

Variable Declaration

$$\mathbf{var} \ x: T \cdot P \quad = \quad \exists x, x': T \cdot P$$

$$\begin{aligned} & \mathbf{var} \ x: \mathit{int} \cdot x:=2. \ y:=x+z \\ = & \exists x, x': \mathit{int} \cdot x'=2 \wedge y' = 2+z \wedge z'=z \\ = & y' = 2+z \wedge z'=z \end{aligned}$$

$$\begin{aligned} & \mathbf{var} \ x: \mathit{int} \cdot y:=x \\ = & \exists x, x': \mathit{int} \cdot x'=x \wedge y'=x \wedge z'=z \\ = & z'=z \end{aligned}$$

$$\begin{aligned} & \mathbf{var} \ x: \mathit{int} \cdot y:=x-x \\ = & y'=0 \wedge z'=z \end{aligned}$$

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

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

Variable Suspension

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

$$\mathbf{frame } w, x. P = P \wedge y'=y \wedge z'=z$$

$$x:=e = \mathbf{frame } x. x' = e$$

$$s:=\Sigma L \Leftarrow \mathbf{frame } s. \mathbf{var } n: \mathit{nat}. s' = \Sigma L$$

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

Array

$$Ai := e \quad = \quad A'i = e \wedge (\forall j. j \neq i \Rightarrow A'j = Aj) \wedge x' = x \wedge y' = y \wedge \dots$$

$$A(A2) := 3$$

$$= \quad A'(A2) = 3 \wedge (\forall j. j \neq A2 \Rightarrow A'j = Aj) \wedge x' = x \wedge y' = y \wedge \dots$$

$$A2 := 3. \quad i := 2. \quad Ai := 4. \quad Ai = A2$$

$$? \quad = \quad A2 := 3. \quad i := 2. \quad 4 = A2$$

$$= \quad A2 := 3. \quad 4 = A2$$

$$? \quad = \quad 4 = 3$$

$$= \quad \perp$$

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

$$? \quad = \quad A2 := 2. \quad A2 = 2$$

$$? \quad = \quad 2 = 2$$

$$= \quad \top$$

$$\begin{aligned}
& Ai := e \\
= & A' i = e \wedge (\forall j. j \neq i \Rightarrow A' j = A j) \wedge x' = x \wedge y' = y \wedge \dots \\
= & A' = i \rightarrow e \mid A \wedge x' = x \wedge y' = y \wedge \dots \\
= & A := i \rightarrow e \mid A
\end{aligned}$$

$$\begin{aligned}
& A := 2 \rightarrow 3 \mid A. \quad i := 2. \quad A := i \rightarrow 4 \mid A. \quad Ai = A2 \\
= & A := 2 \rightarrow 3 \mid A. \quad i := 2. \quad (i \rightarrow 4 \mid A) i = (i \rightarrow 4 \mid A) 2 \\
= & A := 2 \rightarrow 3 \mid A. \quad (2 \rightarrow 4 \mid A) 2 = (2 \rightarrow 4 \mid A) 2 \\
= & A := 2 \rightarrow 3 \mid A. \quad \top \\
= & \top
\end{aligned}$$

$$\begin{aligned}
& A := 2 \rightarrow 2 \mid A. \quad A := A 2 \rightarrow 3 \mid A. \quad A 2 = 2 \\
= & A := 2 \rightarrow 2 \mid A. \quad (A 2 \rightarrow 3 \mid A) 2 = 2 \\
= & ((2 \rightarrow 2 \mid A) 2 \rightarrow 3 \mid 2 \rightarrow 2 \mid A) 2 = 2 \\
= & (2 \rightarrow 3 \mid 2 \rightarrow 2 \mid A) 2 = 2 \\
= & 3 = 2 \\
= & \perp
\end{aligned}$$

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

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

update in place, like $i := i+1$

$A := i \rightarrow e \mid B$

update in place if B not used further

$A := i \rightarrow Aj \mid j \rightarrow Ai \mid A$

do not update in place, like $i := i+1+i$

Record

```
person =    "name" → text  
           | "age" → nat
```

```
var p: person
```

```
p := "name" → "Josh" | "age" → 17
```

```
p "age" := 18
```

```
p := "age" → 18 | p
```

While Loop

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

means

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

Example: prove

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

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

Proof:

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

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

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

$$\mathbf{else } ok$$

Exit Loop

$L \Leftarrow$ **loop**

A.

exit when *b*.

C

end

means

$L \Leftarrow A.$ **if *b* then *ok* 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. if c then ok else (D. Q)

```

```

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

```

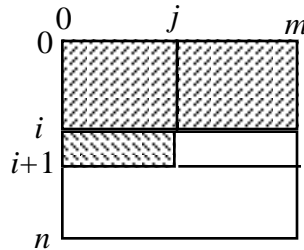
means

$P \Leftarrow A.$ **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

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



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

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

$P \Leftarrow i := 0. i \leq n \Rightarrow Q$

$i \leq n \Rightarrow Q \Leftarrow \text{if } i = n \text{ then } j := m \text{ else } i < n \Rightarrow Q$

$i < n \Rightarrow Q \Leftarrow j := 0. i < n \wedge j \leq m \Rightarrow R$

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

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

$P \Leftarrow i := 0. L0$

$L0 \Leftarrow \text{if } i=n \text{ then } j:=m \text{ else } (j:=0. L1)$

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

else if $A[i][j]=x$ **then** *ok*

else $(j:=j+1. L1)$

in C:

$i = 0;$

L0: $\text{if } (i==n) j = m;$

else { $j = 0;$

$L1: \text{if } (j==m) \{i = i+1; \text{goto } L0;\}$

$\text{else if } (A[i][j]==x);$

$\text{else } \{j = j+1; \text{goto } L1;\}$

}

$$t' \leq t + n \times m \iff i := 0. i \leq n \Rightarrow t' \leq t + (n-i) \times m$$

$$i \leq n \Rightarrow t' \leq t + (n-i) \times m \iff$$

$$\mathbf{if } i=n \mathbf{ then } j:=m \mathbf{ else } i < n \Rightarrow t' \leq t + (n-i) \times m$$

$$i < n \Rightarrow t' \leq t + (n-i) \times m \iff$$

$$j:=0. i < n \wedge j \leq m \Rightarrow t' \leq t + (n-i) \times m - j$$

$$i < n \wedge j \leq m \Rightarrow t' \leq t + (n-i) \times m - j \iff$$

$$t := t + 1.$$

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

$$\mathbf{else } i < n \wedge j < m \Rightarrow t' \leq t + (n-i) \times m - j$$

$$i < n \wedge j < m \Rightarrow t' \leq t + (n-i) \times m - j \iff$$

$$\mathbf{if } A \ i \ j = x \mathbf{ then } ok$$

$$\mathbf{else } (j := j + 1. i < n \wedge j \leq m \Rightarrow t' \leq t + (n-i) \times m - j)$$

For Loop

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

i is a fresh name (a local constant)

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

P is a specification

If $F_{mn} \Leftarrow m=n \wedge ok$

and $F_{ik} \Leftarrow m \leq i < j < k \leq n \wedge (F_{ij}. F_{jk})$

then $F_{mn} \Leftarrow \mathbf{for} \ i := m; ..n \ \mathbf{do} \ m \leq i < n \Rightarrow F_{i(i+1)}$

Example: integer variable x . Define

$$F = \lambda i, j: nat. x' = x \times 2^{j-i}$$

Then

$$x' = 2^n \Leftarrow x := 1. F_{0n}$$

$$F_{0n} \Leftarrow \mathbf{for} \ i := 0; ..n \ \mathbf{do} \ F_{i(i+1)}$$

$$F_{i(i+1)} \Leftarrow x := 2 \times x$$

Example:

$$t' = t + \sum_{i: m, \dots, n} f_i \iff \text{for } i := m; \dots, n \text{ do } t' = t + f_i$$

If $f_i = c$ (a constant) then

$$t' = t + (n - m) \times c \iff \text{for } i := m; \dots, n \text{ do } t' = t + c$$

Example: add 1 to each item in a list

$$\#L' = \#L \wedge \forall i: 0, \dots, \#L. L'i = Li + 1$$

Fik describes an arbitrary segment of iterations:

$$\begin{aligned} Fik &= \#L' = \#L \\ &\wedge (\forall j: i, \dots, k. L'j = Lj + 1) \\ &\wedge (\forall j: (0, \dots, i), (k, \dots, \#L). L'j = Lj) \end{aligned}$$

check

$$F 0 (\#L) \iff 0 = \#L \wedge ok$$

$$Fik \iff 0 \leq i < j < k \leq \#L \wedge (Fij. Fjk)$$

So

$$F 0 (\#L) \iff \text{for } i := 0; \dots, \#L \text{ do } Fi(i+1)$$

$$Fi(i+1) \iff L := i \rightarrow Li + 1 \mid L$$

$$I_m \Rightarrow I'_n \iff \mathbf{for } i := m; ..n \mathbf{ do } m \leq i < n \wedge I_i \Rightarrow I'(i+1)$$

Let $I_i = x = 2^i$

$$x' = 2^n \iff x := 1. I_0 \Rightarrow I'_n$$

$$I_0 \Rightarrow I'_n \iff \mathbf{for } i := 0; ..n \mathbf{ do } I_i \Rightarrow I'(i+1)$$

$$I_i \Rightarrow I'(i+1) \iff x := 2 \times x$$

Go To

acceptable in an execution language

jump, branch

unacceptable in a programming language

no specification

Time Dependence

deadline := $t + 5$

no problem

if $t < \textit{deadline}$ **then** ... **else** ...

no problem

$t := 5$

problem: unimplementable

wait until w = $t := \max t w$

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

Assertions

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

adds robustness

ensure b = **if** b **then** ok **else** \perp

$$= b \wedge ok$$

unimplementable by itself, but may be used in some contexts

Nondeterministic choice (a programming notation):

$$P \text{ or } Q = P \vee Q$$

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

$$= \exists x'', y''. (x'' = 0 \wedge y'' = y \vee x'' = 1 \wedge y'' = y)$$

$$\wedge x'' = 1 \wedge x' = x'' \wedge y' = y''$$

$$= x' = 1 \wedge y' = y$$

$$= x := 1$$

implementation: backtracking

Result Expression

P result e

$$x' = (P \text{ result } e) = P. x'=e$$

var *term, sum: rat := 1.*

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

result *sum*

$$y:= y+1 \text{ result } y = y+1$$

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

$$= x' = (y:= y+1 \text{ result } y) \wedge y'=y$$

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

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

$$= x:= y+1$$

Side-Effects

$$x = x \quad ?$$

$$x + x = 2 \times x \quad ?$$

not if $x = (y := y+1 \text{ **result** } y)$ with side-effect $y := y+1$

$$x := (P \text{ **result** } e)$$

becomes

$$(P. x := e)$$

$$x := y + (P \text{ **result** } e)$$

becomes

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

Don't neglect the time for expression evaluation.

Function

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

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

```
bexp =  $\lambda n: int.$ 
      ( var r: int := 1.
        for i:= 0;..n do r:= r×2.
        assert r: int
        result r )
```

Procedure

procedure = name of procedure
 + parameters
 + scope control

$$P = \lambda x: int. a' < x < b'$$

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

$$P\ (a+1) = a' < a+1 < b'$$

$$a' < x < b' \iff a := x-1. b := x+1$$

$$(\lambda p: D. B)\ a = (\mathbf{var}\ p: D := a. B)$$

reference parameter (**var** parameter)

$$\begin{aligned}
 & (\lambda^*x: \text{int}. x:= 3. b:= a) a \\
 = & a:= 3. b:= a \\
 = & a' = b' = 3
 \end{aligned}$$

$$(\lambda^*p: D \cdot B) a = (\lambda p, p': D \cdot B) a a'$$

Warning: substitute arguments for parameters before any other manipulations. Do not manipulate the procedure body by itself.

Apply programming theory separately for each call.

$$x:= 3. b:= a = b:= a. x:= 3 ?$$

Alias

i : integer variable

A : array

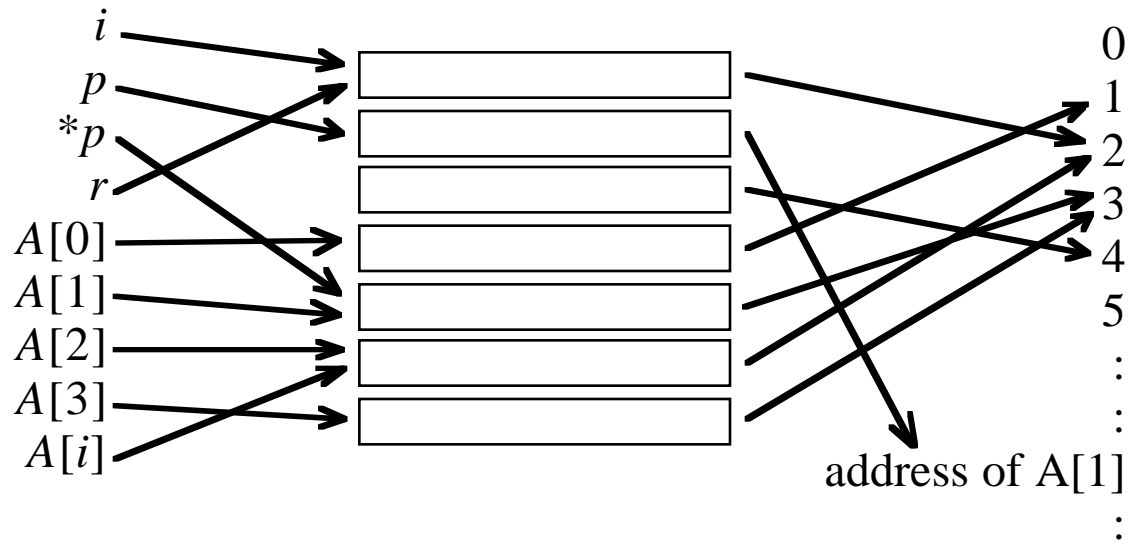
r : reference parameter

p : pointer

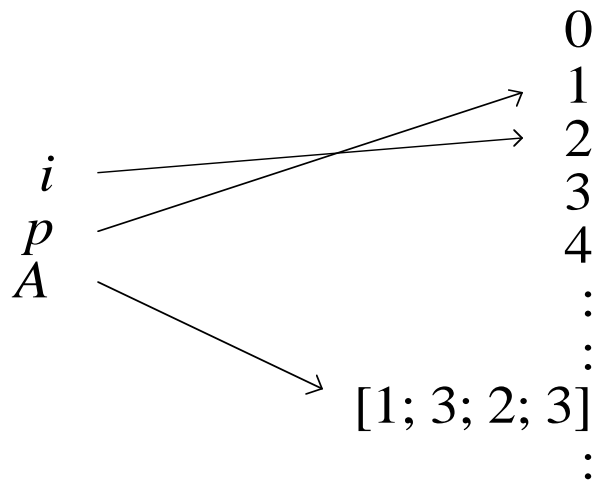
r, i	2
p	address of $A[1]$
	4
$A[0]$	1
$*p, A[1]$	3
$A[i], A[2]$	2
$A[3]$	3

“... the necessary prerequisite for being able to think in terms of safely independent variables is, of course, the condition that no part of the store may assume more than a single name.” [Wirth, 1974]

two-level mapping



one-level mapping



Probabilistic Programming

probability: real number between 0 and 1

$$prob = \{r: real \cdot 0 \leq r \leq 1\}$$

$$\top = 1$$

$$\perp = 0$$

distribution: value is a probability, sum is 1

tells the frequency of occurrence of values of its variables

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

$$(\forall n: nat+1. 2^{-n}: prob) \wedge (\sum n: nat+1. 2^{-n})=1$$

2^{-n} says that n has value 3 one-eighth of the time

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

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

2^{-n-m} says that n has value 3 and m has value 1

one-sixteenth of the time

The average value of number expression e as variables v vary over their domains according to distribution p is

$$\sum v \cdot e \times p$$

The average value of n^2 as n varies over $\text{nat}+1$ according to distribution 2^{-n} is $\sum n: \text{nat}+1 \cdot n^2 \times 2^{-n}$, which is 6.

The average value of $n-m$ as n and m vary over $\text{nat}+1$ according to distribution 2^{-n-m} is $\sum n, m: \text{nat}+1 \cdot (n-m) \times 2^{-n-m}$, which is 0.

Let S be an implementable deterministic specification.

Let p be the distribution describing the initial state σ .

The distribution describing the final state σ' is

$$\sum \sigma \cdot S \times p$$

Example: integer variables x and y

Let $X = (x=7) / 3 + (x=8) \times 2/3$ distribution of x

Let $Y = (y=7) / 3 + (y=8) \times 2/3$ distribution of y

The probability that x has value 7 is

$$\begin{aligned} & (7=7) \times 1/3 + (7=8) \times 2/3 \\ = & \top \times 1/3 + \perp \times 2/3 \\ = & 1 \times 1/3 + 0 \times 2/3 \\ = & 1/3 \end{aligned}$$

The probability that x has value 8 is $2/3$

The probability that x has value 9 is 0

Let $S = \mathbf{if} \ x=y \ \mathbf{then} \ (x:=0, \ y:=0) \ \mathbf{else} \ (x:=abs(x-y), \ y:=1)$

The distribution of final states is

$$\begin{aligned} & \Sigma_{x, y} \cdot S \times X \times Y \\ = & \Sigma_{x, y} \cdot (x=y \wedge x'=y'=0 \vee x \neq y \wedge x'=abs(x-y) \wedge y'=1) \\ & \quad \times ((x=7) / 3 + (x=8) \times 2/3) \\ & \quad \times ((y=7) / 3 + (y=8) \times 2/3) \\ = & (x'=y'=0) \times 5/9 + (x'=y'=1) \times 4/9 \end{aligned}$$

If b is a probability, and P and Q are distributions of final states,

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

$$P.Q = \Sigma \sigma'' \cdot (\text{substitute } \sigma'' \text{ for } \sigma' \text{ in } P) \\ \times (\text{substitute } \sigma'' \text{ for } \sigma \text{ in } Q)$$

are distributions of final states.

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** $x := x+5$

$$= \Sigma x'' \cdot ((x''=0)/3 + (x''=1) \times /3) \\ \times ((x''=0) \times ((x'=x''+2)/2 + (x'=x''+3)/2) \\ + (x'' \neq 0) \times ((x'=x''+4)/4 + (x'=x''+5) \times 3/4)) \\ = (x'=2)/6 + (x'=3)/6 + (x'=5)/6 + (x'=6)/2$$

$$\Sigma \sigma \cdot S \times p = p' \cdot S$$

Random Number Generators

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

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

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

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

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

$x := \text{rand } 2. \quad x := x + \text{rand } 3$

replace the two *rands* with r and s

$= \sum_{r: 0,..2} \cdot \sum_{s: 0,..3} \cdot (x := r. \quad x := x + s) \times 1/2 \times 1/3$

Substitution Law

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

sum

$= ((x' = 0+0) + (x' = 0+1) + (x' = 0+2)$

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

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

$$\begin{aligned}
& x := \text{rand } 2. \quad x := x + \text{rand } 3 && \text{replace assignments} \\
= & (x': 0, \dots, 2) / 2. \quad (x': x + (0, \dots, 3)) / 3 && \text{dependent composition} \\
= & \sum x'' \cdot (x'': 0, \dots, 2) / 2 \times (x': x'' + (0, \dots, 3)) / 3 && \text{sum} \\
= & 1/2 \times (x': 0, \dots, 3) / 3 + 1/2 \times (x': 1, \dots, 4) / 3 \\
= & (x'=0) / 6 + (x'=1) / 3 + (x'=2) / 3 + (x'=3) / 6
\end{aligned}$$

$\text{rand } 8 < 3$ has boolean value b with distribution

$$\begin{aligned}
& \sum r: 0, \dots, 8 \cdot (b = (r < 3)) / 8 \\
= & (b = \top) \times 3/8 + (b = \perp) \times 5/8 \\
= & 5/8 - b/4
\end{aligned}$$

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.

$$\begin{aligned}
 & x := (\text{rand } 13) + 1. \quad \mathbf{if } x < 7 \quad \mathbf{then } x := x + (\text{rand } 13) + 1 \quad \mathbf{else } \textit{ok} \\
 & \hspace{20em} \textit{replace } \textit{rand} \textit{ and } \textit{ok} \\
 = & \quad (x' : (0, \dots, 13) + 1) / 13. \quad \mathbf{if } x < 7 \quad \mathbf{then } (x' : x + (0, \dots, 13) + 1) / 13 \quad \mathbf{else } x' = x \\
 & \hspace{20em} \textit{replace } . \textit{ and } \mathbf{if} \\
 = & \quad \sum_{x''} (x'' : 1, \dots, 14) / 13 \\
 & \quad \times ((x'' < 7) \times (x' : x'' + 1, \dots, x'' + 14) / 13 + (x'' \geq 7) \times (x' = x'')) \\
 & \hspace{15em} \textit{by several omitted steps} \\
 = & \quad ((2 \leq x' < 7) \times (x' - 1) + (7 \leq x' < 14) \times 19 + (14 \leq x' < 20) \times (20 - x')) / 169
 \end{aligned}$$

Player x plays “under n ” and player y plays “under $n+1$ ”

$c := (\text{rand } 13) + 1. \quad d := (\text{rand } 13) + 1.$

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

$y < x \leq 14 \vee x \leq 14 < y$

Replace *rand* and use the functional-imperative law twice.

$= (c': (0, \dots, 13) + 1 \wedge d': (0, \dots, 13) + 1 \wedge x' = x \wedge y' = y) / 13 / 13.$

$x := \text{if } c < n \text{ then } c + d \text{ else } c. \quad y := \text{if } c < n + 1 \text{ then } c + d \text{ else } c.$

$y < x \leq 14 \vee x \leq 14 < y$

Use the substitution law twice.

$= (c': (0, \dots, 13) + 1 \wedge d': (0, \dots, 13) + 1 \wedge x' = x \wedge y' = y) / 169.$

$(\text{if } c < n + 1 \text{ then } c + d \text{ else } c) < (\text{if } c < n \text{ then } c + d \text{ else } c) \leq 14$

$\vee (\text{if } c < n \text{ then } c + d \text{ else } c) \leq 14 < (\text{if } c < n + 1 \text{ then } c + d \text{ else } c)$

$= (c': (0, \dots, 13) + 1 \wedge d': (0, \dots, 13) + 1 \wedge x' = x \wedge y' = y) / 169.$

$c = n \wedge d > 14 - n$

$= \Sigma c'', d'', x'', y''.$

$(c'': (0, \dots, 13) + 1 \wedge d'': (0, \dots, 13) + 1 \wedge x'' = x \wedge y'' = y) / 169$

$\times (c'' = n \wedge d'' > 14 - n)$

$= \Sigma d'': 1, \dots, 14 \cdot (d'' > 14 - n) / 169$

$= c = n - 1 \wedge d > 13 - n$

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

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

probability of a tie is $12/13$

For $n < 8$, “under $n+1$ ” beats “under n ”.

For $n \geq 8$, “under n ” beats “under $n+1$ ”.

Apparently “under 8” is the best strategy.

Dice

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

$$R \Leftarrow u := (\text{rand } 6) + 1. \quad v := (\text{rand } 6) + 1.$$

$$\mathbf{if } u=v \mathbf{ then } ok \mathbf{ else } (t := t+1. \quad R)$$

hypothesis: t' has the distribution $(t' \geq t) \times (5/6)^{t'-t} \times 1/6$

$$\begin{aligned}
 & u := (\text{rand } 6) + 1. \quad v := (\text{rand } 6) + 1. && \text{replace } rand \\
 & \mathbf{if } u=v \mathbf{ then } t'=t \mathbf{ else } (t := t+1. \quad (t' \geq t) \times (5/6)^{t'-t} \times 1/6) && \text{Subs Law} \\
 = & ((u': 1,..7) \wedge v'=v \wedge t'=t)/6. \quad (u'=u \wedge (v': 1,..7) \wedge t'=t)/6. && \text{replace } . \\
 & \mathbf{if } u=v \mathbf{ then } t'=t \mathbf{ else } (t' \geq t+1) \times (5/6)^{t'-t-1} / 6 && \text{simplify} \\
 = & ((u': 1,..7) \wedge (v': 1,..7) \wedge t'=t)/36. && \text{replace } . \\
 & \mathbf{if } u=v \mathbf{ then } t'=t \mathbf{ else } (t' \geq t+1) \times (5/6)^{t'-t-1} / 6 && \text{replace } \mathbf{if} \\
 = & \sum u'', v'': 1,..7. \sum t'' \cdot (t''=t)/36 \times ((u''=v'') \times (t'=t'')) \\
 & \quad \quad \quad + (u'' \neq v'') \times (t' \geq t''+1) \times (5/6)^{t'-t''-1} / 6) && \text{sum} \\
 = & 1/36 \times (6 \times (t'=t) + 30 \times (t' \geq t+1) \times (5/6)^{t'-t-1} / 6) && \text{combine} \\
 = & (t' \geq t) \times (5/6)^{t'-t} \times 1/6
 \end{aligned}$$

The average value of t' is $\sum t' \cdot t' \times (t' \geq t) \times (5/6)^{t'-t} \times 1/6 = t+5$

Functional Programming

specification $\lambda L: [*rat] \cdot \Sigma L$

$$\begin{aligned} \Sigma L &= (\lambda n: 0, ..\#L+1 \cdot \Sigma L [n; ..\#L]) 0 \\ &= (\lambda n: 0, ..\#L, \#L \cdot \Sigma L [n; ..\#L]) 0 \end{aligned}$$

$$\begin{aligned} &\lambda n: 0, ..\#L+1 \cdot \Sigma L [n; ..\#L] \\ = &(\lambda n: 0, ..\#L \cdot \Sigma L [n; ..\#L]) \mid (\lambda n: \#L \cdot \Sigma L [n; ..\#L]) \end{aligned}$$

$$\begin{aligned} &\lambda n: 0, ..\#L \cdot \Sigma L [n; ..\#L] \\ = &\lambda n: 0, ..\#L \cdot Ln + \Sigma L [n+1; ..\#L] \end{aligned}$$

$$\lambda n: \#L \cdot \Sigma L [n; ..\#L] = \lambda n: \#L \cdot 0$$

$$\Sigma L [n+1; ..\#L] = (\lambda n: 0, ..\#L+1 \cdot \Sigma L [n; ..\#L]) (n+1)$$

time specification $\lambda L: [*rat] \cdot \#L$

suppose we charge 1 for addition, 0 for all else

$$\#L = (\lambda n: 0, \dots, \#L+1 \cdot \#L-n) 0$$

$$\lambda n: 0, \dots, \#L+1 \cdot \#L-n$$

$$= (\lambda n: 0, \dots, \#L \cdot \#L-n) \mid (\lambda n: \#L \cdot \#L-n)$$

$$\lambda n: 0, \dots, \#L \cdot \#L-n$$

$$= \lambda n: 0, \dots, \#L \cdot 1 + \#L-n-1$$

$$\lambda n: \#L \cdot \#L-n = \lambda n: \#L \cdot 0$$

$$\#L-n-1 = (\lambda n: 0, \dots, \#L+1 \cdot \#L-n) (n+1)$$

time specification $\lambda L: [*rat] \cdot \#L$

recursive time: charge 1 for recursive call, 0 for all else

$$\#L = (\lambda n: 0, \dots, \#L+1 \cdot \#L-n) 0$$

$$\lambda n: 0, \dots, \#L+1 \cdot \#L-n$$

$$= (\lambda n: 0, \dots, \#L \cdot \#L-n) \mid (\lambda n: \#L \cdot \#L-n)$$

$$\lambda n: 0, \dots, \#L \cdot \#L-n$$

$$= \lambda n: 0, \dots, \#L \cdot \#L-n$$

$$\lambda n: \#L \cdot \#L-n = \lambda n: \#L \cdot 0$$

$$\#L-n = 1 + (\lambda n: 0, \dots, \#L+1 \cdot \#L-n) (n+1)$$

Function Refinement

Specification S is unsatisfiable for domain element x :

$$\wp S x = 0$$

Specification S is deterministic for domain element x :

$$\wp S x = 1$$

Specification S is nondeterministic for domain element x :

$$\wp S x \geq 2$$

Specification S is satisfiable for domain element x :

$$\exists y. y: S x$$

Specification S is implementable:

$$\forall x. \exists y. y: S x$$

$$\forall x. S x \neq \text{null}$$

$P :: S$ “ P is refined by S ”
 $= S : P$

problem: search for an item in a list.

$\lambda L: [*int]. \lambda x: int. \S n: 0,..\#L. Ln = x$ is unimplementable

$\lambda L: [*int]. \lambda x: int.$

if $x: L (0,..\#L)$ **then** $\S n: 0,..\#L. Ln = x$ **else** $\#L,..\infty$

if $x: L (0,..\#L)$ **then** $\S n: 0,..\#L. Ln = x$ **else** $\#L,..\infty ::$

$(\lambda i: nat. \mathbf{if} x: L (i,..\#L) \mathbf{then} \S n: i,..\#L. Ln = x$
 $\mathbf{else} \#L,..\infty) 0$

if $x: L (i,..\#L)$ **then** $\S n: i,..\#L. Ln = x$ **else** $\#L,..\infty ::$

if $i = \#L$ **then** $\#L$

else if $x = Li$ **then** i

else $(\lambda i. \mathbf{if} x: L (i,..\#L) \mathbf{then} \S n: i,..\#L. Ln = x$
 $\mathbf{else} \#L,..\infty) (i+1)$

recursive timing $\lambda L \cdot \lambda x \cdot 0, \dots, \#L+1$

$0, \dots, \#L+1 :: (\lambda i \cdot 0, \dots, \#L-i+1) 0$

$0, \dots, \#L-i+1 ::$ **if** $i = \#L$ **then** 0

else if $x = Li$ **then** 0

else $1 + (\lambda i \cdot 0, \dots, \#L-i+1) (i+1)$