2005-9-7 p. 235 precedence of $\surd$ moved from level 12 to level 2

2005-9-15 p.235 change “means” to “means the same as” 3 times.

2005-9-22 p.210 mid: change “earlier (1990) version” to “earlier (1984, 1990) version”.

2005-9-26 p.0 “A closely related theory is Dijkstra's” to “A closely related theory uses Dijkstra's”.

2005-9-26 p.34 change “Our example state space is infinite” to “Our example state space in the previous paragraph is infinite”.

2005-9-26 p.40 change “In each, the problem (left side) follows from the solution (right side)” to “In each, the problem (left side) is refined by (follows from, is implied by) the solution (right side)”.

2005-10-3 p.32 change “We define *LIM* ” to “We define the *LIM* quantifier”.

2005-10-3 p.33 final formula: make the colon plain (not italic).

2005-10-4 p.-1 increase the section number of 10.0 to 10.8 by 1, then add 10.0 Preface and make the same changes in Chapter 10.

2005-10-18 p.155 Exercise 45 is now called “Cantor's paradise”. Also p.215.

2005-11-2 p.53 change the two sentences “Surprisingly, ... rather than two.” to “According to the recursive measure, the worst case time is not improved at all, and the average time is improved slightly by a factor of $(\#L)/(\#L+1)$ assuming equal probability of finding the item at each index and not finding it at all. And according to the real time measure, both the worst case and average execution times are a lot worse because the loop contains three tests rather than two.”.

2005-11-19 p.54 and p.55 the hint “expand assignment” occurs 3 times, and each is wrong. The first occurrence is on p.54, and it should be changed to the folloing.

$(h<j \Rightarrow R \;\Leftarrow\; j\text{–}h = 1 \;\wedge\; (p{:=}\,Lh{=}x))$ portation

$=\quad j\text{–}h = 1 \;\wedge\; (p{:=}\,Lh{=}x) \;\Rightarrow\; R$ expand assignment and R

$=\quad j\text{–}h = 1 \;\wedge\; p'{=}(Lh{=}x) \;\wedge\; h'{=}h \;\wedge\; i'{=}i \;\wedge\; j'{=}j \;\Rightarrow\; (x{:}\,L\,(h,..j) = p' \Rightarrow Lh' = x)$ use the antecedent as context to simplify the consequent

$=\quad j\text{–}h = 1 \;\wedge\; p'{=}(Lh{=}x) \;\wedge\; h'{=}h \;\wedge\; i'{=}i \;\wedge\; j'{=}j \;\Rightarrow\; (x{=}Lh = Lh{=}x \Rightarrow Lh{=}x)$ Symmetry and Base and Reflexive Laws

$=\quad \mathsf{T}$

The second occurrence is on p.55 and it should be changed to the following.

$(j\text{–}h{\geq}2 \;\Rightarrow\; h'{=}h{<}i'{<}j{=}j' \;\Leftarrow\; i{:=}\,div\;(h{+}j)\;2)$ expand assignment

$=\quad (j\text{–}h{\geq}2 \;\Rightarrow\; h'{=}h{<}i'{<}j{=}j' \;\Leftarrow\; i' = div\;(h{+}j)\;2 \;\wedge\; p'{=}p \;\wedge\; h'{=}h \;\wedge\; j'{=}j)$ use the equations in the antecedent as context to simplify the consequent

$=\quad (j\text{–}h{\geq}2 \;\Rightarrow\; h = h < div\;(h{+}j)\;2 < j = j \;\Leftarrow\; i' = div\;(h{+}j)\;2 \wedge p'{=}p \wedge h'{=}h \wedge j'{=}j)$ simplify $h{=}h$ and $j{=}j$ and use the properties of *div*

$=\quad (j\text{–}h{\geq}2 \;\Rightarrow\; \mathsf{T} \;\Leftarrow\; i' = div\;(h{+}j)\;2 \;\wedge\; p'{=}p \;\wedge\; h'{=}h \;\wedge\; j'{=}j)$ base law twice

$=\quad \mathsf{T}$

The last occurrence is on p.55 and it should be changed to the following.

$(U \Leftarrow j{-}h = 1 \ \wedge \ (p{:=}\, Lh{=}x))$ expand U and the assignment

$= \quad (h{<}j \Rightarrow t' \le t + ceil\ (log\ (j{-}h)) \Leftarrow j{-}h{=}1 \wedge p'{=}(Lh{=}x) \wedge h'{=}h \wedge i'{=}i \wedge j'{=}j \wedge t'{=}t)$

use main antecedent as context in main consequent

$= \quad (h{<}j \Rightarrow t \le t + ceil\ (log\ 1) \Leftarrow j{-}h{=}1 \wedge p'{=}(Lh{=}x) \wedge h'{=}h \wedge i'{=}i \wedge j'{=}j \wedge t'{=}t)$

Use $log\ 1 = 0$

$= \quad (h{<}j \Rightarrow \mathsf{T} \Leftarrow j{-}h{=}1 \wedge p'{=}(Lh{=}x) \wedge h'{=}h \wedge i'{=}i \wedge j'{=}j \wedge t'{=}t)$ base law twice

$= \quad \mathsf{T}$

---

p.53 “introduce variable j ” becomes “introduce natural variables i and j ”

---

p.55 replace first existentially quantified line and following line by

$= \quad (\exists h'', i'', j'', p'' \cdot \ h''{=}h{<}i''{<}j{=}j'' \wedge Li'' \le x$

$\wedge \ (i''{<}j'' \Rightarrow (x{:}\ L\ (i'',..j'') \ = \ p' \implies Lh' = x)))$

$\Rightarrow R$ eliminate p'' , h'' , and j'' by one-point, and rename i'' to i

---

p.104 near bottom: change “two of those items may be lists themselves” to “two of those items are lists themselves”

p.108 middle becomes:

The axioms use an auxiliary specification that helps in writing the axioms, but is not an addition to the theory, and does not need to be implemented: *work* means “Do anything, wander around changing the values of nodes if you like, but do not *go* from this node (your location at the start of *work*) in this direction (the value of variable *aim* at the start of *work*). End where you started, facing the way you were facing at the start.”. Here are the axioms.

$(aim{=}up) = (aim' {\neq} up) \Leftarrow go$

$node'{=}node \wedge aim'{=}aim \Leftarrow go.\ work.\ go$

$work \Leftarrow ok$

$work \Leftarrow node{:=}\, x$

$work \Leftarrow a{=}aim {\neq} b \wedge (aim{:=}\, b.\ go.\ work.\ go.\ aim{:=}\, a)$

$work \Leftarrow work.\ work$

---

p.146 change the final paragraph to the following two paragraphs:

In this book we have looked only at small programs. But the theory is not limited to small programs; it is independent of scale, applicable to any size of software. In a large software project, the first design decision might be to divide the task into several pieces that will fit together in some way. This decision can be written as a refinement, specifying exactly what the parts are and how they fit together, and then the refinement can be proven. Using the theory in the early stages is enormously beneficial, because if an early step is wrong, it is enormously costly to correct later.

For a theory of programming to be in widespread use for industrial program design, it must be supported by tools. Ideally, an automated prover checks each refinement, remaining silent if the refinement is correct, complaining whenever there is a mistake, and saying exactly what is wrong. At present there are a few tools that provide some assistance, but they are far from ideal. There is plenty of opportunity for tool builders, and they need a thorough knowledge of a practical theory of programming.

---

p.16 on the line after

$i: x,..y \;=\; x \le i < y$

insert "where i is an integer.". And on p.227 change the two lines

$i: x,..y \;=\; x \le i < y$ (for extended integers i , x , y , $x \le y$)

$\not\in(x,..y) \;=\; y - x$ (for extended integers x , y , $x \le y$)

to

$i: int \;\wedge\; x, y: xint \;\wedge\; x \le y \;\Rightarrow\; (i: x,..y \;=\; x \le i < y)$

$x, y: xint \;\wedge\; x \le y \;\Rightarrow\; \not\in(x,..y) \;=\; y - x$

---

p.52 mid change paragraph as follows:

Refinement by Parts says that if the same refinement structure can be used for two specifications, then it can be used for their conjunction. If we add $t := t+1$ to the refinements that were not concerned with time, it won't affect their proof, and then we have the same refinement structure for both $\neg\, x: L\,(0,..h') \;\wedge\; (Lh'=x \vee h'=\#L)$ and $t' \le t+\#L$, so we know it works for their conjunction, and that solves the original problem. We could have divided $\neg\, x: L\,(0,..h') \;\wedge\; (Lh'=x \vee h'=\#L)$ into parts also. And of course we should prove our refinements.

---

p.32 change the first sentence of section 3.3 to "For most purposes, a list can be regarded as a kind of function; the domain of list L is $0,..\#L$. And conversely, a function whose domain is $0,..n$ for some natural n , and whose body is an item, can be regarded as a kind of list.".

---

p.128 middle change "intermediate" to "current".

---

p.128 and p.136 change the bottom to

$thermostat \;=\; (gas := \bot \parallel spark := \bot).\; GasIsOff$

$GasIsOff \;=\; \textbf{if}\; temperature < desired - \varepsilon$
$\textbf{then}\; ((gas := \top \parallel spark := \top \parallel t+1 \le t' \le t+3).\; spark := \bot.\; GasIsOn)$
$\textbf{else}\; (((\textbf{frame}\; gas, spark \cdot ok) \parallel t < t' \le t+1).\; GasIsOff)$

$GasIsOn \;=\; \textbf{if}\; temperature < desired + \varepsilon \;\wedge\; flame$
$\textbf{then}\; (((\textbf{frame}\; gas, spark \cdot ok) \parallel t < t' \le t+1).\; GasIsOn)$
$\textbf{else}\; ((gas := \bot \parallel (\textbf{frame}\; spark \cdot ok) \parallel t+20 \le t' \le t+21).\; GasIsOff)$

p.137 change the top to

$thermostat \;=\; ((gasin!\, \bot.\; gasack?) \parallel (sparkin!\, \bot.\; sparkack?)).\; GasIsOff$

$GasIsOff \;=\; ((temperaturereq!.\; temperature?) \parallel (desiredreq!.\; desired?)).$
$\textbf{if}\; temperature < desired - \varepsilon$
$\textbf{then}\; (\; ((gasin!\, \top.\; gasack?) \parallel (sparkin!\, \top.\; sparkack?) \parallel t+1 \le t' \le t+3).$
$\quad sparkin!\, \bot.\; sparkack?.\; GasIsOn\;)$
$\textbf{else}\; (t < t' \le t+1.\; GasIsOff)$

$GasIsOn \;=\; (\; (temperaturereq!.\; temperature?) \parallel (desiredreq!.\; desired?)$
$\parallel (flamereq!.\; flame?)\;).$
$\textbf{if}\; temperature < desired + \varepsilon \;\wedge\; flame$
$\textbf{then}\; (t < t' \le t+1.\; GasIsOn)$
$\textbf{else}\; (((gasin!\, \bot.\; gasack?) \parallel t+20 \le t' \le t+21).\; GasIsOff)$

---

p.191 last line: delete the Hint.

---

p.127 middle, change
"All of the specification laws and refinement laws survive the addition of interactive variables, with one sad exception: the Substitution Law no longer works."
to
"Most of the specification laws and refinement laws survive the addition of interactive variables, but sadly, the Substitution Law no longer works.".

---

p.137, between "End of Monitor" and "9.1.6 Reaction Controller" put
The calculation of space requirements when there is concurrency may sometimes require a monitor for the space variable, so that any process can request an update, and the updates can be communicated to all processes. The monitor for the space variable is also the arbiter between competing space allocation requests.

---

p.145, a paragraph has been slightly rewritten, and it becomes two paragraphs:
"One problem with testing is: how do you know if the outputs are right? Some programs give answers to questions whose answers you do not already know (that is why you wrote the program), and you cannot test them to see if they are right. In that case, you should test to see at least if the answers are reasonable. For some programs, for example, graphics programs for producing pretty pictures, the only way to know if the output is right is to test the program and judge the result.

The other problem with testing is: you cannot try all inputs. Even if all the test cases you try give reasonable answers, there may be errors lurking in untried cases."

p.8 and 57 and 92 change "third" to "next"
p.8 change "the third" to "that"
p.14 and 39 and 62 and 99 and 123 change "third" to "last"
p.45 change "third" to "next" and change "fifth and sixth" to "last two"
p.47 change "third" to "last" twice.
p.59 and 60 change "a third" to "another"
p.50 change "fourth" to "last"
p.176 change "third" to "remaining number"

---

p.192 Exercise 369 is reworded as follows.

369 Let p be a user's boolean variable, and let m be an implementer's natural variable. The operations allow the user to assign a value n to the implementer's variable, and to test whether the implementer's variable is a prime number.

$$assign\ n \ = \ m := n$$
$$check \ = \ p := prime\ m$$

assuming *prime* is suitably defined. If *prime* is an expensive function, and the *check* operation is more frequent than the *assign* operation, we can improve the solution by making *check* less expensive even if that makes *assign* more expensive. Using data transformation, make this improvement.

---

p.67, last two lines before end of variable suspension are changed to:
The definitions of *ok* and assignment using state variables

$$ok \ = \ x'=x \wedge y'=y \wedge \ldots$$
$$x := e \ = \ x'=e \wedge y'=y \wedge \ldots$$

were partly informal, using three dots to say "and other conjuncts for other state variables". If we had defined **frame** first, we could have defined them formally as follows:

$ok \quad = \quad \mathbf{frame} \cdot \top$

$x := e \quad = \quad \mathbf{frame}\ x \cdot \ x' = e$

---

p.3, after the sentence ending with " "power" and "ground"." put the sentences:
Similarly we may choose words from other application areas. Or, to be independent of application, we may call them "top" and "bottom".

---

p.1 replace:
To use model-checking ... are not automatic.
with:
To use model-checking on any program with more than six variables requires abstraction, and each abstraction requires proof that it preserves the properties of interest. These abstractions and proofs are not automatic.

---

p.30 change "It is also the bunch of all functions with domain *nat* and result in *nat* ." to "It is also the bunch of all functions whose domain includes *nat* and whose result is included in *nat* ."

---

p.30 add two hints to get

$suc: nat \to nat$	use Function Inclusion Law
$= \ nat: nat \ \wedge \ \forall n: nat \cdot \ suc\ n: nat$	reflexivity, and apply *suc* to *n*
$= \ \forall n: nat \cdot \ n+1: nat$	*nat* construction axiom
$= \ \top$	

---

p.30, move the "End of Function Inclusion and Equality" line back one paragraph.

---

p.32 change "*f*0 *f*1 *f*2; ... " to "*f*0; *f*1; *f*2; ... "

---

p.33 change "*p*0 *p*1 *p*2 ... " to "*p*0; *p*1; *p*2; ... "

---

p.204 middle: change the sentence beginning "But its benefit ... " to "But its benefit is not worth its trouble, since the same check is made at every dependent composition. Even worse, we would lose the Substitution Law; we want $(n := -1. \ n \geq 0)$ to be $\bot$.".

---

p.35 top: change first two definitions to:

Specification *S* is <u>unsatisfiable</u> for prestate σ : $\quad \not\in (\S \sigma' \cdot S) < 1$

Specification *S* is <u>satisfiable</u> for prestate σ : $\quad \not\in (\S \sigma' \cdot S) \geq 1$

---

p.87 at the very end, just after "item we seek", add "And in the fast exponentiation problem, it is better on average to test *even y* rather than $y=0$ if we have a choice."

---

p.58, last paragraph "We used Refinement by Parts ... not always a successful strategy." is replaced by the following paragraph:
The timing can be written as a conjunction

$(y=0 \Rightarrow t'=t) \ \wedge \ (y>0 \Rightarrow t' \leq t + log\ y)$

and it is tempting to try to prove those two parts separately. Unfortunately we cannot prove the second part of the timing by itself. Separating a specification into parts is not always a successful strategy.

---

p.59 replace function f by function fib everywhere.

---

p.173 Ex.217: replace function f by function fib everywhere.

---

p.183 Ex.301 replace function f by function fib everywhere.

---

p.71 middle, change "For example," to "For example, if L is an implementable specification, then".

---

p.74, replace the two paragraphs starting "As with the previous" and "For example" with

As with the previous loop constructs, we will not define the **for**-loop as a specification, but instead show how it is used in refinement. Let F be a function of two integer variables whose result is an implementable specification. Then

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

is an abbreviation of the three refinements

$Fii \Leftarrow m \le i \le n \wedge ok$

$Fi(i+1) \Leftarrow m \le i < n \wedge P$

$Fik \Leftarrow m \le i < j < k \le n \wedge (Fij.Fjk)$

If $m=n$ there are no iterations, and specification Fmn must be satisfied by doing nothing ok. The body of the loop has to do one iteration $Fi(i+1)$. Finally, Fmn must be satisfied by first doing the iterations from m to an intermediate index j, and then doing the rest of the iterations from j to n.

For example, let the state consist of integer variable x, and let F be defined as

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

Then we can solve the exponentiation problem $x' = 2^n$ in two refinements:

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

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

The first refinement is proven by the Substitution Law. To prove the second, we must prove three theorems

$Fii \Leftarrow 0 \le i \le n \wedge ok$

$Fi(i+1) \Leftarrow 0 \le i < n \wedge (x := 2 \times x)$

$Fik \Leftarrow 0 \le i < j < k \le n \wedge (Fij.Fjk)$

all of which are easy.

---

p.74 very bottom and p.75 top: starting with "We check both" change to

To prove

$F\ 0\ (\#L) \Leftarrow$ **for** $i := 0;..\#L$ **do** $L := i \rightarrow Li+1 \mid L$

we must prove three theorems:

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

$Fi(i+1) \Leftarrow 0 \le i < \#L \wedge (L := i \rightarrow Li+1 \mid L)$

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

2006-4-3 I am changing the word "nullary" to the word "one-operand", the word "binary" to the word "two-operand", and the word "ternary" to the word "three-operand" everywhere that is

appropriate. There are a couple of associated changes on page 3.

2006-4-4 p.147 added missing "End of Preface" line, and reworded Exercise 0 as follows:

0 There are four cards on a table showing symbols D, E, 2, and 3 (one per card). Each card has a letter on one side and a digit on the other. Which card(s) do you need to turn over to determine whether every card with a D on one side has a 3 on the other? Why?

2006-4-4 p.226 corrected spelling of "annihilation".

2006-4-10 **Chinese version done**

2006-4-12 p.91 replace the line "0: *nat* add 1 to each side" with the two lines

T by the axiom, 0: *nat*

$\Rightarrow$ 0: *nat* add 1 to each side

2006-4-28 p.15 the three dots beside *real* are replaced by "..., $2^{1/2}$, ..."

2006-4-28 p.16 replace "where x and y are extended integers" with "where x is an integer and y is an extended integer"

2006-4-28 p.16 final sentence "Since we have given the axiom defining the ,.. notation, it is formal, and can be used in proofs." is replaced by "The ,.. notation is formal. We have an axiom defining it, so we don't have to guess what is included."

2006-4-28 p..227 replace

i: *int* $\wedge$ x, y: *xint* $\wedge$ $x \le y$ $\Rightarrow$ $(i: x,..y = x \le i < y)$

x, y: *xint* $\wedge$ $x \le y$ $\Rightarrow$ $\not\in(x,..y) = y - x$

with

x, i: *int* $\wedge$ y: *xint* $\wedge$ $x \le y$ $\Rightarrow$ $(i: x,..y = x \le i < y)$

x: *int* $\wedge$ y: *xint* $\wedge$ $x \le y$ $\Rightarrow$ $\not\in(x,..y) = y - x$

2006-4-30 p.29 middle just after the words "distributes over bunch union." add the sentence "The range of function f is $f(\Delta f)$."

2006-5-2 p.30 middle replace

f: $A \to B$ = A: Δf $\wedge$ $\forall a$: $A \cdot fa$: B

with

f: $A \to B$ = A: Δf $\wedge$ fA: B

2006-5-3 p.30

suc: $nat \to nat$ use Function Inclusion Law

= nat: Δsuc $\wedge$ $\forall n$: $nat \cdot suc\ n$: nat definition of *suc*

= nat: nat $\wedge$ $\forall n$: $nat \cdot n+1$: nat reflexivity, and *nat* construction axiom

= T

2006-5-3 p.31 The Polish notation paragraph now looks like this:

Suppose x, y: *int* and f, g: *int*→*int* and h: *int*→*int*→*int* . Then

	$h\ f\ x\ g\ y$	juxtaposition is left-associative
=	$(((h\ f)\ x)\ g)\ y$	use function composition on $h\ f$
=	$((h\ (f\ x))\ g)\ y$	use function composition on $(h\ (f\ x))\ g$
=	$(h\ (f\ x))\ (g\ y)$	drop superfluous parentheses
=	$h\ (f\ x)\ (g\ y)$	

2006-5-3 p.31 The last part of the page now looks like this:
As in this example, the shortcut usually works, but beware: it can sometimes lead to inconsistencies. (The word "apposition" has been suggested as a contraction of "application" and "composition", and it perfectly describes the notation, too!)

Like application, composition distributes over bunch union.

$$f\ (g, h) = f\ g, f\ h$$
$$(f, g)\ h = f\ h, g\ h$$

2006-5-14 p.60 line 9 change

$$n{=}0 \Rightarrow T \Leftarrow x{:=}\ 0.\ \ y{:=}\ 1$$

to

$$T \Leftarrow \textbf{if}\ n{=}0\ \textbf{then}\ (x{:=}\ 0.\ \ y{:=}\ 1)\ \textbf{else if}\ \textit{even}\ n\ \textbf{then}\ \textit{even}\ n \wedge n{>}0 \Rightarrow T\ \textbf{else}\ \textit{odd}\ n \Rightarrow T$$

2006-5-20 p.67 delete the sentence "Next, we consider ... focus temporarily." and change subsection 5.0.1 to the following.

5.0.1 Variable Suspension

We may wish, temporarily, to narrow our focus to a part of the state space. If the part is x and y, we indicate this with the notation

frame x, y

It applies to what follows it, according to the precedence table on the final page of the book, just like **var** . The **frame** notation is the formal way of saying "and all other variables (even the ones we cannot say because they are covered by local declarations) are unchanged". This is similar to the "import" statement of some languages, though not identical. If the state variables not included in the frame are w and z, then

$$\textbf{frame}\ x, y \cdot P = P \wedge w'{=}w \wedge z'{=}z$$

Within P the state variables are x and y. It allows P to refer to w and z, but only as local constants (mathematical variables, not state variables; there is no w' and no z'). Time and space variables are implicitly assumed to be in all frames, even though they may not be listed explicitly.

The definitions of *ok* and assignment using state variables

$$ok = x'{=}x \wedge y'{=}y \wedge \ldots$$
$$x{:=}\ e = x'{=}e \wedge y'{=}y \wedge \ldots$$

were partly informal, using three dots to say "and other conjuncts for other state variables". If we had defined **frame** first, we could have defined them formally as follows:

$$ok = \textbf{frame} \cdot \top$$
$$x{:=}\ e = \textbf{frame}\ x \cdot\ x'{=}e$$

End of Variable Suspension

2006-5-20 p.118 change “If we ignore time, ” to “If we ignore time and space, ”.

2006-5-21 p.69 change the first sentence of subsection 5.2.0 to
The **while**-loop of several languages has a syntax similar to

while *b* **do** *P*

where *b* is boolean and *P* is a specification.

2006-5-22 p.74 to the end of "and the initial values of *m* and *n* control the iteration" add "(so the number of iterations is *n–m*)".

2006-5-30 p.94 replace the line

$0, B{+}1 : B \Rightarrow nat : B$ replace B with $0, nat{+}1$

with the two lines

$\top$ *nat* induction axiom

$=$ $0, B{+}1 : B \Rightarrow nat : B$ replace B with $0, nat{+}1$

2006-6-9 p.2 change “In the current edition of this book, there is new material on space bounds, and on probabilistic programming.” to “Since the first edition of this book, new material has been added on space bounds, and on probabilistic programming.”.

2006-6-11 p.107 after the sentence ending “where *balance* is a specification that helps in writing the axioms, but is not an addition to the theory, and does not need to be implemented.” add “To prove an implementation is correct, we must propose a definition for *balance* that uses the implementer's variables, but it doesn't have to be a program.”.

2006-6-11 p.108 last line change “earlier” to “previous”.

---

unknown date Subsection 11.0.1 is renamed Basic Theories. Subsections 11.0.2 and 11.0.3 have become one subsection titled Basic Data Structures. Subsections 11.0.4 through 11.0.10 have their number reduced by 1.

2006-6-12 p.115,116. Change variable *u* to *c* and variable *v* to *x* , except on the last two lines of page 116, where variable *u* becomes *x* .

2006-6-14 p.25 change the paragraph beginning “One final abbreviation ...” to the following.
Some people refer to any expression as a function of its variables. For example, they might write

$x{+}3$

and say it is a function of *x* . They omit the formal variable and domain introduction, supplying them informally. There are problems with this abbreviation. One problem is that there may be variables that don't appear in the expression. For example,

$\lambda x : int \cdot \lambda y : int \cdot x{+}3$

which introduces two variables, would have the same abbreviation as

$\lambda x : int \cdot x{+}3$

Another problem is that there is no precise indication of the scope of the variable(s). And another is that we do not know the order of the variable introductions, so we cannot apply such an abbreviated

function to arguments. We consider this abbreviation to be too much, and we will not use it. We point it out only because it is common terminology, and to show that the variables we introduced informally in earlier chapters are the same as the variables we introduce formally in functions.

2006-6-22 p. 134 replace proof with

if $\surd c$ **then** $c?$ **else** $(t := t+1.\ W)$ replace $\surd c$ and W

$=$ **if** $\mathbb{T}\ r + 1 \le t$ **then** $c?$ **else** $(t := t+1.\ t := max\ t\ (\mathbb{T}\ r + 1).\ c?)$

$=$ **if** $\mathbb{T}\ r + 1 \le t$ **then** $(t := t.\ c?)$ **else** $(t := max\ (t+1)\ (\mathbb{T}\ r + 1).\ c?)$

If $\mathbb{T}\ r + 1 \le t$, then $t = max\ t\ (\mathbb{T}\ r + 1)$.

If $\mathbb{T}\ r + 1 > t$ then $max\ (t+1)\ (\mathbb{T}\ r + 1) = \mathbb{T}\ r + 1 = max\ t\ (\mathsf{T}\ r + 1)$.

$=$ **if** $\mathbb{T}\ r + 1 \le t$ **then** $(t := max\ t\ (\mathbb{T}\ r + 1).\ c?)$ **else** $(t := max\ t\ (\mathbb{T}\ r + 1).\ c?)$

$=$ W

2006-6-24 p.77 change the 2 lines

where b is boolean, and define it as follows.

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

to the following 2 lines

where b is boolean, to mean something like "make b be true". We define it as follows.

ensure b $=$ **if** b **then** ok **else** b

2006-6-24 p.77 after the words " **assert** b where b is boolean" add "to mean something like "I believe b is true". If it comes at the beginning of a procedure or method, it may use the word **precondition** ; if it comes at the end, it may use the word **postcondition** ; if it comes at the start or end of a loop, it may use the word **invariant** ; these are all the same construct.". At the end of the paragraph add the sentence "But it's not free; it costs execution time".

In the index, add p.77 to the bold words above.

2206-6-25 p.78, 79 The paragraph on the bottom of p.78 beginning "When nonlocal variables seem" and continuing on p.79 up to

$x := (P$ **result** $e)$ becomes $(P.\ x := e)$

is replaced by the following.

The expression P **result** e can be implemented as follows. 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. In the implementation of some programming languages, the introduction of fresh local variables for this purpose is not done, so the evaluation of an expression may cause a state change. State changes resulting from the evaluation of an expression are called "side-effects". With side-effects, mathematical reasoning is not possible. For example, we cannot say $x+x = 2 \times x$, nor even $x=x$, since x might be $(y := y+1$ **result** $y)$, and each evaluation results in an integer that is 1 larger than the previous evaluation. Side effects are easily avoided; a programmer can introduce the necessary local variables if the language implementation fails to do so. Some programming languages forbid assignments to nonlocal variables within expressions, so the programmer is required to introduce the necessary local variables.

If a programming language allows side-effects, we have to get rid of them before using any theory. For example,

x:= (P **result** e) becomes (P. x:= e)

On. p.221 change the index entry for "side-effect" from p.79 to p.78.

2006-6-26 p.80 from "Our choice of refinement" change the rest of the subsection to the following.

Our choice of refinement does not alter our definition of P ; it is of no use when using P . The users don't need to know the implementation, and the implementer doesn't need to know the uses.

A procedure and argument can be translated to a local variable and initial value.

(λp: D· B) a = (**var** p: D := a· B) if B doesn't use p' or p:=

This translation suggests that a parameter is really just a local variable whose initial value will be supplied as an argument. In many popular programming languages, that is exactly the case. This is an unfortunate confusion of specification and implementation. The decision to create a parameter, and the choice of its domain, are part of a procedural specification, and are of interest to a user of the procedure. The decision to create a local variable, and the choice of its domain, are normally part of refinement, part of the process of implementation, and should not be of concern to a user of the procedure. When a parameter is assigned a value within a procedure body, it is acting as a local variable and no longer has any connection to its former role as parameter.

Another kind of parameter, usually called a reference parameter or **var** parameter, stands for a nonlocal variable to be supplied as argument. Here is an example, using λ^* to introduce a reference parameter.

($\lambda^* x$: *int*· a:= 3. b:= 4. x:= 5) a
= a:= 3. b:= 4. a:= 5
= a'=5 ∧ b'=4

Reference parameters can be used only when the body of the procedure is pure program, not using any other specification notations. For the above example, if we had written

($\lambda^* x$: *int*· a'=3 ∧ b'=4 ∧ x'=5) a

we could not just replace x with a , nor even x' with a' . Furthermore, we cannot do any reasoning about the procedure body until after the procedure has been applied to its arguments. The following example has a procedure body that is equivalent to the previous example,

($\lambda^* x$: *int*· x:= 5. b:= 4. a:= 3) a
= a:= 5. b:= 4. a:= 3
= a'=3 ∧ b'=4

but the result is different. Reference parameters prevent the use of specification, and they prevent any reasoning about the procedure by itself. We must apply our programming theory separately for each call. This contradicts the purpose of procedures.

2006-6-27 p.86 replace

	u:= (*rand* 6) + 1. v:= (*rand* 6) + 1.	replace *rand* and
	if u=v **then** t'=t **else** (t:= t+1. (t'≥t) × (5/6)$^{t'-t}$ × 1/6)	Substitution Law
=	((u': 1,..7) ∧ v'=v ∧ t'=t)/6. (u'=u ∧ (v': 1,..7) ∧ t'=t)/6.	replace first .
	if u=v **then** t'=t **else** (t'≥t+1) × (5/6)$^{t'-t-1}$ / 6	and simplify
=	((u': 1,..7) ∧ (v': 1,..7) ∧ t'=t)/36.	replace remaining .
	if u=v **then** t'=t **else** (t'≥t+1) × (5/6)$^{t'-t-1}$ / 6	and replace **if**

$=$ $\Sigma u'', v'' : 1,..7 \cdot \Sigma t'' \cdot$ $(t''=t)/36 \times ((u''=v'') \times (t'=t'')$
$+ \quad (u'' \neq v'') \times (t' \geq t''+1) \times (5/6)^{t'-t''-1} / 6)$ sum

$=$ $1/36 \times (6 \times (t'=t) \; + \; 30 \times (t' \geq t+1) \times (5/6)^{t'-t-1} / 6)$ combine

$=$ $(t' \geq t) \times (5/6)^{t'-t} \times 1/6$

with

$u := (rand\ 6) + 1. \quad v := (rand\ 6) + 1.$ replace *rand* and

if $u=v$ **then** $t'=t$ **else** $(t := t+1. \; (t' \geq t) \times (5/6)^{t'-t} \times 1/6)$ Substitution Law

$=$ $(u' : 1,..7 \; \wedge \; v'=v \; \wedge \; t'=t)/6. \; (u'=u \; \wedge \; v' : 1,..7 \; \wedge \; t'=t)/6.$ replace first .

if $u=v$ **then** $t'=t$ **else** $(t' \geq t+1) \times (5/6)^{t'-t-1} / 6$ and simplify

$=$ $(u', v' : 1,..7 \; \wedge \; t'=t)/36.$ replace remaining .

if $u=v$ **then** $t'=t$ **else** $(t' \geq t+1) \times (5/6)^{t'-t-1} / 6$ and replace **if**

$=$ $\Sigma u'', v'' : 1,..7 \cdot \Sigma t'' \cdot \; (t''=t)/36 \times (\; (u''=v'') \times (t'=t'')$
$+ (u'' \neq v'') \times (t' \geq t''+1) \times (5/6)^{t'-t''-1} / 6)$ sum

$=$ $(6 \times (t'=t) \; + \; 30 \times (t' \geq t+1) \times (5/6)^{t'-t-1} / 6) / 36$ combine

$=$ $(t' \geq t) \times (5/6)^{t'-t} \times 1/6$

2006-6-27 p.86 last line becomes
so on average it takes 5 additional throws of the dice (after the first) to get an equal pair.

2006-6-28 p.89 change 2 lines to

Functional specification S is unsatisfiable for domain element x : $\quad \notin Sx < 1$

Functional specification S is satisfiable for domain element x : $\quad \notin Sx \geq 1$

2006-6-28 p.90 the 3 occurrences of $\lambda i \cdot$ should be $\lambda i : nat \cdot$

2006-6-28 p.89 change the start of the final paragraph to
Functional refinement is similar to imperative refinement. An imperative specification is a boolean expression, and imperative refinement is reverse implication. Functional specification is a function, and functional refinement is the reverse of the function ordering. Functional specification P (the problem) is refined by functional specification S (the solution) if and only if $S : P$. To refine, we can either decrease the choice of result, or increase the domain. Now we have a most annoying problem.

2006-7-2 p.180 top line Exercise 274 is now named (call-by-name).

2006-7-11 p.182 Ex.294(a) gets a check mark to say it is done in the text.

2006-7-11 change p.111 to the following.

7.2.0 Security Switch

Exercise 367 is to design a security switch. It has three boolean user's variables a , b , and c . The users assign values to a and b as input to the switch. The switch's output is assigned to c . The output changes when both inputs have changed. More precisely, the output changes when both inputs differ from what they were the previous time the output changed. The idea is that one user might flip their input indicating a desire for the output to change, but the output does not change until the other user flips their input indicating agreement that the output should change. If the first

user changes back before the second user changes, the output does not change.

We can implement the switch with two boolean implementer's variables:

A records the state of input a at last output change

B records the state of input b at last output change

There are two operations:

$a{:=}\ \neg a.$ **if** $a{\neq}A \wedge b{\neq}B$ **then** $(c{:=}\ \neg c.\ A{:=}\ a.\ B{:=}\ b)$ **else** ok

$b{:=}\ \neg b.$ **if** $a{\neq}A \wedge b{\neq}B$ **then** $(c{:=}\ \neg c.\ A{:=}\ a.\ B{:=}\ b)$ **else** ok

In each operation, a user flips their input variable, and the switch checks if this input assignment makes both inputs differ from what they were at last output change; if so, the output is changed, and the current input values are recorded. This implementation is a direct formalization of the problem, but it can be simplified by data transformation.

We replace implementer's variables A and B by nothing according to the transformer

$A{=}B{=}c$

To check that this is a transformer, we check

$\exists A, B{\cdot}\ A{=}B{=}c$ generalization, using c for both A and B

$\Leftarrow\quad \top$

There are no new variables, so there was no universal quantification. The transformation does not affect the assignments to a and b, so we have only one transformation to make.

$\forall A, B{\cdot}\quad A{=}B{=}c$

$\Rightarrow\ \exists A', B'{\cdot}\quad A'{=}B'{=}c'$

$\wedge$ **if** $a{\neq}A \wedge b{\neq}B$ **then** $(c{:=}\ \neg c.\ A{:=}\ a.\ B{:=}\ b)$ **else** ok

expand assignments and ok

$=\quad \forall A, B{\cdot}\quad A{=}B{=}c$

$\Rightarrow\ \exists A', B'{\cdot}\quad A'{=}B'{=}c'$

$\wedge$ **if** $a{\neq}A \wedge b{\neq}B$ **then** $(a'{=}a \wedge b'{=}b \wedge c'{=}\neg c \wedge A'{=}a \wedge B'{=}b)$

else $(a'{=}a \wedge b'{=}b \wedge c'{=}c \wedge A'{=}A \wedge B'{=}B)$

one-point for A' and B'

$=\quad \forall A, B{\cdot}\ A{=}B{=}c \Rightarrow$ **if** $a{\neq}A \wedge b{\neq}B$ **then** $(a'{=}a \wedge b'{=}b \wedge c'{=}\neg c \wedge c'{=}a \wedge c'{=}b)$

else $(a'{=}a \wedge b'{=}b \wedge c'{=}c \wedge c'{=}A \wedge c'{=}B)$

one-point for A and B

$=\quad$ **if** $a{\neq}c \wedge b{\neq}c$ **then** $(a'{=}a \wedge b'{=}b \wedge c'{=}\neg c \wedge c'{=}a \wedge c'{=}b)$

else $(a'{=}a \wedge b'{=}b \wedge c'{=}c \wedge c'{=}c \wedge c'{=}c)$

use **if**-part as context to change **then**-part

$=\quad$ **if** $a{\neq}c \wedge b{\neq}c$ **then** $(a'{=}a \wedge b'{=}b \wedge c'{=}\neg c \wedge c'{=}\neg c \wedge c'{=}\neg c)$

else $(a'{=}a \wedge b'{=}b \wedge c'{=}c \wedge c'{=}c \wedge c'{=}c)$

$=\quad$ **if** $a{\neq}c \wedge b{\neq}c$ **then** $c{:=}\ \neg c$ **else** ok

$=\quad c{:=}\ (a{\neq}c \wedge b{\neq}c) \neq c$

Output c becomes the majority value of a, b, and c. (As a circuit, that's three "exclusive or" gates and one "and" gate.)

———————————————————————————————End of Security Switch

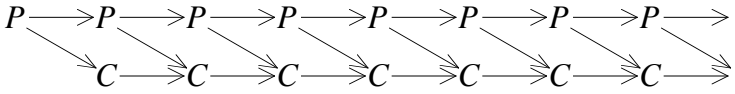
2006-7-11 p.191 In Ex.367 there are two places where "switch" should be changed to "input" to correspond to the previous change.

2006-7-22 p.116 change a paragraph to the following.
The transformed operation offered us the opportunity to rotate the queue within R, but we declined to do so. For other data structures, it is sometimes a good strategy to reorganize the data structure during an operation, and data transformation always tells us what reorganizations are possible. Each of the remaining transformations offers the same opportunity, but there is no reason to rotate the queue, and we decline each time.

2006-7-22 p.122 bottom change to

$produce = \cdots\cdots bw := e.\ w := w+1 \cdots\cdots$
$consume = \cdots\cdots x := br.\ r := r+1 \cdots\cdots$
$control = produce.\ consume.\ control$

then on page 123 change the first line to
If $w \neq r$ then *produce* and *consume* can be executed in parallel, as follows.
and change the picture to



and change the control equation to

$produce = \cdots\cdots bw := e.\ w := mod\ (w+1)\ n \cdots\cdots$
$consume = \cdots\cdots x := br.\ r := mod\ (r+1)\ n \cdots\cdots$
$control = produce.\ consume.\ control$

2006-7-22 p.18, 19, 20, 21, 155, 227, 228 the axioms for strings and lists now allow infinite strings and lists. There are some changes of words, and changes of axioms.

2006-7-23 Replace the old function notation $\lambda x{:}D{\cdot}b$ by the new notation $\langle x{:}D \rightarrow b \rangle$ everywhere. On p.23 after "within the body b." add the sentence "The brackets $\langle\ \rangle$ indicate the scope of the variable and axiom.". On p.25 change the sentence beginning "In this case, we also ." to "In this case, we also omit the scope brackets $\langle\ \rangle$.". On p.26 there are changes in the paragraph beginning "For the sake of convenience and tradition". On. p.36 change a sentence to "This is not the same as the raised dot used in the abbreviated form of quantification.". On p.201 at the end of the first paragraph on notation, add "In the first edition, I used λ notation for functions, thinking that it was standard. Ten years of students convinced me that it was not standard, freeing me to use a better notation in later editions.". Change sections 11.6, 11.7, 11.8 as appropriate. Sometimes parentheses can be deleted; for example $(\lambda x{:}D{\cdot}b)$ becomes just $\langle x{:}D \rightarrow b \rangle$.

2006-7-26 p.76 The entire Section 5.2.4 **Go To** has been replaced.

2006-7-28 p.85 after "as before." add "And **if** *rand* 2 **then** *A* **else** *B* can be replaced by **if** 1/2 **then** *A* **else** *B* .".

2006-7-178
258 Suppose variable declaration with initialization is defined as

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

In what way does this differ from the definition given in Subsection 5.0.0?

2006-7-31 On p.26 I have added a paragraph about a formal notation for substitution. This causes further tiny changes on pages 36, 37, 38, 84, 119, 126, 127, 225, 228, and 231.

2006-8-2 p.25
We may omit the domain of a function (and preceding colon) if ...
We may omit the variable (and following colon) when ...

2006-8-15 p.235 last section changes to the following:
The operators in the following expressions distribute over bunch union in any operand:

$[A]$ $A@B$ $A\,B$ $+A$ $-A$ $\$A$ $\leftrightarrow A$ $\#A$ $\sim A$ $\neg A$
A^B A_B $A\times B$ A/B $A\cap B$ $A+B$ $A-B$ $A+B$ $A\cup B$ $A;B$ $A\,'B$
$\neg A$ $A\wedge B$ $A\vee B$

The operator in A^*B distributes over bunch union in its left operand only.

2006-8-16 p.27 after $\Sigma n, m: 0,..10\cdot n\times m$ add:
These abbreviated quantifier notations make the scope of variables less clear, and they complicate the precedence rules, but the mathematical tradition is strong, and so we will use them.

2006-8-16 p.29 delete
A function whose body is a union is equal to a union of functions

$\langle x{:}D\rightarrow b,c\rangle = \langle x{:}D\rightarrow b\rangle, \langle x{:}D\rightarrow c\rangle$

2006-8-22 p.228 change second axiom of functional intersection to

$(f\,'\,g)\,x = (f\,|\,g)\,x\,'\,(g\,|\,f)\,x$

2006-9-14 p.228 in Axiom of Extension change both occurrences of v to w

---

2009-9-20 p.149 in Exercise 6, modify one sentence to say
The labels are separate from the variables; each label used in a **then**-part or **else**-part must be defined by one of the definitions; exactly one label must be defined but unused.

2006-10-3 p.176 change Exercise 245 to the following:

245 (longest common prefix) A natural number can be written as a sequence of decimal digits with a single leading zero. Given two natural numbers, write a program to find the number that is written as their longest common prefix of digits. For example, given 025621 and 02547 , the result is 025 . Hint: this question is about numbers, not about strings or lists.

2006-10-21 p.180 change Exercises 275 and 276 to the following.

275 We defined **wait until** w $=$ $t{:=}\,max\,t\,w$ where t is an extended integer time variable, and w is an integer expression.

(a)√ Prove **wait until** w $\Leftarrow$ **if** $t\geq w$ **then** ok **else** ($t{:=}\,t{+}1$. **wait until** w)

(b) Now suppose that t is an extended real time variable, and w is an extended real expression. Redefine **wait until** w appropriately, and refine it using the real time measure (assume any positive operation time you need).

276 The specification **wait** w where w is a length of time, not an instant of time, describes a

delay in execution of time w. Formalize and implement it using

(a) the recursive time measure.

(b) the real time measure (assume any positive operation times you need).

2006-10-23 p.99 change to

$t' \geq t \Leftarrow$ **while** b **do** P

if b **then** (P. $t := t+1$. **while** b **do** P) **else** $ok \Leftarrow$ **while** b **do** P

$\forall \sigma, \sigma' \cdot (t' \geq t \wedge$ (**if** b **then** (P. $t := t+1$. W) **else** $ok) \Leftarrow W$)

$\Rightarrow \forall \sigma, \sigma' \cdot$ (**while** b **do** $P \Leftarrow W$)

Recursive timing has been included, but this can be changed to any other timing policy.

2006-10-23 p.99 change to

The last axiom, induction, says that it is the weakest specification that satisfies the first two axioms.

2006-10-23 p.99 change to

while b **do** $P = t' \geq t \wedge$ (**if** b **then** (P. $t := t+1$. **while** b **do** P) **else** ok)

$\forall \sigma, \sigma' \cdot (W = t' \geq t \wedge$ (**if** b **then** (P. $t := t+1$. W) **else** ok))

$\Rightarrow \forall \sigma, \sigma' \cdot$ (**while** b **do** $P \Leftarrow W$)

2006-10-23 p.231 change to

while b **do** $P = t' \geq t \wedge$ (**if** b **then** (P. $t := t+1$. **while** b **do** P) **else** ok)

2006-10-23 p.186 change to

329 The notation **do** P **while** b has been used as a loop construct that is executed as follows. First P is executed; then b is evaluated, and if $\top$ execution is repeated, and if $\bot$ execution is finished. Define **do** P **while** b by construction and induction axioms.

2006-10-23 p.186 change to

330 Using the definition of Exercise 329, but ignoring time, prove

(a) **do** P **while** $b = P$. **while** b **do** P

(b) **while** b **do** $P =$ **if** b **then do** P **while** b **else** ok

(c) $(\forall \sigma, \sigma' \cdot (D =$ **do** P **while** $b)) \wedge (\forall \sigma, \sigma' \cdot (W =$ **while** b **do** $P))$

$= (\forall \sigma, \sigma' \cdot (D = P.\ W)) \wedge (\forall \sigma, \sigma' \cdot (W =$ **if** b **then** D **else** $ok))$

2006-11-9 p.118 change a paragraph to

If we are presented with an independent composition, and we are not told how the variables are partitioned, we have to determine a partitioning that makes sense. Here's a way that usually works: If either x' or $x :=$ appears in a process specification, then x belongs to that process. If neither x' nor $x :=$ appears at all, then x can be placed on either side of the partition. This way of partitioning does not work when x' or $x :=$ appears in both process specifications.

2006-11-13 p.231 after

var x: $T \cdot P = \exists x, x'$: $T \cdot P$

add

frame $x \cdot P = P \wedge y' = y \wedge \ldots$

2006-11-16 p.230 change one restriction, add another, as follows:
One-Point Laws — if element x: D
Solution Laws — if x is an element

2006-11-13 p.160 change
There is a Renaming Axiom
$\langle v \rightarrow b \rangle = \langle w \rightarrow$(substitute w for v in b)$\rangle$
and an Application Axiom
$\langle v \rightarrow b \rangle\, x =$ (substitute x for v in b)
to
There is an Application Axiom
$\langle v \rightarrow b \rangle\, x =$ (substitute x for v in b)
and an Extension Axiom
$f = \langle v \rightarrow fv \rangle$

2006-11-27 p.32 change "For most purposes" to "For some purposes" and change "can be regarded as a kind of list" to "can sometimes be regarded as a kind of list".

2006-11-29 I am replacing the powerset operator $\mathbb{2}$ with the power operator $\mathbb{4}$ everywhere in the book. It's not just the symbol that's different; it works differently. That's pages 17, 155, 227, 234, 235.

2006-12-4 p.-4 Section 3.4 now starts on p. 33. On p.30 Section 3.2.1 has bee rewritten. On p.209 an acknowledgement to Peter Kanareitsev is added. On p.218 limit is now p.33.

2007-1-16 p.93 near top an arrow is changed to a dot

2007-1-16 p.208 an r should be a 0 in
$= \exists \mathbb{M}, \mathbb{T}, r, r', w, w' \cdot t' = max\ t\ (\mathbb{T}0 + 1) \wedge r'=1 \wedge w'=0$

2007-1-16 p.139 one occurrence and p.140 two occurrences of T should be $\mathbb{T}$.

2007-3-7 p.77&78 change the bottom of p. 77 to
ensure b $=$ **if** b **then** ok **else** $b' \wedge ok$
$=$ $b' \wedge ok$
Like **assert** b, **ensure** b is equal to ok if b is true. But when b is false, there is a problem: the computation must make b true without changing anything. This is unimplementable ...
and on the top of p.78, second line, change a double prime to a single prime
$= \exists x'', y'' \cdot (x''=0 \wedge y''=y \vee x''=1 \wedge y''=y) \wedge x'=1 \wedge x'=x'' \wedge y'=y''$

2007-3-7 p.81,82 The square brackets around array indexes have been removed, and on p.82 the sentence
A procedure with reference parameters can be replaced by a function that returns a structured value (not shown).
is replaced by
Reference parameters are unnecessary if functions can return structured values.

2007-5-22 p.54 change "this level is best done with the aid of an automated theorem prover" to "to achieve this level, an automated theorem prover is very helpful".

2007-7-31 p.225 change second-last line to

$-(x/y) = -x \,/\, y$ semi-distributivity

2007-7-31 p.235 change "(and initial operand of **or**)" to "(and to both operands of **or**)".

2007-9-27 p.1 change "That is more than the estimated number of atoms in the universe!" to "That is something like the number of atoms in our galaxy!"

2007-10-22 p.17 change "all sets that contain elements of the bunch" to "all sets that contain only elements of the bunch".

2007-10-22 p.165 Ex.132 change "Can we prove the refinement" to "Let t be the time variable. Can we prove the refinement".

2007-10-22 p.183 top add quantifiers to Exercise 300 as follows:

300 Function f is called monotonic if $\forall i, j \cdot\ i \le j \Rightarrow f\,i \le f\,j$.

(a) Prove f is monotonic if and only if $\forall i, j \cdot\ f\,i < f\,j \Rightarrow i < j$.

(b) Let f: $int \to int$. Prove f is monotonic if and only if $\forall i \cdot\ f\,i \le f\,(i{+}1)$.

2007-10-24 p.223 the second law of exclusion changes from $a = \neg b \ =\ b = \neg a$ to

$a = \neg b \ =\ a \neq b \ =\ \neg a = b$

2007-10-24 p.227 change

$x, i{:}\ int \ \wedge\ y{:}\ xint \ \wedge\ x{\le}y \ \Rightarrow\ (i{:}\ x,..y \ =\ x{\le}i{<}y)$

to

$x{:}\ int \ \wedge\ y{:}\ xint \ \wedge\ x{\le}y \ \Rightarrow\ (i{:}\ x,..y \ =\ i{:}\ int \ \wedge\ x{\le}i{<}y)$

2007-10-25 p.32 change "or even this way:" to "It would be even nicer if we could write them this way:".

2007-11-15 p.209 last two lines: change "The backtracking implementation of unimplementable specifications comes from Greg Nelson (1989)." to "The backtracking implementation of unimplementable specifications is an adaptation of a technique due to Greg Nelson (1989) for implementing angelic nondeterminism."

2007-11-29 p.11 add a new context rule

In **if** *expression0* **then** *expression1* **else** *expression2* , when changing *expression0* ,

we can assume *expression1*≠*expression2* .

2007-11-29 p.206 add a paragraph

An alternative way to define variable declaration is

var $x{:}\ T \ =\ x'{:}\ T \ \wedge\ ok$

which starts the scope of x , and

end x $=$ ok

which ends the scope of x . In each of these programs, ok maintains the other variables. This kind of declaration does not require scopes to be nested; they can be overlapped.

2007-12-9 p.86 replace the two lines

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

for an appropriate definition of R .

with the one line

$u'{=}v' \Leftarrow u{:=}(rand\ 6)+1.\ v{:=}(rand\ 6)+1.$ **if** $u{=}v$ **then** ok **else** $(t{:=}t{+}1.\ u'{=}v')$

2008-4-16 p.153 change "double negation rule" to "double negation rule: empty for x"

2008-4-30 p.229 the last two laws on the page are renamed "Antidistributive Laws".

2008-4-30 p.223-233 change occurrences of "Axiom" to "Law".

2008-6-27 p.142 near the bottom, delete "before we read another,".

2008-8-1 p.194 Ex.378 add the word "state" in front of each occurrence of "variable" or "variables".

2008-8-8 due to a printing and pdf problem I have to get rid of "math" font and replace all symbols in that font. The only noticeable change is that the string length operator $\#$ is changed to $\leftrightarrow$ throughout the book. In Exercise 90 p. 160 the right-pointing triangle has become » and in Exercise 309 p.184 the bunch subtraction symbol has changed to \ and in Exercise 378 p.194 the semi-dependent composition symbol has changed to $|||$.

2008-8-9 p.112 replace sentences with "Although the exercise talks about a list, we see from the operations that the items are always distinct, their order is irrelevant, and there is no nesting structure; that suggests using a bunch variable. But we will need to quantify over this variable, so we need it to be an element. We therefore use a set variable $s \subseteq \{nat\}$ as our implementer's variable."

2008-9-22 p.151 Exercise 19 change "on Island X" to "on these islands".

2008-9-22 p.34 change "Although the memory contents may physically be a sequence of bits, we can consider it to be a list of any items;" to "Although the memory contents may physically be a string of bits, we can consider it to be a string of any items;", and in "[–2; 15; `A; 3.14]" and also in "[*int*; (0,..20); *char*; *rat*]" delete the square brackets. And change "σ 0 , σ 1 , σ 2" to "σ_0 , σ_1 , σ_2". On p.46 change "σ = [*t*; *x*; *y*; ...]" to "σ = *t*; *x*; *y*; ...".

2008-9-22 p.34 change "and in a later chapter we will consider communication during the course of a computation" to "and in Chapter 9 we will consider intermediate states and communication during the course of a computation".

2008-10-12 p.71 change "means" to "is an alternative notation for".

2008-11-21 p.119 change "In the next chapter we introduce interactive variables and communication channels between processes, but in this chapter our processes are not able to interact." to "In the next chapter we introduce interactive variables and communication channels between processes so they can see the intermediate values of each other's variables, but in this chapter processes are not able to interact.".

2008-11-25 p.7, 8 At the end of p.7 add a new paragraph: "A formal proof is a proof in which every step fits the form of the law given as hint. The advantage of making a proof formal is that each step can be checked by a computer, and its correctness is not a matter of opinion." And on p.8 change "From the first line of the proof ... The hint now is “Duality”" to

From the first line of the proof, we are told to use “Material Implication”, which is the first of the Laws of Inclusion, to obtain the second line of the proof. The first two lines together

$$a \wedge b \Rightarrow c \;=\; \neg(a \wedge b) \vee c$$

fit the form of the Law of Material Implication, which is

$$a \Rightarrow b \;=\; \neg a \vee b$$

because $a \wedge b$ in the proof fits where a is in the law, and c in the proof fits where b is in the law. The next hint is “Duality”

Also, in front of "Sometimes it is clear enough how to get from one line to the next without a hint" add "The proofs in this book are intended to be read by people, rather than by a computer.".

2008-12-17 p.164 change

121 Is the refinement

to

121 Let x be an integer variable. Is the refinement

2008-12-25 p.227 in **11.4.5 Strings** change $\leftrightarrow S; T)$ to $\leftrightarrow(S; T)$

and change one occurrence of

$$\leftrightarrow S{<}\infty \;\Rightarrow\; (i{<}j \;=\; S; i; T \;<\; S; j; U)$$

to

$$\leftrightarrow S{<}\infty \;\Rightarrow\; (i{=}j \;=\; S; i; T \;=\; S; j; T)$$

and add

$$S_{\{A\}} \;=\; \{S_A\}$$

2008-12-28 p.228 In **11.4.6 Lists** replace

$$[S]\, n \;=\; S_n$$

with

$$[S]\, T \;=\; S_T\,.$$

and add

$$S_{[T]} \;=\; [S_T]$$

and delete

let n be a natural number;

$$(L\, M)\, n \;=\; L\, (M\, n)$$

$$L\, null \;=\; null$$

$$L\, (A, B) \;=\; L\, A, L\, B$$

$L\ nil\ =\ nil$
$L\ (S; T)\ =\ L\ S; L\ T$
$L\ (M+N)\ =\ L\ M + L\ N$

2008-12-28 I changed text from a list of characters to a string of characters. The character `A is now identical to the one-character string "A" , so I got rid of ` everywhere. Pages 19, 20, and 21 have been rewritten quite a lot. So have pages 113 and 114. There are minor changes on pages 62, 133, 156, 168, 177, 188, 190, 197, 222, and 234.

2009-1-28 I added a new string operator ⧏⧐, resulting in changes to pages 18, 19, 21, 34, 36, 227, 228, 234, 235.

2009-1-29 p.155 Exercise 46 is slightly changed.

2009-1-29 p.202 the sentence "A formal metalanguage is helpful for the presentation and comparison of a variety of competing formalisms, and necessary for proving theorems about formalisms." is replaced with "A formal metalanguage may be considered helpful (though it is not necessary) for the presentation and comparison of a variety of competing formalisms, and for proving theorems about formalisms.".

2009-2-4 p.82, 83, 84, 85, and the last lines of p.86 I rewrote the section on Probabilistic Programming.

2009-5-6 p.6 bottom change "prove something" to "prove a boolean expression".

2009-5-24 p.107 to "so that *count* counts the number of pushes and pops." add "since the last use of *start* ".

2009-6-28 p.161 change Exercise 101 to

101 (rolling)

(a) Can we always unroll a loop? If $S \Leftarrow A.S.Z$, can we conclude $S \Leftarrow A.A.S.Z.Z$?

(b) Can we always roll up a loop? If $S \Leftarrow A.A.S.Z.Z$, can we conclude $S \Leftarrow A.S.Z$?

2009-6-30 p.162 change Exercise 109 to

109 Let x and y be real variables. Prove that if $y=x^2$ is true before

$x := x+1.\ y := y + 2\times x - 1$

is executed, then it is still true after.

2009-7-23 p.1 change "This theory is also more general than those just mentioned" to "This theory is also more comprehensive than those just mentioned".

2009-8-10 I have changed message and time scripts from lists to strings. It's a small conceptual change, but a large notational change, affecting much of Chapter 9, and part of Chapters 10 and 11.

2009-9-9 p.32 change

It is handy, and not harmful, to mix lists and functions in a composition. For example,

suc [3; 5; 2] = [4; 6; 3]

to

It is handy, and not harmful, to mix operators and lists and functions in a composition. For example,

– [3; 5; 2] = [–3; –5; –2]

suc [3; 5; 2] = [4; 6; 3]

2009-9-9 p.12 remove

$+x$ "plus x "

and on p.225 delete

$+x = x$ identity

and on p.235 delete prefix+ and replace infix+ by + and delete $+A$.

2009-9-11 p.156 Ex.57 becomes

Write an expression equivalent to each of the following without using § .

2009-9-21 p.63 after "Let m be the maximum space occupied before the start of execution" add "(remember that any program may be part of a larger program that started execution earlier)".

2009-10-1 p.187 change Exercise 336 to the following.

336 (widgets) A theory of widgets is presented in the form of some new syntax and some axioms. An implementation of widgets is written.

(a) How do we know whether the theory of widgets is consistent or inconsistent?

(b) How do we know whether the theory of widgets is complete or incomplete?

(c) How do we know whether the implementation of widgets is correct or incorrect?

2009-10-7 p.226 the law

$x\times(y/z) = (x\times y)/z$ associativity

is changed to the two laws

$x\times(y/z) = (x\times y)/z = x/(z/y)$ multiplication-division

$y\neq 0 \Rightarrow (x/y)/z = x/(y\times z)$ multiplication-division

---

2009-10-22 p.203 change "For example, = was at first just a syntax for the statement that two things are (in some way) the same, but now it is an operator with algebraic properties." to "For example, = was at first just a syntax for the informal statement that two things are (in some way) the same, but now it is a formal operator with algebraic properties.".

2009-11-2 p.166 change Exercise 144 to

144 (combinations) Write a program to find the number of ways to partition $a+b$ things into a things in the left part and b things in the right part. Include recursive time.

2009-11-2 p.206 change

$$s + \Sigma\, L\, [n;..\#L] = \Sigma L \ \wedge\ n \neq \#L \ \Rightarrow\ s' + \Sigma\, L\, [n';..\#L] = \Sigma L \ \wedge\ 0 \le \#L - n' < v$$

to

$$s + \Sigma\, L\, [n;..\#L] = \Sigma L \ \wedge\ n \neq \#L \ \Rightarrow\ s' + \Sigma\, L\, [n';..\#L] = \Sigma L \ \wedge\ 0 \le \#L - n' < \#L - n$$

2009-11-2 p.206 after
The proof method given in Chapter 5 is easier and more information (time) is obtained.
add
Sometimes the Method of Invariants and Variants requires the introduction of extra constants (mathematical variables) not required by the proof method in Chapter 5. For example, to add 1 to each item in list L requires introducing list constant M to stand for the initial value of L.

2009-11-25 p.94 After "This is exactly how we defined *null* in Chapter 2." add the sentence
The predicate form of *null* induction is

$$\forall x{:}\ null\cdot\ P\ x$$

2010 -1-11 p.99 at the end of Chapter 6 add
which was easily proved according to Chapter 5.

2010-3-2 p.208 I have added the paragraph
"If x is an interactive variable, $(t'{=}\infty.\ x{:=}\ 2.\ x{:=}\ 3)$ is unfortunately $\bot$; thus the theory of interactive variables is slightly too strong. Likewise, $(wc'{=}\infty.\ c!\ 2.\ c!\ 3)$ is unfortunately $\bot$; thus the theory of communication is slightly too strong. To eliminate these unwanted results we would have to weaken the definition of assignment to an interactive variable with the antecedent $t'{<}\infty$, and weaken output with the antecedent $wc'{<}\infty$. But I think these pathological cases are not worth complicating the theory."
and to make room for it I have removed the paragraph on page 207
"The combination of construction and induction is so useful that it has a name (generation) and a notation (::=). To keep terminology and notation to a minimum, I have not used them."
and rearranged a paragraph earlier on page 208 to
"Here are two execution patterns.

As we saw in Chapter 8, the first pattern can be expressed as $((A \parallel B).\ (C \parallel D))$ without any synchronization primitives. But the second pattern cannot be expressed using only parallel and sequential composition. This second pattern occurs in the buffer program."

2010-4-15 p.195 change Exercise 386 to
386 Let a and b be boolean interactive variables. Define

$$loop\ =\ \textbf{if}\ b\ \textbf{then}\ loop\ \textbf{else}\ ok$$

Add a time variable according to any reasonable measure, and then express

$$b{:=}\ \bot\ \parallel\ loop$$

as an equivalent program but without using $\parallel$.

2010-4-19 p.7 add the sentence "What is clear to one person may not be clear to another, so a proof is written for an intended reader." and change the word "correctness" to "validity".

2010-7-7 p.67 the end of Subsection 5.0.1 is moved one paragraph earlier.

2010-7-15 p.28 add

Once again, for tradition and convenience, when the solution quantifier is used within a set, we can abbreviate by omitting the quantifier. For example, instead of writing $\{\S n\colon nat\cdot n<3\}$, we might write $\{n\colon nat\cdot n<3\}$, which is a standard notation for sets.

2010-8-29 p.200 Exercise 418 becomes

418 (mutual exclusion) Process P is an endless repetition of a "non-critical section" PN and a "critical section" PC. Process Q is similar.

$$P = PN.\ PC.\ P$$
$$Q = QN.\ QC.\ Q$$

They are executed in parallel $(P \parallel Q)$. Specify formally that the two critical sections are never executed at the same time

(a) by inserting variables that are assigned but never used.

(b) by inserting outputs on channels that are never read.

2010-8-30 p.159 tiny rewording:

81 (Russell's barber) Bertrand Russell stated: "In a small town there is a male barber who shaves the men in the town who do not shave themselves.". Then Russell asked: "Does the barber shave himself?". If we say yes, then we can conclude from the statement that he does not, and if we say no, then we can conclude from the statement that he does. Formalize this paradox, and thus explain it.

2010-9-21 p.206 add the paragraph

Subsection 5.2.0 says that $W \Leftarrow$ **while** b **do** P is an abbreviation, but it is a dangerously misleading one. It looks like W is being refined by a program involving only b and P; in fact, W is being refined by a program involving b, P, and W.

2010-9-23 p.42 replace paragraph

For the "implemented expressions" ...

with

For the "implemented expressions" referred to in (b) and (c), we take the expressions of Chapters 1 and 2: booleans, numbers, characters, bunches, sets, strings, and lists, with all their operators. We omit functions and quantifiers because they are harder to implement, but they are still welcome in specifications.

2010-9-23 p.9 replace paragraph

The implication operator is reflexive ...

with

The implication operator is reflexive $a\Rightarrow a$, antisymmetric $(a\Rightarrow b) \wedge (b\Rightarrow a) = (a=b)$, and transitive $(a\Rightarrow b) \wedge (b\Rightarrow c) \Rightarrow (a\Rightarrow c)$. It is therefore an ordering (just like $\leq$ for numbers). We pronounce $a\Rightarrow b$ either as "a implies b", or, to emphasize the ordering, as "a is <u>stronger</u> than or equal to b". Likewise $a\Leftarrow b$ is pronounced either as "a is implied by b", or as "a is <u>weaker</u> than or equal to b". The words "stronger" and "weaker" may have come from a philosophical origin; we ignore any meaning they may have other than the boolean order, in which $\bot$ is stronger than $\top$.

2010-10-6 p.244 added

Laws of Equality and Difference

$a{=}b \;=\; (a \wedge b) \vee (\neg a \wedge \neg b)$

$a{\neq}b \;=\; (a \wedge \neg b) \vee (\neg a \wedge b)$

2010-10-24 p.145 replace a paragraph:

One problem with testing is: how do you know if the outputs are right? For some programs, such as graphics programs for producing pretty pictures, the only way to know if the output is right is to test the program and judge the result. But in other cases, a program may give answers you do not already know (that may be why you wrote the program), and testing it does not tell you if it is right. In such cases, you should test to see at least if the answers are reasonable.

2010-10-27 p.85 change the line

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

into two lines

$=$ $\quad (\Sigma r\colon 0,..2\cdot \Sigma s\colon 0,..3\cdot (x{:=}\, r.\ x{:=}\, x+s))/6$ replace final assignment, Substitution Law

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

---

2010-11-17 p.118 change

If we are presented with an independent composition, and we are not told how the variables are partitioned, we have to determine a partitioning that makes sense.

to

The person who introduces the independent composition is responsible for deciding how to partition the variables. If we are presented with an independent composition, and the person who wrote it failed to record the partitioning, we have to determine a partitioning that makes sense.

2010-12-17 p.224 added two more One Case Laws

if a **then** $\top$ **else** $b \;=\; a \vee b$

if a **then** $\bot$ **else** $b \;=\; \neg a \wedge b$

2011-3-30 p.6 a sentence near the bottom has been changed to "Instead of saying that *expression* is an antiaxiom or antitheorem, we can say that $\neg$*expression* is an axiom or theorem." and on p.7 the first sentence becomes the two sentences "We now replace the boolean antiaxiom ($\bot$) with an axiom ($\neg\bot$). With our two boolean axioms and five proof rules we can now prove theorems.".

2011-5-12 p.117 replace "But it is possible to find two programs that behave identically from a user's view, but for which there is no data transformer to transform one into the other." with "But it is possible to find two specifications of identical behavior (from a user's point of view) for which there is no data transformer to transform one into the other.".

2011-8-5 p.41 in the first paragraph of subsection 4.0.4 add the words "executing the program" and the word "specified", and change "disk head may crash" to "computer component may break".

2011-9-27 p.229 delete the two Change of Variable Laws

$\Sigma r\colon fD\cdot n \;=\; \Sigma d\colon D\cdot \langle r\colon fD{\to}n\rangle\ (fd)$

$\Pi r\colon fD\cdot n \;=\; \Pi d\colon D\cdot \langle r\colon fD{\to}n\rangle\ (fd)$

because they require $¢D = ¢fD$ for finite $¢D$ and a 1-to-1 property for infinite $¢D$.

2011-11-17 p.230 change the Extreme Law to

$(MIN\ v\cdot\ n) \le n \le (MAX\ v\cdot\ n)$

2011-12-5 p.100 and 101, replace "Let s, t: *stack* and x, y: X; then" with "Let s and t be elements of *stack*, and let x and y be elements of X; then". On p.102 replace "Let s: *stack* and x: X; then" with "Let s be an element of *stack* and let x be an element of X; then". On p.103 replace "Let q, r: *queue* and x, y: X." with "Let q and r be elements of *queue*, and let x and y be elements of X.". On p.104 replace "Let t, u, v, w: *tree* and x, y: X." with "Let t, u, v, and w be elements of *tree*, and let x and y be elements of X.".

2012-4-12 p.209 replace the year 2004 with the year 2011 in the list "leading to the present work", and on p.212 replace the "Probabilistic Predicative Programming" reference with

E.C.R.Hehner: "a Probability Perspective", *Formal Aspects of Computing* volume 23, number 4, pages 391,..420, 2011

2012-7-21 p.78: change Section 5.5.0 to

5.5.0 Result Expression

Let P be a specification and e be an expression in unprimed variables. Then

P **result** e

is an expression of the initial state. It expresses the result that would be obtained by executing P and then evaluating e. The base of the natural logarithms can be expressed as follows.

var *term*, *sum*: *rat* := 1·
for *i*:= 1;..15 **do** (*term*:= *term*/*i*. *sum*:= *sum*+*term*)
result *sum*

The scope of local variables *term* and *sum* extends to the end of the **result** expression.

The **result** expression axiom is

$P.\ (P$ **result** $e)=e$

except that $(P$ **result** $e)$ is not subject to double-priming in dependent composition, nor to substitution when using the Substitution Law. For example,

	⊤	use the **result** axiom
=	$x:= x+1.\ (x:= x+1$ **result** $x)=x$	use the Substitution Law, leaving the **result** expression unchanged
=	$(x:= x+1$ **result** $x) = x+1$	

The result is as if $x:= x+1$ were executed, then x is the result, except that the value of variable x is unchanged.

	$y:= (x:= x+1$ **result** $x)$	by the previous calculation
=	$y:= x+1$	

The expression P **result** e can be implemented as follows. 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 the modified P and evaluate the modified e.

In the implementation of some programming languages, the introduction of fresh local variables for this purpose is not done, so the evaluation of an expression may cause a state change. State changes resulting from the evaluation of an expression are called "side-effects". With side-effects, mathematical reasoning is not possible. For example, we cannot say $x+x = 2\times x$, nor even $x=x$, since x might be $(y:= y+1$ **result** $y)$, and each evaluation results in an integer that is 1 larger than the previous evaluation. Side effects are easily avoided; a programmer can introduce the necessary local variables if the language implementation fails to do so. Some programming languages forbid assignments to nonlocal variables within expressions, so the programmer is required to introduce the necessary local variables. If a programming language allows side-effects, we have to get rid of them before using any theory. For example,

$x:= (P$ **result** $e)$ becomes $(P.\ x:= e)$

after renaming local variables within P as necessary to avoid clashes with nonlocal variables, and allowing the scope of variables declared in P to extend through $x:= e$. For another example,

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

with similar provisos.

The recursive time measure that we have been using neglects the time for expression evaluation. This is reasonable in some applications for expressions consisting of a few operations implemented in computer hardware. For expressions using operations not implemented in hardware (perhaps list catenation) it is questionable. For **result** expressions containing loops, it is unreasonable. But allowing a **result** expression to increase a time variable would be a side-effect, so here is what we do. We first include time in the **result** expression for the purpose of calculating a time bound. Then we remove the time variable from the **result** expression (to get rid of the side-effect) and we put a time increment in the program surrounding the **result** expression.

End of Result Expression

2012-7-21 p.231: change the **result** axiom to

$P.\ (P$ **result** $e)=e$ but do not double-prime or substitute in $(P$ **result** $e)$

2012-7-21 p.179 change Exercise 271 to

271 Could we define the programmed expression P **result** e with the axiom

(a) $x'=(P$ **result** $e)\ =\ P.\ x'=e$

(b) $x'=(P$ **result** $e)\ \Longrightarrow\ P.\ x'=e$

(c) $P\ \Rightarrow\ (P$ **result** $e)=e'$

(d) $x'=(P$ **result** $e)\ \wedge\ P\ \Rightarrow\ x'=e'$

2012-12-4 p.42 add the sentence

We allow recursion because we know how to implement recursion.

and change the following sentence to

A computer executes $x\geq 0\ \Rightarrow\ x'=0$ by behaving according to the solution, and whenever the problem is encountered again, the behavior is again according to the solution.

2013-5-16 p.122 change

Or, we could do this capture at source level by splitting b into two variables, p and c , as follows.

produce = ·····$p:= e$·····

consume = ·····$x:= c$·····

$control = produce.\ newcontrol$
$newcontrol = c{:=}\,p.\ (consume \parallel produce).\ newcontrol$

Using B for the assignment $c{:=}\,p$, execution is
to
Or, we could do this capture at source level, using variable c, as follows.

$produce = \cdots\cdots b{:=}\,e\cdots\cdots$
$consume = \cdots\cdots x{:=}\,c\cdots\cdots$
$control = produce.\ newcontrol$
$newcontrol = c{:=}\,b.\ (consume \parallel produce).\ newcontrol$

Using B for the assignment $c{:=}\,b$, execution is

2014-1-19 p.203 at the end of the paragraph beginning "The subject" add
The problem was already in mathematics before programming. For example, 1 has two square roots, namely 1 and –1, neither of which deserves to be called "the principle square root". It is preferable to say $1^{1/2} = 1, -1$. Then we can say $(1^{1/2})^2 = (1, -1)^2 = 1^2, (-1)^2 = 1, 1 = 1$.

2014-7-26 p.207 change the sentence "They have been thought to be different for the following reasons: imperative programmers adhere to clumsy loop notations, complicating proofs; functional programmers adhere to equality, rather than refinement, making nondeterminism difficult." to "They have been thought to be different for the following reasons: imperative programmers adhere to clumsy loop notations rather than recursive refinement, complicating proofs; functional programmers adhere to equality rather than refinement, making nondeterminism difficult.".

2014-7-26 p.99 change the first line to "We replace n with ∞ (ignoring $t{=}{-}\infty$)".

2014-10-17 p.231add law

$P.$ **if** b **then** Q **else** R $=$ **if** $(P.\ b)$ **then** $(P.\ Q)$ **else** $(P.\ R)$ distributivity (unprimed b)

2014-10-18 p.64,65 change the increase in p from s to $s{\times}1$

2014-10-18 move the End of Average Space line up to just after "But we leave that as Exercise 212(f).".

2014-10-18 p.223 add

Mirror Law

$$a \Leftarrow b = b \Rightarrow a$$

2014-10-18 p.224 move Case Idempotent Law and Case Reversal Law from Boolean Laws to Generic Laws

---

2014-12-3 Section 8.1 replace the word "implementation" by the word "compiler"

2015-8-15 An extensive but shallow revision in which **if** is closed with **fi** and **do** is closed with **od**. And all texts like "abc" have left and right quotes "abc". And "boolean" is changed to "binary" and *bool* to *bin* throughout the book.

2015-9-18 The domain operator changed from Δ to $\square$.

2015-10-9 About 75 new exercises were added, about 12 old exercises were deleted, and this added 8 pages to the book. All the exercises are renumbered.

2015-10-9 The axioms for string order were correct in the original 1993 version of the book. But on 2006-7-22 strings were generalized to allow strings of infinite length, and the new axiom

$$\leftrightarrow S{<}\infty \;\Rightarrow\; (i{<}j \;=\; S;i;T \;<\; S;j;U)$$

was too strong. It is now weakened to

$$\leftrightarrow S{<}\infty \;\Rightarrow\; (i{<}j \;\Rightarrow\; S;i;T \;<\; S;j;U)$$

2015-11-11 The precedence of superscripting and subscripting changed from level 2 to level 0.

2015-11-23 On pages 120, 239, 241 the law

$$P \| ok \;=\; P \;=\; ok \| P$$

was changed to

$$P \,\|\, t'{=}t \;=\; P \;=\; t'{=}t \,\|\, P$$

The law was correct for the original 1993-2002 definition of $\|$. It should have been changed in 2002 when $\|$ was redefined.

2016-2-9 On pages 120 and 121 timing is added to *findmax*.

2016-3-12 pages 1 and 147: "Solutions to exercises are at
www.cs.utoronto.ca/~hehner/solutions.html ."

2016-3-12 page 211 I deleted a few sentences, and added the paragraph:
Let x be real, and let p be positive real. Then $0 < 0;x < p$. Hence $0;x$ is an infinitessimal, larger than 0 but smaller than any positive real. And $0 < 0;0;x < 0;p < p$, so $0;0;x$ is an infinitessimal smaller than $0;p$. And so on. Similarly $x < \infty < \infty;x < \infty;\infty < \infty;\infty;x$ and so on. But this book is not about infinitessimals and infinities.

2016-4-12 page 186 Ex.319(a) A small $=$ should be a large $\;=\;$.

2016-4-12 page 171 Ex.189 Delete n and replace " $log\ n$ where n is the length of the list." with " $log(\#L)$.".

2016-4-12 page 171 Ex.191 Replace first "list" with "list of natural numbers".

2016-4-12 page 172 Ex.201 Replace "at most $n \times log\ n$ " with " n ".

2016-4-14 pages 117 and 199 Ex.422 has been reworded for better clarity.

2016-5-3 image on front cover changed to inukshuk

2016-5-3 Chapter 0 is now Introduction, and the former Preface is now untitled.

2016-5-30 p.154 Ex.31 Change "There is a guard who is either a knight or a knave. Knights always tell the truth; knaves always lie." to "There is a guard. If you ask the guard a question, the guard may tell the truth, or the guard may lie.".

2016-6-1 p.-5 add

The cover picture is an inukshuk, which is a human-like figure made of piled stones. Inukshuks are found throughout arctic Canada. They are built by the Inuit people, who use them to mean "You are on the right path.".

2016-6-1 p.158 change the second sentence of Ex.69 to "For example, [0; 2; 1] is, but [2; 0; 1] is not, a sublist of [0; 1; 2; 2; 1; 0] .", and to the end of Ex.72 add "That's $3\times3 = 9$ questions.".

2016-6-25 p.16 change to "This makes it easy to express the plural naturals $(nat{+}2)$, the even naturals $(nat{\times}2)$, the square naturals (nat^2), the natural powers of two (2^{nat}), and many other things. (The operators that distribute over bunch union are listed on the final page of this book.)".

2016-6-25 p.17 and 235 change $\{{\sim}S\} = S$ to $\{{\sim}A\} = A$.

2016-8-19 p.183 Ex.304 now has a part (b).

2016-8-19 The paragraphing of **else if** is changed throughout the book, except for one occurrence on page 52 where it wouldn't fit.

2016-9-21 p.24 Delete "The substitution must replace every occurrence of v with w .".

2016-10-8 In Exercise 54 change S to B twice. In Exercise 48 replace $\#$ with $\otimes$. On p.233 replace the distributivity law $-(x{-}y) = -x - -y$ with the identity law $x{-}0 = x$.

2016-12-12 p.6 replace "useless theory" with "theory with no applications".

2017-4-4 p.23 replace "For example, we might introduce the name pi and the axiom $3.14 < pi < 3.15$, but we do not mean that every number is between 3.14 and 3.15 ." with "For example, we might introduce the name pi , and some axioms, and prove $3.14 < pi < 3.15$, but we do not mean that every number is between 3.14 and 3.15 .".

2017-5-11 p.152 Ex.26 add part (h): "P, Q, and R each say: "The other two are knaves.". How many knaves are there?". To fit it on the page, change part (a) to "P says: "If I am a knight, I'll eat my hat.". Does P eat his hat?". From Ex.27 remove "also".

2017-6-6 p.173 in Exercise 213 change "without using any functions (div , mod , $floor$, $ceil$, ..." to "without using functions div , mod , $floor$, or $ceil$ ".

2017-6-14 p.28 delete last paragraph, and just after the 3 lines starting "MAX" at the top of the page, add the paragraph:

These definitions make MIN and MAX more applicable than is traditional; for example,

$$MIN\ n{:}\ nat\cdot\ 1/(n{+}1) = 0$$

even though 0 is never a result of the function $\langle n: nat \rightarrow 1/(n+1) \rangle$.

2017-11-14 p.98 just before Subsection 6.1.0 add

But if we want to define *zap* as solution (a), we can do so by adding a fixed-point induction axiom

$$\forall \sigma, \sigma' \cdot (Z = \textbf{if } x=0 \textbf{ then } y:=0 \textbf{ else } x:=x-1.\ t:=t+1.\ Z \textbf{ fi})$$
$$\Rightarrow \forall \sigma, \sigma' \cdot zap \Leftarrow Z$$

This induction axiom says: if any specification Z satisfies the construction axiom, then *zap* is weaker than or equal to Z. So *zap* is the weakest solution of the construction axiom.

Although we defined *zap* using fixed-point construction and induction, we could have defined it equivalently using ordinary construction and induction.

$$\textbf{if } x=0 \textbf{ then } y:=0 \textbf{ else } x:=x-1.\ t:=t+1.\ zap \textbf{ fi} \Leftarrow zap$$
$$\forall \sigma, \sigma' \cdot (\textbf{if } x=0 \textbf{ then } y:=0 \textbf{ else } x:=x-1.\ t:=t+1.\ Z \textbf{ fi} \Leftarrow Z)$$
$$\Rightarrow \forall \sigma, \sigma' \cdot zap \Leftarrow Z$$

2017-11-14 p.192 as a result of the previous change, Ex.369 changes.

369 Section 6.1 defines program *zap* by the fixed-point equation

$$zap = \textbf{if } x=0 \textbf{ then } y:=0 \textbf{ else } x:=x-1.\ t:=t+1.\ zap \textbf{ fi}$$

(a)√ What axiom is needed to make *zap* the weakest fixed-point?

(b) What axiom is needed to make *zap* the strongest fixed-point?

(c) Section 6.1 gives six solutions to this equation. Find more solutions. Hint: strange things can happen at time ∞ .

2017-11-14 p.152 Exercise 24 has changed to the following.

24 (caskets) There are two caskets; one is gold and one is silver. In one casket there is a million dollars, and in the other there is a lump of coal. On the gold casket there is an inscription: the money is not in here. On the silver casket there is an inscription: exactly one of these inscriptions is true. Each inscription is either true or false (not both). On the basis of the inscriptions, choose a casket.

2017-12-30 p.218 add the following two sentences to the end of the section:

The theory in this book has been applied to object oriented programming by Yannis Kassios in his PhD thesis (2006), and to lazy execution by Albert Lai in his PhD thesis (2012); lazy execution timing can also be found in my paper (2018). Albert's thesis also proved the soundness of the theory in this book.

2017-12-30 pages 220 and 221 add the following three references:

E.C.R.Hehner: "a Theory of Lazy Imperative Timing", Workshop on Refinement, Oxford U.K., 2018 July

I.T.Kassios: *a Theory of Object-Oriented Refinement*, PhD thesis, University of Toronto, 2006

A.Y.C.Lai: *Eager, Lazy, and Other Executions for Predicative Programming*, PhD thesis, University of Toronto, 2013

2018-4-12 On p.163 Ex.111 add "or "top" ". On p.179 Ex.275 remove "*t*" twice. On p.207

Ex.477 add the word "then" four times. On p.208 Ex.483 change "must" to "should".

2018-4-14 I am making a systematic change throughout the book. Time is now either extended natural or extended nonnegative real. By not allowing time to be negative, I eliminate the problem of having the start time be $-\infty$. By not allowing it to be rational, I eliminate the problem of having a limit of rationals that is not rational. On page 46, change to: "The number system we extend can be the naturals or the nonnegative reals, whichever we prefer." and change "*xreal*" to "extended nonnegative real". On p.48 change "*xint*" to "*xnat*". On p.76 change "extended integer" to "extended natural" and "*xint*" to "*xnat*". On p.98 remove "(ignoring $t=-\infty$)". On p.126 change "usually either the extended integers or the extended reals" to "either the extended naturals or the extended nonnegative reals". On p.127, 134, 135, 136, 139 change "integer" to "natural". On p.139 change "*xint*" to "*xnat*" twice, and on p.140 the same change four times. On p.140 remove "For start time $t>-\infty$" twice. On p.185 Ex.312 change "extended integer time" to "extended natural time", "an integer expression" to "an extended natural expression", "extended real time" to "extended nonnegative real time", and "extended real expression" to "extended nonnegative real expression". On p.204 Ex.448 change "integer" to "natural". On p.206 Ex.466, change "integer" to "natural", and "real" to "nonnegative real". On p.206 Ex.469 change "integer" to "natural". On. p.207 Ex.474 change "integer" to "extended natural".

2018-5-17 In place of the two symbols $\sim$ and $\backsim$, I am just using one symbol $\sim$ because the two symbols obey the same laws. So change $\backsim$ to $\sim$ on p.20 twice, p.21 twice, p.236 twice. On p.242 delete the line "$\backsim$ 20 contents of a list" and change the line "$\sim$ 17 contents of a set" to "$\sim$ 17,20 contents of a set or list". On p.243 delete $\backsim$ and $\backsim A$.

2018-6-1 on p. 238 change the Extreme Law to the Extreme Laws

$(MIN\ n{:}\ int{\cdot}\ n) = (MIN\ n{:}\ real{\cdot}\ n) = -\infty$

$(MAX\ n{:}\ int{\cdot}\ n) = (MAX\ n{:}\ real{\cdot}\ n) = \infty$

and on p.239 add the limit law

$(LIM\ n{\cdot}\ n) = \infty$

2018-6-22 p.233 add the generic laws

$x \le x$	reflexivity
$x \le y \wedge y \le z \Rightarrow x \le z$	transitivity

and on p.236 move the Cardinality Law to p.237, and delete the long Selective Union law because it's almost the same as another Law of Selective Union.

2018-6-22 p.1 add the sentence:
The complete log of changes to the book is at www.cs.utoronto.ca/~hehner/aPToP/changelog.pdf .

2018-6-29 p.13 bottom: delete the line

$= \quad \neg\bot$	Binary Axiom, or Truth Table for $\neg$

2018-6-29 p.13 change "without them we cannot even prove 5=5 ." to "without them we cannot even prove 5=5 or "a"="a" ; with them, we can.".

2018-6-29 **Chinese version updated**

2018-10-15 p.167 Ex.148 is replaced by

148 Let x be an integer variable. Let P be a specification refined as follows.

$$P \Leftarrow \textbf{if } x>0 \textbf{ then } x:= x-2.\ P \textbf{ else if } x<0 \textbf{ then } x:= x+1.\ P \textbf{ else } ok \textbf{ fi fi}$$

(a) Prove the refinement when $P = x'=0$.

(b) Add recursive time and find and prove an upper bound for the execution time.

2018-11-7 p.93 first proof: change a hint from “distribution” to “antidistribution”.

2018-11-7 p.99 replace the paragraph

Beginning our recursive construction with $t' \geq t$, we have constructed a stronger but still implementable fixed-point. In this example, if we begin our recursive construction with $\bot$ we obtain the strongest fixed-point, which is unimplementable.

with the paragraph

Beginning our recursive construction with $t' \geq t$, we have constructed a stronger but still implementable fixed-point. We can choose any specification to start with, and we may obtain different fixed-points. In this example, if we begin our recursive construction with $\bot$ we obtain the strongest fixed-point, which is unimplementable.

2018-12-6 p.80 change $\langle * \rangle$ to $\langle \textbf{var} \rangle$ and on page 142 change $\langle ! \rangle$ to $\langle \textbf{chan} \rangle$.
