)
2. $\{z = x - 1 \geq 0\}\ P(x-1)\ \{y = F(z)\}$ (instantiation)
3. $\{z - 1 = x - 1 \geq 0\}\ P(x-1)\ \{y = F(z-1)\}$ (substitution)
4. $\{z = x > 1\}\ P(x-1)\ \{y = F(z-1)\}$ (consequence)
5. $\{z = x > 1\}\ P(x-1)\ \{z = x > 1)\}$ (invariance)
6. $\{z = x > 1\}\ P(x-1)\ \{y = F(z-1) \wedge z = x > 1\}$ (conjunction plus consequence)

To prove

$$\{z = x \geq 0\}\ P(x)\ \{y = F(z)\}$$
$$\vdash$$
$$\{u = F(z-1) \wedge z = x > 1\}\ P(x-2)\ \{u = F(z-1) \wedge y = F(z-2)\}$$

1. $\{z = x \geq 0\}\ P(x)\ \{y = F(z)\}$ (assumption)
2. $\{z = x - 2 \geq 0\}\ P(x-2)\ \{y = F(z)\}$ (instantiation)
3. $\{z - 2 = x - 2 \geq 0\}\ P(x-2)\ \{y = F(z-2)\}$ (substitution)
4. $\{z = x > 1\}\ P(x-2)\ \{y = F(z-2)\}$ (consequence)
5. $\{u = F(z-1)\}\ P(x-2)\ \{u = F(z-1))\}$ (invariance)

   $\{u = F(z-1) \wedge z = x > 1\}$
6. $P(x-2)$ (conjunction)

   $\{u = F(z-1) \wedge y = F(z-2)\}$

Mutual Recursive Procedures: An Example

Given the procedure declarations

$$O :: \textbf{if } n = 0 \textbf{ then } b := \textbf{false else } n := n - 1; E \textbf{ fi}$$

and

$$E :: \textbf{if } n = 0 \textbf{ then } b := \textbf{true else } n := n - 1; O \textbf{ fi}$$

prove

$$\{n \geq 0 \wedge \mathit{Even}(n)\}\ E\ \{b\}.$$

Solution

We introduce the assumptions:

- $\{n \geq 0 \wedge \mathit{Even}(n)\}\ E\ \{b\}$
- $\{n \geq 0 \wedge \mathit{Odd}(n)\}\ O\ \{b\}$

and prove

- $$\{n \geq 0 \wedge \mathit{Even}(n)\}$$
  $$\textbf{if } n = 0 \textbf{ then } b := \textbf{true else } n := n - 1; O \textbf{ fi}$$
  $$\{b\}$$
- $$\{n \geq 0 \wedge \mathit{Odd}(n)\}$$
  $$\textbf{if } n = 0 \textbf{ then } b := \textbf{false else } n := n - 1; E \textbf{ fi}$$
  $$\{b\}$$

And Here We Go

$\{n \geq 0 \wedge Even(n)\}$
if $n = 0$
then $\{\textbf{true}\}$
$b := \textbf{true}$
$\{b\}$
else $\{n - 1 \geq 0 \wedge Odd(n - 1)\}$
$n := n - 1;$
$\{n \geq 0 \wedge Odd(n)\}$
O
$\{b\}$
fi
$\{b\}$

And Now Against All Odds ...

$\{n \geq 0 \wedge \mathit{Odd}(n)\}$
if $n = 0$
then $\{$**false**$\}$
$\quad b := \textbf{false}$
$\quad \{b\}$
else $\{n - 1 \geq 0 \wedge \mathit{Even}(n-1)\}$
$\quad n := n - 1;$
$\quad \{n \geq 0 \wedge \mathit{Even}(n)\}$
$\quad E$
$\quad \{b\}$
fi
$\{b\}$

Program Correctness in Practice

- Microsoft Spec#
- The Eiffel programming language
- The Java Modeling Language (JML)
- The Object Constraint Language (OCL)
- Run-time Assertion Checking
- The KeY Project: Integrated Deductive Software Design
- Separation Logic AnalYzER
- Temporal Logic of Actions, Leslie Lamport (Turing Award 2013)

Major Challenges

- Tool support (e.g., theorem proving, proof reuse, counter-example generation, ...)
- Object-orientation (heaps, sub-typing, ...)
- Multithreading (interference, locks, ...)

Example: Object-Orientation

Field assignment

$$\{p[x.f := s]\}\ x.f := s\ \{p\}$$

where

$$y.f[x.f := s] \equiv \textbf{if } y = x \textbf{ then } s \textbf{ else } y.f \textbf{ fi}$$

Method calls

$$o.m(\bar{s}) = m(o, \bar{s})$$

Object creation

$$\{p[x := new]\}\ x := new\ \{p\}$$

Example: Shared Variable Concurrency

How to prove

$$\{x = 0\}\ x := x + 1 \parallel x := x + 1\ \{x = 2\}$$

Interference free proof-outlines:

$$\begin{array}{ll} \{x = y + z \wedge y = 0\} & \{x = y + z \wedge z = 0\} \\ x, y := x + 1, 1 & x, z := x + 1, 1 \\ \{x = y + z \wedge y = 1\} & \{x = y + z \wedge z = 1\} \end{array}$$

Interference freedom test:

$$\begin{array}{c} \{x = y + z \wedge y = 0 \wedge z = 0\} \\ x, z := x + 1, 1\rangle \\ \{x = y + z \wedge y = 0\} \end{array}$$

Conjunction rule:

$$\frac{\{x = y + z \wedge y = 0 \wedge z = 0\}}{x, z := x + 1, 1 \parallel x, y := x + 1, 1}{\{x = y + z \wedge y = 1 \wedge z = 1\}}$$

Consequence rule:

$$\frac{\{x = 0 \wedge y = 0 \wedge z = 0\}}{x, z := x + 1, 1 \parallel x, y := x + 1, 1}{\{x = 2\}}$$

Elimination auxiliary variables:

$$\frac{\{x = 0 \wedge y = 0 \wedge z = 0\}\ x := x + 1 \parallel x := x + 1\ \{x = 2\}}{}$$

Substitution and Consequence rule:

$$\frac{\{x = 0\}\ x := x + 1 \parallel x := x + 1\ \{x = 2\}}{}$$