Variable Declaration

var x: T· P declare local state variable x with type T and scope P

$=$ $\exists x, x'\colon T\cdot P$

var x: int· $x{:=}\ 2.\ \ y{:=}\ x{+}z$

$=$ $\exists x, x'\colon int\cdot\ x'{=}2\ \land\ y' = 2{+}z\ \land\ z'{=}z$ one-point for x' and idempotent for x

$=$ $y' = 2{+}z\ \land\ z'{=}z$

var x: int· $y{:=}\ x$

$=$ $\exists x, x'\colon int\cdot\ x'{=}x \land y'{=}x \land z'{=}z$ one-point for x and x'

$=$ $z'{=}z$

var x: int· $y{:=}\ x{-}x$

$=$ $y'{=}0 \land z'{=}z$

Variable Declaration

$$\textbf{var } x{:}\ T\cdot\ P$$

$$=\quad \exists x{:}\ \mathit{undefined}\cdot\ \exists x'{:}\ T,\mathit{undefined}\cdot\ P$$

$$\textbf{var } x{:}\ T := e\cdot\ \ P$$

$$=\quad \exists x{:}\ e\cdot\ \exists x'{:}\ T\cdot\ P$$

Variable Suspension

Suppose the state consists of variables w , x , y , and z .

frame $w, x \cdot P$ within P , y and z are constants (no y' and z')

$=$ $P \wedge y'=y \wedge z'=z$

$x := e \;\; = \;\;$ **frame** $x \cdot x' = e$

$ok \;\; = \;\;$ **frame** $\cdot \top$

$s := \Sigma L \;\; \Leftarrow \;\;$ **frame** $s \cdot$ **var** $n: nat \cdot \;\; s' = \Sigma L$

$s' = \Sigma L \;\; \Leftarrow$

Array

$A\ i := e \quad = \quad A'i{=}e \wedge (\forall j\cdot\ j{\neq}i \Rightarrow A'j{=}Aj) \wedge x'{=}x \wedge y'{=}y \wedge \ldots$

	$A2{:=}\ 3.\ \ i{:=}\ 2.\ \ Ai{:=}\ 4.\ \ Ai{=}A2$	X Substitution Law
$=$	$A2{:=}\ 3.\ \ i{:=}\ 2.\ \ 4{=}A2$	√ Substitution Law
$=$	$A2{:=}\ 3.\ \ 4{=}A2$	X Substitution Law
$=$	$4{=}3$	
$=$	$\bot$ X	

	$A2{:=}\ 2.\ \ A(A2){:=}\ 3.\ \ A2{=}2$	X Substitution Law
$=$	$A2{:=}\ 2.\ \ A2{=}2$	X Substitution Law
$=$	$2{=}2$	
$=$	$\top$ X	

Array

$A\ i := e \quad = \quad A'i=e \land (\forall j\cdot\ j\ne i \Rightarrow A'j=Aj) \land x'=x \land y'=y \land \ldots$

$= \quad A' = i\to e \mid A \ \land\ x'=x \ \land\ y'=y \ \land \ldots$

$= \quad A{:=}\ i\to e \mid A$

$A2{:=}\ 3.\ \ i{:=}\ 2.\ \ Ai{:=}\ 4.\ \ Ai=A2$

$= \quad A{:=}\ 2\to 3 \mid A.\ \ i{:=}\ 2.\ \ A{:=}\ i\to 4 \mid A.\ \ Ai = A2$ — Substitution Law

$= \quad A{:=}\ 2\to 3 \mid A.\ \ i{:=}\ 2.\ \ (i\to 4 \mid A)i = (i\to 4 \mid A)2$ — Substitution Law

$= \quad A{:=}\ 2\to 3 \mid A.\ \ (2\to 4 \mid A)2 = (2\to 4 \mid A)2$ — $=$ is reflexive

$= \quad A{:=}\ 2\to 3 \mid A.\ \ \top$ — Substitution Law

$= \quad \top$

Array

$A\ i := e \quad = \quad A'i=e \land (\forall j\cdot\ j \neq i \Rightarrow A'j=Aj) \land x'=x \land y'=y \land \ldots$

$= \quad A' = i \to e \mid A \ \land\ x'=x \ \land\ y'=y \ \land\ \ldots$

$= \quad A := i \to e \mid A$

$A2 := 2.\ \ A(A2) := 3.\ \ A2=2$

$= \quad A := 2\to 2 \mid A.\ \ A := A2\to 3 \mid A.\ \ A2=2$ Substitution Law

$= \quad A := 2\to 2 \mid A.\ \ (A2\to 3 \mid A)2 = 2$ Substitution Law

$= \quad ((2\to 2 \mid A)2\to 3 \mid 2\to 2 \mid A)\ 2 = 2$

$= \quad (2\to 3 \mid 2\to 2 \mid A)\ 2 = 2$

$= \quad 3 = 2$

$= \quad \bot$

Array

remember

$A\,i := e$ becomes $A := i \rightarrow e \mid A$

$A\,i\,j := e$ becomes $A := (i;j) \rightarrow e \mid A$

Record

$$person \;=\; \text{"name"} \rightarrow text$$
$$\quad\quad\quad \mid \; \text{"age"} \rightarrow nat$$

var p: $person$

p:= "name" → "Josh" | "age" → 17

p "age":= 18

p:= "age" → 18 | p

While Loop

$$W \Leftarrow \textbf{while } b \textbf{ do } P$$

means

$$W \Leftarrow \textbf{if } b \textbf{ then } (P.\ W) \textbf{ else } ok$$

to prove

$$s' = s + \Sigma\, L\,[n;..\#L] \;\wedge\; t' = t + \#L - n \;\Leftarrow$$

$$\textbf{while } n{\neq}\#L \textbf{ do } (s{:=}\ s + Ln.\ \ n{:=}\ n{+}1.\ \ t{:=}\ t{+}1)$$

prove instead

$$s' = s + \Sigma\, L\,[n;..\#L] \;\wedge\; t' = t + \#L - n \;\Leftarrow$$

$$\textbf{if } n{\neq}\#L \textbf{ then } (\ s{:=}\ s + Ln.\ \ n{:=}\ n{+}1.\ \ t{:=}\ t{+}1.$$

$$s' = s + \Sigma\, L\,[n;..\#L] \;\wedge\; t' = t + \#L - n)$$

$$\textbf{else } ok$$

Exit Loop

$L \Leftarrow$ **loop**

$A.$

exit when $b.$

C

end

means

$$L \Leftarrow A.\ \textbf{if } b \textbf{ then } ok \textbf{ else } (C.\ L)$$

Deep Exit

$$P \Leftarrow$$ **loop**

A.

loop

B.

exit 2 **when** c.

D

end.

E

end

means

$$P \Leftarrow A.\ Q$$

$$Q \Leftarrow B.\ \mathbf{if}\ c\ \mathbf{then}\ ok\ \mathbf{else}\ (D.\ Q)$$

Deep Exit

$P \Leftarrow$ **loop**
  A.
  exit 1 **when** b.
  C.
  loop
    D.
    exit 2 **when** e.
    F.
    exit 1 **when** g.
    H
  end.
  I
end

means

$P \Leftarrow A.$ **if** b **then** ok **else** $(C. Q)$

$Q \Leftarrow D.$ **if** e **then** ok **else** $(F.$ **if** g **then** $(I. P)$ **else** $(H. Q))$

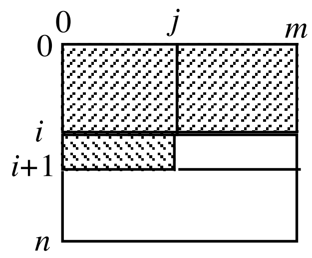
Two-Dimensional Search

$P \;=\;$ **if** x: $A\ (0,..n)\ (0,..m)$ **then** $x = A\ i'\ j'$ **else** $i'{=}n \wedge j'{=}m$

$Q \;=\;$ **if** x: $A\ (i,..n)\ (0,..m)$ **then** $x = A\ i'\ j'$ **else** $i'{=}n \wedge j'{=}m$

$R \;=\;$ **if** x: $A\ i\ (j,..m),\ A\ (i{+}1,..n)\ (0,..m)$ **then** $x = A\ i'\ j'$ **else** $i'{=}n \wedge j'{=}m$

$P \;\Leftarrow\; i{:=}\,0.\ \ i{\le}n \Rightarrow Q$



$i{\le}n \Rightarrow Q \;\Leftarrow\;$ **if** $i{=}n$ **then** $j{:=}\ m$ **else** $i{<}n \Rightarrow Q$

$i{<}n \Rightarrow Q \;\Leftarrow\; j{:=}\,0.\ \ i{<}n \wedge j{\le}m \Rightarrow R$

$i{<}n \wedge j{\le}m \Rightarrow R \;\Leftarrow\;$ **if** $j{=}m$ **then** $(i{:=}\ i{+}1.\ \ i{\le}n \Rightarrow Q)$ **else** $i{<}n \wedge j{<}m \Rightarrow R$

$i{<}n \wedge j{<}m \Rightarrow R \;\Leftarrow\;$ **if** $A\ i\ j = x$ **then** *ok* **else** $(j{:=}\ j{+}1.\ \ i{<}n \wedge j{\le}m \Rightarrow R))$

$$t' \le t + n \times m \quad \Leftarrow \quad i := 0.\;\; i \le n \;\Rightarrow\; t' \le t + (n-i) \times m$$

$$i \le n \;\Rightarrow\; t' \le t + (n-i) \times m \quad \Leftarrow \quad \textbf{if } i=n \textbf{ then } j := m \textbf{ else } i<n \;\Rightarrow\; t' \le t + (n-i) \times m$$

$$i<n \;\Rightarrow\; t' \le t + (n-i) \times m \quad \Leftarrow \quad j := 0.\;\; i<n \land j \le m \;\Rightarrow\; t' \le t + (n-i) \times m - j$$

$$i<n \land j \le m \;\Rightarrow\; t' \le t + (n-i) \times m - j \quad \Leftarrow$$

$$t := t+1.$$

$$\textbf{if } j=m \textbf{ then } (i := i+1.\;\; i \le n \;\Rightarrow\; t' \le t + (n-i) \times m)$$

$$\textbf{else } i<n \land j<m \;\Rightarrow\; t' \le t + (n-i) \times m - j$$

$$i<n \land j<m \;\Rightarrow\; t' \le t + (n-i) \times m - j \quad \Leftarrow$$

$$\textbf{if } A\;i\;j = x \textbf{ then } ok \textbf{ else } (j := j+1.\;\; i<n \land j \le m \;\Rightarrow\; t' \le t + (n-i) \times m - j)$$

$P \;\Leftarrow\; i{:=}\,0.\;\; L0$

$L0 \;\Leftarrow\;$ **if** $i{=}n$ **then** $j{:=}\,m$

else $(j{:=}\,0.\;\; L1)$

$L1 \;\Leftarrow\;$ **if** $j{=}m$ **then** $(i{:=}\,i{+}1.\;\; L0)$

else if $A\;i\;j = x$ **then** ok

else $(j{:=}\,j{+}1.\;\; L1)$

in C:

```
P:   i = 0;
L0:  if (i==n) j = m;
     else {      j = 0;
            L1: if (j==m) {i = i+1;  goto L0;}
                else if (A[i][j]==x);
                else {j = j+1;  goto L1;}
          }
```

For Loop

for i:= m;..n **do** P

i is a fresh name (a local constant)

m and n are integer expressions such that $m \le n$

the number of iterations is $n-m$

P is a specification

For Loop

$$Fmn \quad \Leftarrow \quad \textbf{for } i{:=}\ m;..n \textbf{ do } P$$

means

$$Fii \quad \Leftarrow \quad m{\le}i{\le}n \land ok$$

$$Fi(i{+}1) \quad \Leftarrow \quad m{\le}i{<}n \land P$$

$$Fik \quad \Leftarrow \quad m{\le}i{<}j{<}k{\le}n \land (Fij.\ Fjk)$$

For Loop

example: $x'=2^n$

$$F \;=\; \langle i, j: nat \rightarrow x' = x \times 2^{j-i} \rangle$$

$$x'=2^n \;\Leftarrow\; x:= 1.\; F0n$$

proof

	$x:= 1.\; F0n$	expand $F0n$
$=$	$x:= 1.\; x' = x \times 2^{n-0}$	simplify and Substitution Law
$=$	$x'=2^n$	

For Loop

example: $x'=2^n$

$$F \;=\; \langle i, j\colon nat \to x' = x\times 2^{j-i}\rangle$$

$$x'=2^n \;\Leftarrow\; x:=1.\; F0n$$

$$F0n \;\Leftarrow\; \textbf{for } i:=0;..n \textbf{ do } x:=2\times x$$

proof

	$F\ i\ i$	
$=$	$x' = x\times 2^{i-i}$	law of exponents
$=$	$x' = x\times 1$	simplify
$=$	$x' = x$	
$\Leftarrow$	ok	

For Loop

example: $x'=2^n$

$$F \;=\; \langle i, j: nat \rightarrow x' = x\times 2^{j-i}\rangle$$

$$x'=2^n \;\Leftarrow\; x:= 1.\;\; F0n$$

$$F0n \;\Leftarrow\; \textbf{for } i:= 0;..n \textbf{ do } x:= 2\times x$$

proof

$Fi(i+1)$ expand

$=$ $x' = x\times 2^{i+1-i}$

$\Leftarrow$ $x:= 2\times x$

For Loop

example: $x'=2^n$

$F \;=\; \langle i, j: nat \rightarrow x' = x \times 2^{j-i} \rangle$

$x'=2^n \;\Leftarrow\; x:= 1.\; F0n$

$F0n \;\Leftarrow\;$ **for** $i:= 0;..n$ **do** $x:= 2 \times x$

proof

$F\ i\ j.\ F\ j\ k$

$=\quad x' = x \times 2^{j-i}.\ x' = x \times 2^{k-j}$ dependent composition

$=\quad x' = x \times 2^{j-i} \times 2^{k-j}$ law of exponents

$=\quad x' = x \times 2^{(j-i)+(k-j)}$ simplify

$=\quad x' = x \times 2^{k-i}$

For Loop

example:

$$t' = t + \Sigma i{:}\ m,..n\cdot fi \quad \Leftarrow \quad \textbf{for } i{:=}\ m;..n \textbf{ do } t' = t{+}fi$$

If $fi = c$ (a constant) then

$$t' = t + (n{-}m)\times c \quad \Leftarrow \quad \textbf{for } i{:=}\ m;..n \textbf{ do } t' = t{+}c$$

For Loop

example: add 1 to each item in a list

$$\#L'=\#L \ \land\ \forall i\colon 0,..\#L\cdot\ L'i=Li+1$$

Fik describes an arbitrary segment of iterations:

$$Fik \ = \ \ \#L'=\#L$$
$$\land\ (\forall j\colon i,..k\cdot\ L'j=Lj+1)$$
$$\land\ (\forall j\colon (0,..i), (k,..\#L)\cdot\ L'j=Lj)$$

$$F\,0\,(\#L) \ \Leftarrow\ \textbf{for}\ i{:=}\ 0;..\#L\ \textbf{do}\ L{:=}\ i\to Li+1 \mid L$$

prove

$$Fii \ \Leftarrow\ 0\le i\le\#L \land ok$$

$$Fi(i+1) \ \Leftarrow\ 0\le i<\#L \land (L{:=}\ i\to Li+1 \mid L)$$

$$Fik \ \Leftarrow\ 0\le i<j<k\le\#L \land (Fij.\ Fjk)$$

For Loop

special case: invariant

$$Im \Rightarrow I'n \quad \Leftarrow \quad \textbf{for } i := m;..n \textbf{ do } m \le i < n \wedge Ii \Rightarrow I'(i+1)$$

means

$$Ii \Rightarrow I'i \quad \Leftarrow \quad m \le i \le n \wedge ok$$

$$Ii \Rightarrow I'(i+1) \quad \Leftarrow \quad m \le i < n \wedge (m \le i < n \wedge Ii \Rightarrow I'(i+1))$$

$$Ii \Rightarrow I'k \quad \Leftarrow \quad m \le i < j < k \le n \wedge (Ii \Rightarrow I'j.\ \ Ij \Rightarrow I'k)$$

For Loop

special case: invariant

$Im \Rightarrow I'n \quad \Leftarrow \quad \textbf{for } i := m;..n \textbf{ do } m \le i < n \land Ii \Rightarrow I'(i+1)$

example: $x' = 2^n$

$Ii \quad = \quad x = 2^i$

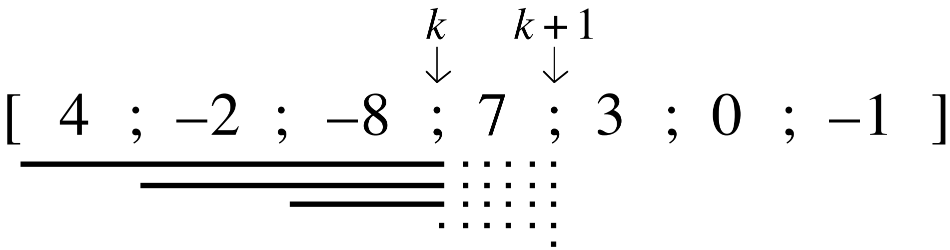
$x' = 2^n \quad \Leftarrow \quad x := 1.\ \ I0 \Rightarrow I'n$

$I0 \Rightarrow I'n \quad \Leftarrow \quad \textbf{for } i := 0;..n \textbf{ do } Ii \Rightarrow I'(i+1)$

$Ii \Rightarrow I'(i+1) \quad \Leftarrow \quad x := 2 \times x$

Minimum Sum Segment

Given a list L of integers, possibly including negatives, write a program to find the minimum sum of any segment.



$s' \;=\; MIN\; i, j\cdot \Sigma\, L\, [i;..j] \;\Leftarrow\; s{:=}\, 0.\;\; c{:=}\, 0.\;\; I\, 0 \Rightarrow I'(\#L)$

$I0 \Rightarrow I'(\#L) \;\Leftarrow\;$ **for** $k{:=}\, 0;..\#L$ **do** $Ik \Rightarrow I'(k{+}1)$

$Ik \Rightarrow I'(k{+}1) \;\Leftarrow\; c{:=}\, min\, (c + L\, k)\, 0.\;\; s{:=}\, min\; c\; s$

$I\,k \;=\; s \;=\; (MIN\; i{:}\; 0,..k{+}1\cdot MIN\; j{:}\; i,..k{+}1\cdot \Sigma\, L\, [i;..j])$

$\wedge \quad c \;=\; (MIN\; i{:}\; 0,..k{+}1\cdot \Sigma\, L\, [i;..k])$

Review

Boolean Theory	laws	proof	
Number Theory	Character Theory		
Bunches	Sets	Strings	Lists
Functions	Quantifiers		
Specification	Refinement	exact precondition	exact postcondition
Program Development	Time Calculation	real time	recursive time
Space Calculation	maximum space	average space	
Scope	variable declaration	frame	
Data Structures	array element assignment		
Control Structures	while loop	loop with exit	for loop

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

constant n: *nat* variables x, y: *nat*

$x'=n^3 \Leftarrow x:= n.\ x'=x\times n.\ x'=x\times n$

$x'=x\times n \Leftarrow y:= x.\ x:= 0.\ x' = x + y\times n$

$x' = x + y\times n \Leftarrow$ **if** $y=0$ **then** *ok* **else** $(x:= x+n.\ y:= y-1.\ x' = x + y\times n)$

In C

```
void P (void) { x = n;  Q( );  Q( );}
void Q (void) { y = x;  x = 0;  R( );}
void R (void) { if (y==0); else {x += n;  y--;  R( );}}
```

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

constant n: *nat* variables x, y: *nat*

$x'=n^3 \Leftarrow x:= n.\ x'=x\times n.\ x'=x\times n$

$x'=x\times n \Leftarrow y:= x.\ x:= 0.\ x' = x + y\times n$

$x' = x + y\times n \Leftarrow$ **if** $y=0$ **then** *ok* **else** $(x:= x+n.\ y:= y-1.\ x' = x + y\times n)$

In C

```
void P (void) { x = n;  Q( );  Q( );}
void Q (void) { y = x;  x = 0;  R: if (y==0); else {x += n;  y--;  goto R;}}
```

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

constant n: *nat* variables x, y: *nat*

$x'=n^3 \Leftarrow x{:=}\ n.\ x'=x{\times}n.\ x'=x{\times}n$

$x'=x{\times}n \Leftarrow y{:=}\ x.\ x{:=}\ 0.\ x' = x + y{\times}n$

$x' = x + y{\times}n \Leftarrow$ **if** $y{=}0$ **then** *ok* **else** $(x{:=}\ x{+}n.\ y{:=}\ y{-}1.\ x' = x + y{\times}n)$

In C

```
void P (void) { x = n;  Q( );  Q( );}
void Q (void) { y = x;  x = 0;  while (y!=0) {x += n;  y--;}}
```

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

constant n: *nat* variables x, y: *nat*

$x'=n^3 \;\Leftarrow\; x:= n.\;\; x'=x\times n.\;\; x'=x\times n$

$x'=x\times n \;\Leftarrow\; y:= x.\;\; x:= 0.\;\; x' = x + y\times n$

$x' = x + y\times n \;\Leftarrow\;$ **if** $y=0$ **then** *ok* **else** $(x:= x+n.\;\; y:= y-1.\;\; x' = x + y\times n)$

In C

```
x = n;
y = x;  x = 0;  while (y!=0) {x += n;  y--;}
y = x;  x = 0;  while (y!=0) {x += n;  y--;}
```

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

constant n: *nat* variables x, y: *nat* time t: *xnat*

$x'=n^3 \quad \Leftarrow \quad x:= n.\ x'=x\times n \quad .\ x'=x\times n$

$x'=x\times n \quad \Leftarrow \quad y:= x.\ x:= 0.\ x' = x + y\times n \wedge t'=t+y$

$x' = x + y\times n \wedge t'=t+y \quad \Leftarrow$

$\quad$ **if** $y=0$ **then** *ok* **else** $(x:= x+n.\ y:= y-1.\ t:= t+1.\ x' = x + y\times n \wedge t'=t+y)$

proof

	$y=0 \wedge ok$	expand *ok*
$=$	$y=0 \wedge x'=x \wedge y'=y \wedge t'=t$	context
$=$	$y=0 \wedge x' = x + y\times n \wedge y'=y \wedge t'=t+y$	specialize
$\Rightarrow$	$x' = x + y\times n \wedge t'=t+y$	

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

constant n: *nat*   variables x, y: *nat*   time t: *xnat*

$x'=n^3 \Leftarrow x:= n.\ x'=x\times n\ .\ x'=x\times n$

$x'=x\times n \Leftarrow y:= x.\ x:= 0.\ x' = x + y\times n \land t'=t+y$

$x' = x + y\times n \land t'=t+y \Leftarrow$

if y=0 **then** *ok* **else** $(x:= x+n.\ y:= y-1.\ t:= t+1.\ x' = x + y\times n \land t'=t+y)$

proof

$y\neq0 \land (x:= x+n.\ y:= y-1.\ t:= t+1.\ x' = x + y\times n \land t'=t+y)$ substitution law

$= \quad y\neq0 \land x' = x + n + (y-1)\times n \land t'=t+1+y-1$ simplify and specialize

$\Rightarrow \quad x' = x + y\times n \land t'=t+y$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

constant n: *nat* variables x, y: *nat* time t: *xnat*

$x'=n^3 \land t'=t+n^2+n \quad \Leftarrow \quad x:= n.\ x'=x\times n \land t'=t+x\ .\ x'=x\times n \land t'=t+x$

$x'=x\times n \land t'=t+x \quad \Leftarrow \quad y:= x.\ x:= 0.\ x' = x + y\times n \land t'=t+y$

$x' = x + y\times n \land t'=t+y \quad \Leftarrow$

if $y=0$ **then** *ok* **else** $(x:= x+n.\ y:= y-1.\ t:= t+1.\ x' = x + y\times n \land t'=t+y)$

proof

	$x:= n.\ x'=x\times n \land t'=t+x.\ x'=x\times n \land t'=t+x$	substitution law
$=$	$x'=n^2 \land t'=t+n.\ x'=x\times n \land t'=t+x$	dependent composition
$=$	$\exists x'', y'', t''\cdot\ x''=n^2 \land t''=t+n \land x'=x''\times n \land t'=t''+x''$	1-pt for x'', t'', idempotent for y''
$=$	$x'=n^3 \land t'=t+n^2+n$	

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$n^3 \;=\; (n\text{–}1)^3 + 3{\times}n^2 - 3{\times}n + 1$

$n^2 \;=\; (n\text{–}1)^2 + 2{\times}n - 1$

variables x, y, n: *nat*

$x'{=}n^3 \;\Leftarrow\; x'{=}n^3 \land y'{=}n^2$

$x'{=}n^3 \land y'{=}n^2 \;\Leftarrow$

if $n{=}0$ **then** $(x{:=}\,0.\;\; y{:=}\,0)$

else ($n{:=}\,n\text{–}1.\;\; x'{=}n^3 \land y'{=}n^2.$

$y{:=}\,y + n + n - 1.$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$n^3 \;=\; (n-1)^3 + 3\times n^2 - 3\times n + 1$

$n^2 \;=\; (n-1)^2 + 2\times n - 1$

variables x, y, n: nat

$x'=n^3 \;\Leftarrow\; x'=n^3 \land y'=n^2 \land n'=n$

$x'=n^3 \land y'=n^2 \land n'=n \;\Leftarrow$

if $n=0$ **then** $(x:=0.\;\; y:=0)$

else ($n:=n-1.\;\; x'=n^3 \land y'=n^2 \land n'=n.$

$y:=y+n+n-1.$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$n^3 \;=\; (n-1)^3 + 3\times n^2 - 3\times n + 1$

$n^2 \;=\; (n-1)^2 + 2\times n - 1$

variables x, y, n: *nat*

$x'=n^3 \land t'=t+n \;\Leftarrow\; x'=n^3 \land y'=n^2 \land n'=n \land t'=t+n$

$x'=n^3 \land y'=n^2 \land n'=n \land t'=t+n \;\Leftarrow$

if $n=0$ **then** $(x:=0.\;\; y:=0)$

else $(\, n:=n-1.\;\; t:=t+1.\;\; x'=n^3 \land y'=n^2 \land n'=n \land t'=t+n.\;\; n:=n+1.$

$y:=y+n+n-1.\;\; x:=x+y+y+y-n-n-n+1)$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$$x'=n^3 \quad \Leftarrow \quad x:=0.\ \ I0 \Rightarrow I'n$$

$$I0 \Rightarrow I'n \quad \Leftarrow \quad \textbf{for } k:=0;..n \textbf{ do } Ik \Rightarrow I'(k+1)$$

$$Ik \Rightarrow I'(k+1) \quad \Leftarrow \quad x:=x+?$$

$$Ik \quad = \quad x=k^3$$

$$Ik \Rightarrow I'(k+1)$$

$= \quad x=k^3 \ \Rightarrow\ x'=(k+1)^3$

$= \quad x=k^3 \ \Rightarrow\ x' \ = \ k^3 + 3\times k^2 + 3\times k + 1$

$\Leftarrow \quad x:=x + 3\times k^2 + 3\times k + 1$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$$x'=n^3 \quad \Leftarrow \quad x:=0.\ \ y:=1.\ \ I0 \Rightarrow I'n$$

$$I0 \Rightarrow I'n \quad \Leftarrow \quad \mathbf{for}\ k:=0;..n\ \mathbf{do}\ Ik \Rightarrow I'(k+1)$$

$$Ik \Rightarrow I'(k+1) \quad \Leftarrow \quad x:=x+y$$

$$Ik \quad = \quad x=k^3 \ \wedge\ y = 3\times k^2 + 3\times k + 1$$

$$Ik \Rightarrow I'(k+1)$$

$$= \quad x=k^3 \ \wedge\ y = 3\times k^2 + 3\times k + 1 \quad \Rightarrow \quad x'=(k+1)^3 \ \wedge\ y' = 3\times(k+1)^2 + 3\times(k+1) + 1$$

$$= \quad x=k^3 \ \wedge\ y = 3\times k^2 + 3\times k + 1 \quad \Rightarrow \quad x' = x+y \ \wedge\ y' = 3\times k^2 + 9\times k + 7$$

$$= \quad x=k^3 \ \wedge\ y = 3\times k^2 + 3\times k + 1 \quad \Rightarrow \quad x' = x+y \ \wedge\ y' = y + 6\times k + 6$$

$$\Leftarrow \quad x:=x+y.\ \ y:=y+k+k+k+k+k+k+6$$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$$x'=n^3 \quad \Leftarrow \quad x:=0.\ \ y:=1.\ \ z:=6.\ \ I0 \Rightarrow I'n$$

$$I0 \Rightarrow I'n \quad \Leftarrow \quad \textbf{for } k:=0;..n \textbf{ do } Ik \Rightarrow I'(k+1)$$

$$Ik \Rightarrow I'(k+1) \quad \Leftarrow \quad x:=x+y.\ \ y:=y+z$$

$$Ik \quad = \quad x=k^3 \ \wedge\ y = 3\times k^2 + 3\times k + 1 \ \wedge\ z = 6\times k + 6$$

$$Ik \Rightarrow I'(k+1)$$

$$= \qquad x=k^3 \ \wedge\ y = 3\times k^2 + 3\times k + 1 \ \wedge\ z = 6\times k + 6$$

$$\Rightarrow \quad x'=(k+1)^3 \ \wedge\ y' = 3\times(k+1)^2 + 3\times(k+1) + 1 \ \wedge\ z' = 6\times(k+1) + 6$$

$$\Leftarrow \qquad x' = x+y \ \wedge\ y' = y+z \ \wedge\ z' = z+6$$

$$= \qquad x:=x+y.\ \ y:=y+z.\ \ z:=z+6$$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$x'=n^3 \quad \Leftarrow \quad x:= 0.\ \ y:= 1.\ \ z:= 6.\ \ I0 \Rightarrow I'n$

$I0 \Rightarrow I'n \quad \Leftarrow \quad \mathbf{for}\ k:= 0;..n\ \mathbf{do}\ Ik \Rightarrow I'(k+1)$

$Ik \Rightarrow I'(k+1) \quad \Leftarrow \quad x:= x+y.\ \ y:= y+z.\ \ z:= z+6$

$Ik \quad = \quad x=k^3 \ \land\ y = 3\times k^2 + 3\times k + 1 \ \land\ z = 6\times k + 6$

$x:= 0.\ \ y:= 1.\ \ z:= 6.\ \ \mathbf{for}\ k:= 0;..n\ \mathbf{do}\ (x:= x+y.\ \ y:= y+z.\ \ z:= z+6)$

Cube

Write a program that cubes using only addition, subtraction, and test for zero.

$x'=n^3 \wedge t'=t+n \quad \Leftarrow \quad x:=0.\ \ y:=1.\ \ z:=6.\ \ Q \wedge t'=t+n$

$Q \wedge t'=t+n \quad \Leftarrow$

$\qquad$ **if** $n=0$ **then** ok **else** $(x:=x+y.\ \ y:=y+z.\ \ z:=z+6.\ \ n:=n-1.\ \ t:=t+1.\ \ Q \wedge t'=t+n)$

$$Q \quad = \quad \forall k: nat\cdot\ x=k^3 \ \wedge\ y = 3\times k^2 + 3\times k + 1 \ \wedge\ z = 6\times k + 6 \quad \Rightarrow \quad x' = (k+n)^3$$

$$Q \quad = \quad y = 3\times x^{2/3} + 3\times x^{1/3} + 1 \ \wedge\ z = 6\times x^{1/3} + 6 \quad \Rightarrow \quad x' = (x^{1/3}+n)^3$$

```
    x = 0;  y = 1;  z = 6;
Q:  if (n!=0) {x+=y;  y+=z.  z+=6;  n--; goto Q;}
```

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ...	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := max\ t\ w$	busy-wait loop

wait until w $\Leftarrow$ **if** $t \ge w$ **then** ok **else** $(t := t+1.$ **wait until** $w)$

proof

$t \ge w \wedge ok$

$= \quad t \ge w \wedge (t := t)$

$= \quad t \ge w \wedge (t := max\ t\ w)$

$\Rightarrow$ **wait until** w

Time Depedence

deadline:= $t + 5$	no problem
if $t <$ *deadline* **then** ... **else** ...	no problem
t:= 5	problem: unimplementable
wait until w $=$ t:= *max* t w	busy-wait loop

wait until w $\Leftarrow$ **if** $t \geq w$ **then** *ok* **else** (t:= $t+1$. **wait until** w)

proof

$t<w \wedge$ (t:= $t+1$. **wait until** w)

$=$ $t<w \wedge$ (t:= $t+1$. t:= *max* t w)

$=$ $t+1 \leq w \wedge$ (t:= *max* $(t+1)$ w)

$=$ $t<w \wedge$ (t:= w)

$=$ $t<w \wedge$ (t:= *max* t w)

$\Rightarrow$ **wait until** w

Space Dependence

if $s < 1000000$ **then** ... **else** ... no problem

$s := 5$ problem

assignments to s must account for space

real space implementation dependent

Assertions

assert b

= “I believe b is true”

= **precondition** b

= **postcondition** b

= **invariant** b

= **if** b **then** ok **else** ($print$ "error: ... ". **wait until** ∞)

redundant, adds robustness, costs execution time

ensure b

= “make b be true”

= **if** b **then** ok **else** $b' \wedge ok$

= $b' \wedge ok$

unimplementable by itself, but may be used in some contexts

nondeterministic choice (a programming notation):

$P \textbf{ or } Q \quad = \quad P \lor Q$

$x:= 0 \textbf{ or } x:= 1.\ \textbf{ensure } x=1$

$= \quad x'=0 \land y'=y \ \lor \ x'=1 \land y'=y.\ x'=1 \land x'=x \land y'=y$

$= \quad \exists x'', y''\cdot\ (x''=0 \land y''=y \ \lor \ x''=1 \land y''=y) \land x'=1 \land x'=x'' \land y'=y''$

$= \quad (x'=0 \land y'=y \ \lor \ x'=1 \land y'=y) \land x'=1$

$= \quad x'=1 \land y'=y$

$= \quad x:= 1$

implementation: **backtracking**

natural square root Given natural n find natural s satisfying $s^2 \le n < (s+1)^2$

$s:= 0,..n+1.\ \textbf{ensure } s^2 \le n < (s+1)^2$

Result Expression

P **result** e — execute P then evaluate e but no state change

axiom $x' = (P \textbf{ result } e) \;=\; P.\; x'=e$

var $term, sum: rat := 1\cdot$

for $i:= 1;..15$ **do** $(term:= term/i.\; sum:= sum+term)$

result sum

$$x:= (\, y:= y+1 \textbf{ result } y)$$

$$= \quad x' = (y:= y+1 \textbf{ result } y) \;\wedge\; y'=y$$

$$= \quad (y:= y+1.\; x'=y) \wedge y'=y$$

$$= \quad x' = y+1 \;\wedge\; y'=y$$

$$= \quad x:= y+1$$

Result Expression

P **result** e execute P then evaluate e but no state change

implementation

Replace each nonlocal variable within P and e that is assigned within P

by a fresh local variable initialized to the value of the nonlocal variable.

Then execute P and evaluate e .

but some language implementations don't introduce local variables

so expression evaluation can cause state change

Side Effects

$x = x$ **?** not if there are side-effects **!**

$x + x = 2{\times}x$ **?** not if there are side-effects **!**

for reasoning

$x{:=}\ (P$ **result** $e)$	becomes	$P.\ \ x{:=}\ e$
$x{:=}\ y + (P$ **result** $e)$	becomes	(**var** $z{:=}\ y\cdot\ \ P.\ \ x{:=}\ z{+}e$)

Don't neglect the time for expression evaluation.

Function

```
int bexp (int n)
{  int r = 1;
   int i;
   for (i=0; i<n; i++) r = r*2;
   return r; }
```

$bexp \;=\; \langle\; n{:}\ int \rightarrow$
$\qquad$ **var** $r{:}\ int := 1\cdot$
$\qquad$ **for** $i{:}= 0;..n$ **do** $r{:}= r\times 2.$
$\qquad$ **assert** $r{:}\ int$
$\qquad$ **result** $r\;\rangle$

C function = assertion about the result
+ name
+ parameters
+ scope control
+ result expression

Procedure

procedure = name of procedure

+ parameters

+ scope control

$$P \;=\; \langle x\colon int \rightarrow a' < x < b' \rangle$$

$$P\,3 \;=\; a' < 3 < b'$$

$$P\,(a{+}1) \;=\; a' < a{+}1 < b'$$

$$a' < x < b' \;\Leftarrow\; a{:=}\, x{-}1.\;\; b{:=}\, x{+}1$$

$$\langle p\colon D \rightarrow B \rangle\, a \;=\; (\mathbf{var}\; p\colon D := a\cdot\; B)$$ if B doesn't use p' or $p{:=}$

Procedure

reference parameter **var** parameter

$\langle *x: int \rightarrow a:= 3.\ b:= 4.\ x:= 5\rangle\ a$

$= \quad a:= 3.\ b:= 4.\ a:= 5$

$= \quad a'=5 \wedge b'=4$

$\langle *x: int \rightarrow x:= 5.\ b:= 4.\ a:= 3\rangle\ a$

$= \quad a:= 5.\ b:= 4.\ a:= 3$

$= \quad a'=3 \wedge b'=4$

$\langle *x: int \rightarrow a'=3 \wedge b'=4 \wedge x'=5\rangle\ a \quad = \quad ?$

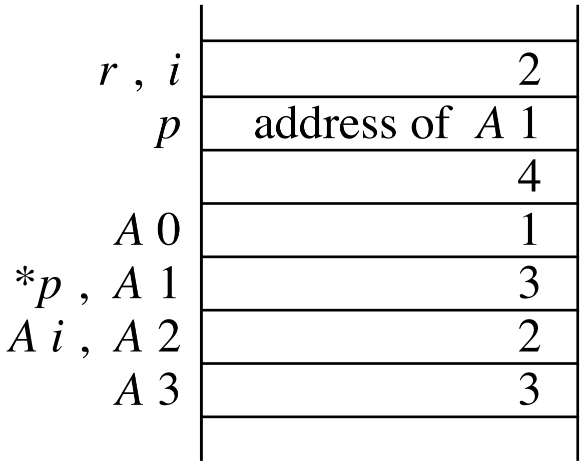
warning Use only for programs, not for arbitrary specifications.

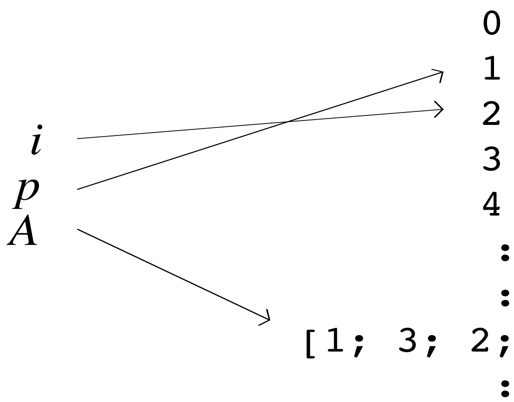
Do not manipulate the procedure body.

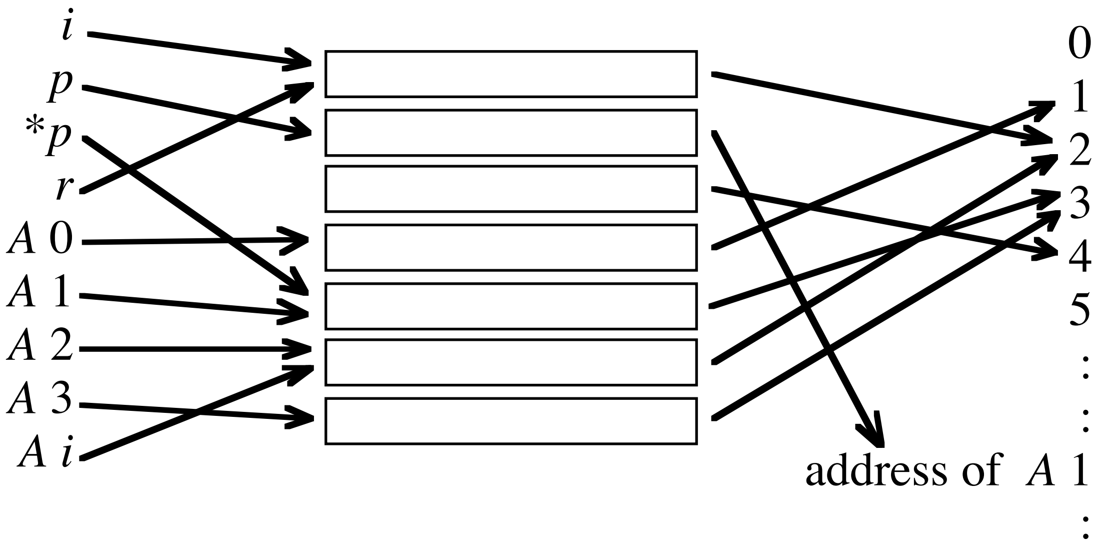
Substitute arguments for parameters before any other manipulations.

Apply programming theory separately for each call.

Alias


r , i
2
p
address of A 1
4
A 0
1
*p , A 1
3
A i , A 2
2
A 3
3


0
1
2
3
4
i
p
A
[1; 3; 2;


i
p
*p
r
A 0
A 1
A 2
A 3
A i
0
1
2
3
4
5
address of A 1

Probabilistic Programming

probability real number between 0 and 1

$prob = \S r: real\cdot 0\le r\le 1$ $\top = 1$ $\bot = 0$

$\neg x = 1-x$ $x\wedge y = x\times y$ $x\vee y = x - x\times y + y$

distribution value is a probability, sum is 1

says the probability of each state

2^{-n} is a distribution of $n: nat+1$ because $(\forall n: nat+1\cdot 2^{-n}: prob) \wedge (\Sigma n: nat+1\cdot 2^{-n})=1$

2^{-n} says $n=3$ with probability $1/8$

2^{-n-m} is a distribution of $n, m: nat+1$ because

$(\forall n, m: nat+1\cdot 2^{-n-m}: prob) \wedge (\Sigma n, m: nat+1\cdot 2^{-n-m})=1$

2^{-n-m} says $n=3 \wedge m=1$ with probability $1/16$

Probabilistic Programming

$n' = n+1$ says: if $n=5$ then $n'=6$ with probability 1

and $n'=7$ with probability 0

$(\forall n, n'{:}\ nat\cdot\ n' = n+1 : prob) \ \wedge\ (\Sigma n, n' \cdot\ n' = n+1) = \infty$

so $n' = n+1$ is not a distribution of n and n'

$(\forall n'{:}\ nat\cdot\ n' = n+1 : prob) \ \wedge\ (\Sigma n' \cdot\ n' = n+1) = 1$

so (for any value of n) $n' = n+1$ is a one-point distribution of n'

Any implementable deterministic specification is a one-point distribution of the final state.

Probabilistic Programming

$ok \;=\; (x'=x) \times (y'=y) \times \ldots$

$x{:=}\,e \;=\; (x'=e) \times (y'=y) \times \ldots$

$\textbf{if } b \textbf{ then } P \textbf{ else } Q \;=\; b \times P + (1-b) \times Q$

$P.\,Q \;=\; \Sigma x'', y'', \ldots\cdot$ (for $x', y', \ldots$ substitute $x'', y'', \ldots$ in P)

$\times$ (for $x, y, \ldots$ substitute $x'', y'', \ldots$ in Q)

example

$$\textbf{if } 1/3 \textbf{ then } x{:=}\ 0 \textbf{ else } x{:=}\ 1$$

$= \quad 1/3 \times (x'{=}0) + (1 - 1/3) \times (x'{=}1)$

evaluate using 0 for x'

$= \quad 1/3 \times (0{=}0) + (1 - 1/3) \times (0{=}1)$

$= \quad 1/3 \times 1 + 2/3 \times 0$

$= \quad 1/3$

evaluate using 1 for x'

$= \quad 1/3 \times (1{=}0) + (1 - 1/3) \times (1{=}1)$

$= \quad 1/3 \times 0 + 2/3 \times 1$

$= \quad 2/3$

evaluate using 2 for x'

$= \quad 1/3 \times (2{=}0) + (1 - 1/3) \times (2{=}1)$

$= \quad 1/3 \times 0 + 2/3 \times 0$

$= \quad 0$

example in one integer variable x

if $1/3$ **then** $x:= 0$ **else** $x:= 1$.

if $x=0$ **then if** $1/2$ **then** $x:= x+2$ **else** $x:= x+3$

else if $1/4$ **then** $x:= x+4$ **else** $x:= x+5$

$$= \quad \Sigma x''\cdot \quad ((x''=0)/3 + (x''=1)\times 2/3)$$
$$\times \quad (\quad (x''=0)\times((x'=x''+2)/2 + (x'=x''+3)/2)$$
$$+ \quad (x''\neq 0)\times((x'=x''+4)/4 + (x'=x''+5)\times 3/4))$$

$$= \quad (x'=2)/6 + (x'=3)/6 + (x'=5)/6 + (x'=6)/2$$

Average

after P, average value of e is $P.\ e$

as n varies over $nat+1$ according to distribution 2^{-n} the average value of n^2 is

$$2^{-n'}.\ n^2$$

$$= \quad \Sigma n'':\ nat+1\cdot\ 2^{-n''} \times n''^2$$

$$= \quad 6$$

Average

after P , average value of e is $P.\ e$

if $1/3$ **then** $x{:=}\ 0$ **else** $x{:=}\ 1$.

if $x{=}0$ **then if** $1/2$ **then** $x{:=}\ x{+}2$ **else** $x{:=}\ x{+}3$

else if $1/4$ **then** $x{:=}\ x{+}4$ **else** $x{:=}\ x{+}5$.

x

$= \quad (x'{=}2)/6 + (x'{=}3)/6 + (x'{=}5)/6 + (x'{=}6)/2.\ x$

$= \quad \Sigma x''\cdot\ ((x''{=}2)/6 + (x''{=}3)/6 + (x''{=}5)/6 + (x''{=}6)/2) \times x''$

$= \quad 1/6 \times 2\ +\ 1/6 \times 3\ +\ 1/6 \times 5\ +\ 1/2 \times 6$

$= \quad 4 + 2/3$

Average

after P , average value of e is $P.\ e$

after P , probability that b is true is $P.\ b$

Probability is just the average value of a boolean expression.

if 1/3 **then** $x:= 0$ **else** $x:= 1$.

if $x=0$ **then if** 1/2 **then** $x:= x+2$ **else** $x:= x+3$

else if 1/4 **then** $x:= x+4$ **else** $x:= x+5$.

$x>3$

$=\quad (x'=2)/6 + (x'=3)/6 + (x'=5)/6 + (x'=6)/2.\ \ x>3$

$=\quad \Sigma x''\cdot\ \ ((x''=2)/6 + (x''=3)/6 + (x''=5)/6 + (x''=6)/2)\ \times\ (x''>3)$

$=\quad 1/6 \times (2>3)\ +\ 1/6 \times (3>3)\ +\ 1/6 \times (5>3)\ +\ 1/2 \times (6>3)$

$=\quad 2/3$

Random Number Generator

$rand\ n$ has value r with probability $(r: 0,..n) / n$

$x{=}x$ therefore $rand\ n = rand\ n$?

$x{+}x = 2{\times}x$ therefore $rand\ n + rand\ n = 2 \times rand\ n$?

Replace $rand\ n$ with $r: int$ with distribution $(r: 0,..n) / n$

Replace $rand\ n$ with $r: 0,..n$ with distribution $1/n$

$x{:=} rand\ 2.\ \ x{:=} x + rand\ 3$ — replace one $rand$ with r and one with s

$= \Sigma r: 0,..2\cdot \Sigma s: 0,..3\cdot (x{:=} r)/2.\ \ (x{:=} x + s)/3$ — Substitution Law

$= \Sigma r: 0,..2\cdot \Sigma s: 0,..3\cdot (x' = r{+}s) / 6$ — sum

$= ((x' = 0{+}0) + (x' = 0{+}1) + (x' = 0{+}2) + (x' = 1{+}0) + (x' = 1{+}1) + (x' = 1{+}2)) / 6$

$= (x'{=}0) / 6 + (x'{=}1) / 3 + (x'{=}2) / 3 + (x'{=}3) / 6$

Random Number Generator

$rand\ n$ has value r with probability $(r: 0,..n) / n$

$x=x$ therefore $rand\ n = rand\ n$?

$x+x = 2\times x$ therefore $rand\ n + rand\ n = 2 \times rand\ n$?

Replace $rand\ n$ with $r: int$ with distribution $(r: 0,..n) / n$

Replace $rand\ n$ with $r: 0,..n$ with distribution $1/n$

$x:= rand\ 2.\ \ x:= x + rand\ 3$ replace $rand$

$=$ $(x': 0,..2)/2.\ \ (x': x+(0,..3))/3$ dependent composition

$=$ $\Sigma x''\cdot\ (x'': 0,..2)/2 \times (x': x''+(0,..3))/3$ sum

$=$ $1/2 \times (x': 0,..3)/3\ +\ 1/2 \times (x': 1,..4)/3$

$=$ $(x'=0) / 6\ +\ (x'=1) / 3 + (x'=2) / 3\ +\ (x'=3) / 6$

Blackjack

You are dealt a card from a deck; its value is in the range 1 to 13 inclusive. You may stop with just one card, or have a second card if you want. Your object is to get a total as near as possible to 14, but not over 14. Your strategy is to take a second card if the first is under 7.

	$x := (rand\ 13) + 1.\ \textbf{if}\ x<7\ \textbf{then}\ x := x + (rand\ 13) + 1\ \textbf{else}\ ok$	replace *rand* and *ok*
=	$(x'\colon (0,..13)+1)/13.\ \textbf{if}\ x<7\ \textbf{then}\ (x'\colon x+(0,..13)+1)/13\ \textbf{else}\ x'=x$	replace . and **if**
=	$\Sigma x''\cdot\ (x''\colon 1,..14)/13 \times ((x''<7)\times(x'\colon x''+1,..x''+14)/13 + (x''\geq 7)\times(x'=x''))$	
		by several omitted steps
=	$((2\leq x'<7)\times(x'-1) + (7\leq x'<14)\times 19 + (14\leq x'<20)\times(20-x'))\ /\ 169$	

Player x plays "under n" and player y plays "under n+1"

c:= ($rand$ 13) + 1. d:= ($rand$ 13) + 1.

if $c < n$ **then** x:=c+d **else** x:=c. **if** $c < n$+1 **then** y:=c+d **else** y:=c.

$y<x\le 14 \lor x\le 14<y$

replace $rand$

= $(c'\!: (0,..13)+1 \;\land\; d'\!: (0,..13)+1 \;\land\; x'=x \;\land\; y'=y) / 13 / 13.$

if $c < n$ **then** x:=c+d **else** x:=c. **if** $c < n$+1 **then** y:=c+d **else** y:=c.

$y<x\le 14 \lor x\le 14<y$

4 omitted steps

= $(n-1) / 169$

probability that x wins is $(n-1) / 169$

probability that y wins is $(14-n) / 169$

probability of a tie is $12/13$

"under 8" beats both "under 7" and "under 9"

Dice

If you repeatedly throw a pair of six-sided dice until they are equal, how long does it take?

$$R \;\Longleftarrow\; u{:=}\,(rand\ 6)+1.\ \ v{:=}\,(rand\ 6)+1.\ \ \textbf{if}\ u{=}v\ \textbf{then}\ ok\ \textbf{else}\ (t{:=}\,t{+}1.\ \ R)$$

	$u{:=}\,(rand\ 6)+1.\ \ v{:=}\,(rand\ 6)+1.$	replace *rand*
	$\textbf{if}\ u{=}v\ \textbf{then}\ t'{=}t\ \textbf{else}\ (t{:=}\,t{+}1.\ \ (t'{\ge}t)\times(5/6)^{t'-t}\times 1/6)$	Substitution Law
$=$	$(u'{:}\ 1,..7\ \wedge\ v'{=}v\ \wedge\ t'{=}t)/6.\ \ (u'{=}u\ \wedge\ v'{:}\ 1,..7\ \wedge\ t'{=}t)/6.$	replace first .
	$\textbf{if}\ u{=}v\ \textbf{then}\ t'{=}t\ \textbf{else}\ (t'{\ge}t{+}1)\times(5/6)^{t'-t-1}\ /\ 6$	
$=$	$(u', v'{:}\ 1,..7\ \wedge\ t'{=}t)/36.$	replace .
	$\textbf{if}\ u{=}v\ \textbf{then}\ t'{=}t\ \textbf{else}\ (t'{\ge}t{+}1)\times(5/6)^{t'-t-1}\ /\ 6$	replace **if**
$=$	$\Sigma u'', v''{:}\ 1,..7\cdot\ \Sigma t''\cdot\ \ (t''{=}t)/36\times(\ \ (u''{=}v'')\times(t'{=}t'')$	
	$+\ \ (u''{\ne}v'')\times(t'{\ge}t''{+}1)\times(5/6)^{t'-t''-1}\ /\ 6)$	sum
$=$	$(6\times(t'{=}t)\ +\ 30\times(t'{\ge}t{+}1)\times(5/6)^{t'-t-1}\ /\ 6)\ /\ 36$	combine
$=$	$(t'{\ge}t)\times(5/6)^{t'-t}\times 1/6$	

Dice

If you repeatedly throw a pair of six-sided dice until they are equal, how long does it take?

$$R \;\Leftarrow\; u{:=}\,(\mathit{rand}\ 6)+1.\ \ v{:=}\,(\mathit{rand}\ 6)+1.\ \ \textbf{if}\ u{=}v\ \textbf{then}\ \mathit{ok}\ \textbf{else}\ (t{:=}\,t{+}1.\ \ R)$$

The average value of t' is $(t'{\ge}t) \times (5/6)^{t'-t} \times 1/6.\ t \quad = \quad t{+}5$

Functional Programming

– assignment

– dependent composition

\+ functions

specification = function from input to output

program = implemented specification

application √

composition √

selective union √

quantifiers X

program + inputs = function + arguments

example specification $\langle L\text{: } [*rat] \rightarrow \Sigma L\rangle$

$$\Sigma L \ = \ \langle n\text{: } 0,..\#L+1 \rightarrow \Sigma\, L\, [n;..\#L]\rangle\ 0$$

$$\langle n\text{: } 0,..\#L+1 \rightarrow \Sigma\, L\, [n;..\#L]\rangle$$

$$= \quad \langle n\text{: } 0,..\#L\ ,\ \#L \rightarrow \Sigma\, L\, [n;..\#L]\rangle$$

$$= \quad \langle n\text{: } 0,..\#L \rightarrow \Sigma\, L\, [n;..\#L]\rangle \mid \langle n\text{: } \#L \rightarrow \Sigma\, L\, [n;..\#L]\rangle$$

$$\langle n\text{: } 0,..\#L \rightarrow \Sigma\, L\, [n;..\#L]\rangle$$

$$= \quad \langle n\text{: } 0,..\#L \rightarrow Ln + \Sigma\, L\, [n+1;..\#L]\rangle$$

$$\langle n\text{: } \#L \rightarrow \Sigma\, L\, [n;..\#L]\rangle \quad = \quad \langle n\text{: } \#L \rightarrow 0\rangle$$

$$\Sigma\, L\, [n+1;..\#L] \quad = \quad \langle n\text{: } 0,..\#L+1 \rightarrow \Sigma\, L\, [n;..\#L]\rangle\ (n+1)$$

time specification $\langle L\colon [{*}rat] \to \#L\rangle$

$$\#L \;=\; \langle n\colon 0,..\#L{+}1 \to \#L{-}n\rangle\, 0$$

$$
\begin{aligned}
& \langle n\colon 0,..\#L{+}1 \to \#L{-}n\rangle \\
= \quad & \langle n\colon 0,..\#L \;\;,\;\; \#L \to \#L{-}n\rangle \\
= \quad & \langle n\colon 0,..\#L \to \#L{-}n\rangle \mid \langle n\colon \#L \to \#L{-}n\rangle
\end{aligned}
$$

$$
\begin{aligned}
& \langle n\colon 0,..\#L \to \#L{-}n\rangle \\
= \quad & \langle n\colon 0,..\#L \to 1 + \#L{-}n{-}1\rangle
\end{aligned}
$$

$$\langle n\colon \#L \to \#L{-}n\rangle \quad = \quad \langle n\colon \#L \to 0\rangle$$

$$\#L{-}n{-}1 \quad = \quad \langle n\colon 0,..\#L{+}1 \to \#L{-}n\rangle\,(n{+}1)$$

time specification $\langle L: [*rat] \rightarrow \#L \rangle$

$$\#L \;=\; \langle n: 0,..\#L{+}1 \rightarrow \#L{-}n \rangle\ 0$$

$$\langle n: 0,..\#L{+}1 \rightarrow \#L{-}n \rangle$$

$$=\quad \langle n: 0,..\#L\ ,\ \#L \rightarrow \#L{-}n \rangle$$

$$=\quad \langle n: 0,..\#L \rightarrow \#L{-}n \rangle \mid \langle n: \#L \rightarrow \#L{-}n \rangle$$

$$\langle n: 0,..\#L \rightarrow \#L{-}n \rangle$$

$$=\quad \langle n: 0,..\#L \rightarrow \#L{-}n \rangle$$

$$\langle n: \#L \rightarrow \#L{-}n \rangle \quad = \quad \langle n: \#L \rightarrow 0 \rangle$$

$$\#L{-}n \quad = \quad 1 + \langle n: 0,..\#L{+}1 \rightarrow \#L{-}n \rangle\ (n{+}1)$$

Function Refinement

Specification S is unsatisfiable for domain element x : $¢\, S\, x < 1$

Specification S is satisfiable for domain element x : $¢\, S\, x \geq 1$

Specification S is deterministic for domain element x : $¢\, S\, x \leq 1$

Specification S is nondeterministic for domain element x : $¢\, S\, x > 1$

Specification S is satisfiable for domain element x : $\exists y \cdot\ y: S\, x$

Specification S is implementable: $\forall x \cdot\ \exists y \cdot\ y: S\, x$

$\forall x \cdot\ S\, x \neq null$

P is refined by S $S: P$

$P{::}\ S$

example search for an item in a list

$\langle L\text{: } [*int] \to \langle x\text{: } int \to §n\text{: } 0,..\#L\cdot\ Ln = x\ \rangle\rangle$ unimplementable

$\langle L\text{: } [*int] \to \langle x\text{: } int \to \textbf{if } x\text{: } L\ (0,..\#L)\ \textbf{then } §n\text{: } 0,..\#L\cdot\ Ln = x\ \textbf{else } \#L,..\infty\rangle\rangle$

$\textbf{if } x\text{: } L\ (0,..\#L)\ \textbf{then } §n\text{: } 0,..\#L\cdot\ Ln = x\ \textbf{else } \#L,..\infty$ ::

$\langle i\text{: } nat \to \textbf{if } x\text{: } L\ (i,..\#L)\ \textbf{then } §n\text{: } i,..\#L\cdot\ Ln = x\ \textbf{else } \#L,..\infty\rangle\ 0$

$\textbf{if } x\text{: } L\ (i,..\#L)\ \textbf{then } §n\text{: } i,..\#L\cdot\ Ln = x\ \textbf{else } \#L,..\infty$ ::

$\textbf{if } i = \#L\ \textbf{then } \#L$

$\textbf{else if } x = Li\ \textbf{then } i$

$\textbf{else } \langle i\text{: } nat \to \textbf{if } x\text{: } L\ (i,..\#L)\ \textbf{then } §n\text{: } i,..\#L\cdot\ Ln = x\ \textbf{else } \#L,..\infty\rangle\ (i{+}1)$

recursive timing $\langle L \rightarrow \langle x \rightarrow 0,..\#L{+}1\rangle\rangle$

$$0,..\#L{+}1 \;::\; \langle i \rightarrow 0,..\#L{-}i{+}1\rangle\, 0$$

$$0,..\#L{-}i{+}1 \;::\; \textbf{if } i = \#L \textbf{ then } 0$$

$$\textbf{else if } x = Li \textbf{ then } 0$$

$$\textbf{else } 1 + \langle i \rightarrow 0,..\#L{-}i{+}1\rangle\,(i{+}1)$$

$$1 + \langle i \rightarrow 0,..\#L{-}i{+}1\rangle\,(i{+}1)$$

$$= \quad 1 + (0,..\#L{-}(i{+}1){+}1)$$

$$= \quad 1 + (0,..\#L{-}i)$$

$$= \quad 1,..\#L{-}i{+}1$$

$$: \quad 0,..\#L{-}i{+}1$$

functional versus imperative

same programming steps, different notation

functional programming has Application Axiom

$$\langle v\colon D\cdot\ b\rangle\ x \quad = \quad (\text{for } v \text{ substitute } x \text{ in } b\ )$$

imperative programming has Substitution Law

$$x{:=}\ e.\ P \quad = \quad (\text{for } x \text{ substitute } e \text{ in } P\ )$$