286 Well-founded induction

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

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

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.





Lists

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

xs < ys iff **#** xs < **#** ysHowever, an interesting alternative is to define xs < ys iff xs is the tail of ys

This gives the principle of list induction.

÷	h: *, t: [*]	P(t)
$P(\Box)$:
		P(h:t)

 $\forall xs : [*]. P(xs)$

induction

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) < (a', b') iff $a < a' \lor (a = a' \land b < b')$

Proof Suppose there is an infinite descending chain $(a_1, b_1) > (a_2, b_2) > (a_3, b_3) > \ldots$. 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. \Box

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,

 $`taxis' > `a1taxis' > `aa1taxis' > `aaa1taxis' > `aaaa1taxis' > \dots$

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	$P \to ff \equiv \neg P$			
complement laws	$P \land \neg P \equiv f\!\!f$	$P \lor \neg P \equiv tt$		
idempotence	$P \land P \equiv P$	$P \lor P \equiv P$		
commutativity	$P \land Q \equiv Q \land P$	$P \lor Q \equiv Q \lor P$		
associativity	$P \land (Q \land R) \equiv (P \land Q) \land R$	$P \lor (Q \lor R) \equiv (P \lor Q) \lor R$		
De Morgan's laws	$\neg (P \land Q) \equiv \neg P \lor \neg Q$	$\neg (P \lor Q) \equiv \neg P \land \neg Q$		
distributivity	$P \land (Q \lor R) \equiv (P \land Q) \lor (P \land$	R)		
	$R \to P \land Q \equiv (R \to P) \land (R$	$\rightarrow Q$)		
	$P \to (Q \to R) \equiv (P \land Q) \to R$			
	$P \lor (Q \land R) \equiv (P \lor Q) \land (P \lor$	R)		
	$(P \lor Q) \to R \equiv (P \to R) \land (Q$	$\rightarrow R$)		
others	$\neg (P \to Q) \equiv P \land \neg Q$			
	$\neg (P \leftrightarrow Q) \equiv (P \land \neg Q) \lor (\neg P \land$	$\land Q)$		
	$P \to Q \equiv \neg P \lor Q \equiv \neg (P \land \neg Q)$	$) \equiv \neg Q \to \neg P$		
	$P \leftrightarrow Q \equiv (P \land Q) \lor (\neg P \land \neg Q$	$) \equiv (P \to Q) \land (Q \to P)$		
Equivalent predicate forms:				
$\forall x. \ \forall y. \ G(x,y) \equiv \forall y. \ \forall x. \ G(x,y)$				
$\exists x. \ \exists y. \ F(x,y) \equiv \exists y. \ \exists x. \ F(x,y)$				
$\neg \forall x. \ F(x) \equiv \exists x. \ \neg F(x)$				
$\neg \exists x. \ F(x) \equiv \forall x. \ \neg F(x)$				
$Qx. [S \land F(x)] \equiv S \land Qx. F(x) \{Q \text{ can be } \forall \text{ or } \exists\}$				
$Qx. [S \lor F(x)] \equiv S \lor Qx. F(x)$				
$\forall x. \ [S \to F(x)] \equiv S \to \forall x. \ F(x)$				
$\forall x. \ [F(x) \to S] \equiv \exists x. \ F(x) \to S$				
$\forall x. \ [F(x) \land G(x)] \equiv \forall x. \ F(x) \land \forall x. \ G(x) \{ or \equiv \forall u. \ F(u) \land \forall v. \ G(v) \}$				
$\exists x. \ [F(x) \lor G(x)] \equiv \exists x. \ F(x) \lor \exists x. \ G(x)$				

Summary of natural deduction rules

 $\wedge \mathcal{E}, \ \wedge \mathcal{I}, \ \lor \mathcal{E}, \ \mathbf{and} \ \lor \mathcal{I} \ \mathbf{rules}$

•
$$\wedge \mathcal{E}$$

$$\frac{P_1 \wedge \ldots \wedge P_n}{P_i \quad (\wedge \mathcal{E})}$$

for each of P_i , $i = 1, \dots, n$.

• $\wedge \mathcal{I}$

$$\begin{array}{c|c}
\vdots \\
P_1 \\
\hline
P_1 \land \dots \land P_n \\
\hline
(\land \mathcal{I})
\end{array}$$

 $\bullet \ \forall \mathcal{E}$

$$P_1 \lor \ldots \lor P_n \qquad \begin{array}{c|c} P_1 & \ldots & P_n \\ \vdots & & \vdots \\ C & & C \\ \hline C & & (\lor \mathcal{E}) \end{array}$$

 $\bullet ~ \lor \mathcal{I}$

$$\frac{P_i}{P_1 \vee \ldots \vee P_n} \quad (\vee \mathcal{I})$$

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



Equality rules

• *eqsub*

$$\begin{array}{ccc} a = b & S[a] \\ \hline S[b] & (eqsub) \end{array}$$

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

 \bullet reflex

$$a = a$$
 (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 $\forall \mathcal{E}$

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

• $\forall \mathcal{I}$

$$\begin{array}{c} c \forall \mathcal{I} \\ \vdots \\ P[c] \\ \forall x. \ P[x] \qquad (\forall \mathcal{I}) \end{array}$$

where c must be new to the current context.

 $\bullet \ \mathrm{typed} \ \forall \mathcal{I}$

$$c \forall \mathcal{I} \quad is\text{-}t(c)$$

$$\vdots$$

$$P[c]$$

$$\forall x: t. \ P[x] \quad (\forall \mathcal{I})$$

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

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

Existential quantifier rules

• 3*I*

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

where b occurs in the current context.

• typed $\exists \mathcal{I}$

$$\frac{is\text{-}type(b)}{\exists x: type. \ P[x] \quad (\exists \mathcal{I})}$$

• ∃*E*



where c is new to the current context.

 $\bullet \ \mathrm{typed} \ \exists \mathcal{E}$

$$\exists x : t. \ P[x]$$

$$c \exists \mathcal{E} \quad P[c]$$

$$is - t(c)$$

$$\vdots$$

$$Q$$

$$Q \quad (\exists \mathcal{E})$$

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. Dahl. 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-2*. 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 lavout, 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