had $P$ expressing the correct working of some function $f$, and it could be put into the form

$$
P(x) \equiv \operatorname{pre}(x) \rightarrow \operatorname{post}(x, f(x))
$$

where pre and post together give the specification. $v$ is now the recursion variant, and the 'principle of circular reasoning' comes out (after incorporating some $\forall \mathcal{I}$ ) in Figure A.5.

| $a: A$ | $\forall y: A .(\operatorname{pre}(y) \wedge v(y)<v(a) \rightarrow \operatorname{post}(y, f(y)))$ |  |
| :--- | :--- | :--- |
|  | $\operatorname{pre}(a)$ |  |
|  | $\vdots$ |  |
|  | $\operatorname{post}(a, f(a))$ |  |
|  | $\operatorname{pre}(a) \rightarrow \operatorname{post}(a, f(a))$ | $\rightarrow \mathcal{I}$ |

$\forall x: A .(\operatorname{pre}(x) \rightarrow \operatorname{post}(x, f(x)))$
induction

Figure A. 5

## Lists

For lists xs, ys: [*], we can define a well-founded order easily enough by using the length, \# (for example, as a recursion variant):

$$
x s<y s \text { iff \# } x s<\# y s
$$

However, an interesting alternative is to define
$x s<y s$ iff $x s$ is the tail of $y s$
This gives the principle of list induction.

| $\vdots$ |  |
| :--- | :--- | :--- |
| $P([])$ |  |
|  | $h:{ }^{*}, t:\left[\begin{array}{ll}* & P(t) \\ \vdots \\ & \\ & \\ & \\ \hline\end{array}\right](h: t)$ |

Figure A. 6

Figure A. 6 contains an example of structural induction.

## Pairs and tuples

Theorem A. 3 Let $A$ and $B$ be two sets with well-founded orderings. We shall (naughtily) write the same symbol ' $<$ ' for both the orderings. Then $A \times B$ can be given a well-founded ordering by

$$
(a, b)<\left(a^{\prime}, b^{\prime}\right) \text { iff } a<a^{\prime} \vee\left(a=a^{\prime} \wedge b<b^{\prime}\right)
$$

Proof Suppose there is an infinite descending chain $\left(a_{1}, b_{1}\right)>\left(a_{2}, b_{2}\right)>$ $\left(a_{3}, b_{3}\right)>\ldots$. We have $a_{1}>a_{2}>a_{3}>\ldots$ and it follows from the well-foundedness of $a$ that the $a_{i}$ s take only finitely many values as they go down. Suppose $a_{n}$ is the last one, then eventually $a_{n}=a_{n+1}=a_{n+2}=\ldots$ and $b_{n}>b_{n+1}>b_{n+2}>\ldots$. But this is impossible by well-foundedness on $B$.

This can be extended to well-founded orderings on tuples, and it is really the same idea as lexicographic (alphabetical) ordering. BUT note that this depends critically on the fixed length of the tuples. For strings of arbitrary (though finite) length, lexicographic ordering is not well-founded. For example,

```
'taxis'> 'a1taxis'> 'aa1taxis'> 'aaa1taxis'> 'aaaa1taxis'> ...
```

There is a reasoning principle associated with the well-founded orderings on tuples (see Exercise 2), but perhaps the most common way to exploit the ordering is by choosing a recursion variant whose value is a tuple instead of a natural number.

## A. 1 Exercises

1. Another variant of the principle of course of values induction, shown in Figure A.2, is obtained by using a well-founded ordering on any subset of the natural numbers (for example, $<$ on the set of even natural numbers). Write down the proof obligations using proof boxes for such a variant.
2. Write down the proof obligations using proof boxes for a reasoning principle based on a well-founded ordering on tuples.

## Summary of equivalences

## Equivalent propositional forms:

zero law

$$
\begin{array}{lll}
\text { zero law } & P \rightarrow f f \equiv \neg P & \\
\text { complement laws } & P \wedge \neg P \equiv f f & P \vee \neg P \equiv \# \\
\text { idempotence } & P \wedge P \equiv P & P \vee Q \equiv Q \vee P \\
\text { commutativity } & P \wedge Q \equiv Q \wedge P & P \vee(Q \vee R) \equiv(P \vee Q) \vee R \\
\text { associativity } & P \wedge(Q \wedge R) \equiv(P \wedge Q) \wedge R & P \vee(P \vee Q) \equiv \neg P \wedge \neg Q \\
\text { De Morgan's laws } & \neg(P \wedge Q) \equiv \neg P \vee \neg Q & \neg(P) \\
\text { distributivity } & P \wedge(Q \vee R) \equiv(P \wedge Q) \vee(P \wedge R) \\
& R \rightarrow P \wedge Q \equiv(R \rightarrow P) \wedge(R \rightarrow Q) \\
& P \rightarrow(Q \rightarrow R) \equiv(P \wedge Q) \rightarrow R & \\
& P \vee(Q \wedge R) \equiv(P \vee Q) \wedge(P \vee R) \\
& (P \vee Q) \rightarrow R \equiv(P \rightarrow R) \wedge(Q \rightarrow R) \\
\text { others } & \neg(P \rightarrow Q) \equiv P \wedge \neg Q & \\
& \neg(P \leftrightarrow Q) \equiv(P \wedge \neg Q) \vee(\neg P \wedge Q) \\
& P \rightarrow Q \equiv \neg P \vee Q \equiv \neg(P \wedge \neg Q) \equiv \neg Q \rightarrow \neg P \\
& P \leftrightarrow Q \equiv(P \wedge Q) \vee(\neg P \wedge \neg Q) \equiv(P \rightarrow Q) \wedge(Q \rightarrow P)
\end{array}
$$

complement laws
idempotence
commutativity
associativity
distributivity
others

## Equivalent predicate forms:

$\forall x . \forall y . G(x, y) \equiv \forall y . \forall x . G(x, y)$
$\exists x . \exists y . \quad F(x, y) \equiv \exists y . \exists x . \quad F(x, y)$
$\neg \forall x . F(x) \equiv \exists x . \neg F(x)$
$\neg \exists x . F(x) \equiv \forall x . \neg F(x)$
$Q x .[S \wedge F(x)] \equiv S \wedge Q x . F(x) \quad\{Q$ can be $\forall$ or $\exists\}$
$Q x .[S \vee F(x)] \equiv S \vee Q x . F(x)$
$\forall x .[S \rightarrow F(x)] \equiv S \rightarrow \forall x . F(x)$
$\forall x .[F(x) \rightarrow S] \equiv \exists x . \quad F(x) \rightarrow S$
$\forall x .[F(x) \wedge G(x)] \equiv \forall x . F(x) \wedge \forall x . G(x)\{o r \equiv \forall u . F(u) \wedge \forall v . G(v)\}$
$\exists x .[F(x) \vee G(x)] \equiv \exists x . F(x) \vee \exists x . G(x)$

## Summary of natural deduction rules

$\wedge \mathcal{E}, \wedge \mathcal{I}, \vee \mathcal{E}$, and $\vee \mathcal{I}$ rules

- $\wedge \mathcal{E}$

$$
\frac{P_{1} \wedge \ldots \wedge P_{n}}{P_{i}(\wedge \mathcal{E})}
$$

for each of $P_{i}, i=1, \cdots, n$.

- $\wedge \mathcal{I}$

- $\vee \mathcal{E}$

- $\vee \mathcal{I}$

$$
\frac{P_{i}}{P_{1} \vee \ldots \vee P_{n}}(\vee \mathcal{I})
$$

for each of $P_{i}, i=1, \cdots, n$
$\rightarrow \mathcal{I}, \rightarrow \mathcal{E}, \neg \mathcal{I}, \neg \mathcal{E}$ and $\neg \neg$ rules

- $\rightarrow$ I

- $\rightarrow \mathcal{E}$

$$
\begin{array}{cr}
P & P \rightarrow Q \\
\hline Q & (\rightarrow \mathcal{E})
\end{array}
$$

- $\neg \mathcal{I}$

- $\neg \mathcal{E}$

$$
\begin{aligned}
& P \quad \neg P \\
& \hline \perp \quad(\neg \mathcal{E})
\end{aligned}
$$

- ᄀᄀ

$$
\frac{\neg \neg Q}{Q \quad(\neg \neg)}
$$

Equality rules

- eqsub

$$
\begin{array}{cc}
a=b & S[a] \\
\hline S[b] \quad(e q s u b)
\end{array}
$$

where $S[a]$ means a sentence $S$ with one or more occurrences of $a$ identified and $S[b]$ means those occurrences replaced by $b$.

- reflex

$$
a=a \quad(\text { reflex })
$$

## Universal quantifier rules

- $\forall \mathcal{E}$

$$
\frac{\forall x . P[x]}{P[t] \quad(\forall \mathcal{E})}
$$

where $t$ occurs in the current context.

- typed $\forall \mathcal{E}$

$$
\frac{i s \text {-type }(t) \quad \forall x: \text { type. } P[x]}{P[t] \quad(\forall \mathcal{E})}
$$

- $\forall \mathcal{I}$

| $c \forall \mathcal{I}$ |  |  |
| :--- | :--- | :--- |
|  | $\vdots$ |  |
|  | $P[c]$ |  |
|  | $\forall x . P[x] \quad(\forall \mathcal{I})$ |  |

where $c$ must be new to the current context.

- typed $\forall \mathcal{I}$

$$
\begin{array}{|lll|}
\hline c \forall \mathcal{I} & \text { is-t(c) } & \\
& \vdots & \\
& P[c] & \\
& \forall x: t . & P[x]
\end{array}(\forall \mathcal{I})
$$

- $\forall \rightarrow \mathcal{E}$ and $\forall \neg \mathcal{E}$

$$
\frac{\forall x \cdot[P[x] \rightarrow Q[x]]}{Q[c]}(\forall \rightarrow \mathcal{E}) \quad \text { and } \frac{\forall x . \neg P[x]}{\perp} \frac{P[c]}{(\forall \neg \mathcal{E})}
$$

## Existential quantifier rules

- $\exists \mathcal{I}$

$$
\frac{P[b]}{\exists x . P[x] \quad(\exists \mathcal{I})}
$$

where $b$ occurs in the current context.

- typed $\exists \mathcal{I}$

$$
\begin{array}{cr}
\text { is-type }(b) & P[b] \\
\hline \exists x: \text { type. } P[x] & (\exists \mathcal{I})
\end{array}
$$

- $\exists \mathcal{E}$

where $c$ is new to the current context.
- typed $\exists \mathcal{E}$

\[

\]

## Further reading

R.C. Backhouse. Program Construction and Verification. Prentice Hall, 1986. R. Bird and P. Wadler. Introduction to Functional Programming. Prentice Hall, 1988.
R. Bornat. Programming from First Principles. Prentice Hall, 1987.
O. Dah1. Verifiable Programming. Prentice Hall, 1992.
E. W. Dijkstra. A Discipline of Programming. Addison-Wesley, 1976.
E. W. Dijkstra and W.H.J. Feijen. A Method of Programming. Addison-Wesley, 1988.
S. Eisenbach and C. Sadler. Program Design with Modula-D. Addison-Wesley, 1989.
D. Gries. The Science of Programming. Springer Verlag, 1981.
C. Morgan. Programming from Specifications. Prentice Hall, 1990.
S. Reeve and M. Clarke. Logic for Computer Science. Addison-Wesley, 1990.
J. C. Reynolds. The Craft of Programming. Prentice Hall, 1981.
R. Smullyan. What is the Name of this Book? Prentice Hall, 1978.
V. Sperschneider and G. Antoniou. Logic: A Foundation for Computer Science. Addison-Wesley, 1991.
N. Wirth. Programming in Modula-2. Springer Verlag, 1982.

## Index

accumulating parameter, 191
actual parameter, 15
adjacency matrix, 177
aggregate type, 68
and, 9, 198
append, 69, 86
argument, 15, 214
arithmetic, 41
arity, 102, 200, 261
assertion, 143
associative, 69, 209
atom, 199
axiomatic approach, 81
base case, 53, 65, 84
bind, 204
black box, 15
bottom, 222
box proof, 84
built-in functions, 20, 41
characters, 41
Church-Rosser property, 54
circular reasoning, 58
code, 2
comparison operators, 40, 42
completeness, 260, 264, 271
complexity, 181
components, 91
composition, 19
compound types, 96
concatenate, 69
conclusion, 9
conjunction, 198, 206
connectives, 8
cons, 69, 70
consistency, 275
constant, 204
construct, 70
context, 273
contract, 27
contradiction, 209, 222
correct, 7, 214
course of values induction, 60
curried functions, 94
currying, 94
data structures, 68
data types, 40
decidable, 272
declaration, 18
deduction, 197
defensive specification, 29
defining functions, 46
defining values, 45
definition, 18, 21, 38
derived rules, 227
disjunction, 198
domain, 262, 279
double induction, 63

Dutch national flag algorithm, 164
edge, 176
elimination rules, 216
eqsub rule, 249
equality, 247
equation, 17, 47, 247
equivalent, 208
errors, 1
Euclid's algorithm, 56, 63
exclusive or, 11, 202
expression evaluation, 22
falsehood, 209
forall, 9
formal, 9
formal methods, 11
formal parameter, 17
formal parameters, 47, 138
formality, 10
formula, 216
function, 6, 15
function application, 15
functional composition, 18
functional language evaluator, 22
functional term, 200
generic, 99
global, 3
graph, 176
ground term, 238
guard, 48
head, 69, 86
higher-order function, 117
identifier, 44
implication, 9, 198
inconsistency, 275
induction hypothesis, 60, 84
induction step, 84
infinite lists, 73
infix, 50, 200
insertion sort, 76
instantiation, 54
interpretation, 261, 279
introduction rules, 216
invariant, 142
iteration, 186
layout, 47
lazy evaluation, 55, 65
length, 177
lists, 68
local, 3, 52
local definitions, 50
logic, 8
logic operators, 43
logical constants, 134
logical entailment, 260
logical implication, 260
logical notation, 8
loop invariant, 141, 144
loop test, 144
loop variant, 145
looping, 53, 186
map, 18
mapping diagram, 17
mathematical induction, 60
mathematical logic, 197
meaning, 25
mid-condition, 131, 143
model, 260, 264, 279
module, 6
Modus Ponens, 222
mutually exclusive, 48
node, 103,176
nullary constructor, 102
offside rule, 47,52
or, 198
partial application, 95
partition, 165
path, 176
pattern, 48, 49, 111
pattern matching, 48
patterns of recursion, 117

PC, 227
polymorphic type, 76
polymorphism, 97
post-condition, 28, 29
pre-condition, 28, 29
precedence, 24
predicates, 40, 199
prefix, 50
premiss, 9
preparation, 218
primitive functions, 20
primitive types, 96
Principle of course of values induction, 62
Principle of list induction, 84
Principle of mathematical induction, 60
procedure, 6
procedure call, 133
proof by contradiction, 227
propositional logic, 199
qualifier, 206
quality, 7
quantification, 204
quantifier, 204
reasoned program, 4
recurrence relationship, 53
recursion, 53, 186
recursion variant, 65
recursive, 53, 54
redex, 54
reduction strategy, 55
reflex rule, 249
relation, 176
relational operators, 42
reserved words, 44
result, 15, 139
rule, 17, 47
rule of substitution, 249
scheme, 227
semantics, 11
semi-decidable, 272
sentences, 199
sequent, 216
signature, 261, 279
simple induction, 60
simplification, 54
soundness, $260,264,271$
specification, 5, 20, 21, 27, 38
string, 71, 87
strong typing, 97
structural induction, 106
structure, 262, 279
substitution, 54
symmetry law of equality, 250
syntax analysis, 97
tail, 69, 86
tail recursion, 186
tautology, 209
terms, 199, 200
theorem, 227
theorem tactics, 230
top-down design, 20
transitive closure, 176, 177
truth table, 201, 211
tuple, 91, 111
type checking, 97
type variables, 98
typed quantifiers, 206
types, 28, 68
union types, 101
unit law, 69
universal quantifier, 204
user-defined constructors, 100
user-defined functions, 44
valid, $9,214,260$
values, 45
variable, 130, 204
variant, 143
weak completeness, 278
well-founded induction, 64, 282

