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.

$$\begin{array}{|l|}
\hline
a : A \quad \forall y : A.\ (pre(y) \wedge v(y) < v(a) \rightarrow post(y, f(y))) \\
\begin{array}{|l|}
\hline
\quad pre(a) \\
\quad \vdots \\
\quad post(a, f(a)) \\
\hline
\end{array} \\
\quad pre(a) \rightarrow post(a, f(a)) \qquad\qquad \rightarrow\mathcal{I} \\
\hline
\end{array}$$

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

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):

$xs < ys$ iff `#` $xs <$ `#` ys

However, an interesting alternative is to define

$xs < ys$ iff xs is the tail of ys

This gives the principle of list induction.

$$\begin{array}{|l|}
\hline
\vdots \\
P(\texttt{[]}) \\
\hline
\end{array}
\quad
\begin{array}{|l|}
\hline
h : *, t : \texttt{[*]} \quad P(t) \\
\quad \vdots \\
\quad P(h : t) \\
\hline
\end{array}$$

$$\forall xs : \texttt{[*]}.\ P(xs) \qquad\qquad \text{induction}$$

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

$$(a, b) < (a', b') \text{ iff } a < a' \vee (a = a' \wedge b < b')$$

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_is 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,

$$\text{'}taxis\text{'} > \text{'}a1taxis\text{'} > \text{'}aa1taxis\text{'} > \text{'}aaa1taxis\text{'} > \text{'}aaaa1taxis\text{'} > \ldots$$

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.

Appendix B

Summary of equivalences

Equivalent propositional forms:

zero law	$P \to \mathit{ff} \equiv \neg P$	
complement laws	$P \wedge \neg P \equiv \mathit{ff}$	$P \vee \neg P \equiv \mathit{tt}$
idempotence	$P \wedge P \equiv P$	$P \vee P \equiv P$
commutativity	$P \wedge Q \equiv Q \wedge P$	$P \vee Q \equiv Q \vee P$
associativity	$P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$	$P \vee (Q \vee R) \equiv (P \vee Q) \vee R$
De Morgan's laws	$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$	$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
distributivity	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$	
	$R \to P \wedge Q \equiv (R \to P) \wedge (R \to Q)$	
	$P \to (Q \to R) \equiv (P \wedge Q) \to R$	
	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$	
	$(P \vee Q) \to R \equiv (P \to R) \wedge (Q \to R)$	
others	$\neg(P \to Q) \equiv P \wedge \neg Q$	
	$\neg(P \leftrightarrow Q) \equiv (P \wedge \neg Q) \vee (\neg P \wedge Q)$	
	$P \to Q \equiv \neg P \vee Q \equiv \neg(P \wedge \neg Q) \equiv \neg Q \to \neg P$	
	$P \leftrightarrow Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q) \equiv (P \to Q) \wedge (Q \to P)$	

Equivalent predicate forms:

$\forall x.\ \forall y.\ G(x,y) \equiv \forall y.\ \forall x.\ G(x,y)$

$\exists x.\ \exists y.\ F(x,y) \equiv \exists y.\ \exists x.\ F(x,y)$

$\neg\forall x.\ F(x) \equiv \exists x.\ \neg F(x)$

$\neg\exists x.\ F(x) \equiv \forall x.\ \neg F(x)$

$Qx.\ [S \wedge F(x)] \equiv S \wedge Qx.\ F(x)$ $\{Q$ can be $\forall$ or $\exists\}$

$Qx.\ [S \vee F(x)] \equiv S \vee Qx.\ F(x)$

$\forall x.\ [S \to F(x)] \equiv S \to \forall x.\ F(x)$

$\forall x.\ [F(x) \to S] \equiv \exists x.\ F(x) \to S$

$\forall x.\ [F(x) \wedge G(x)] \equiv \forall x.\ F(x) \wedge \forall x.\ G(x) \{\mathit{or} \equiv \forall u.\ F(u) \wedge \forall v.\ G(v)\}$

$\exists x.\ [F(x) \vee G(x)] \equiv \exists x.\ F(x) \vee \exists x.\ G(x)$

Summary of natural deduction rules

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

- $\wedge\mathcal{E}$

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

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

- $\wedge\mathcal{I}$

  $$\begin{array}{c} \boxed{\begin{array}{c} \vdots \\ P_1 \end{array}} \;\cdots\; \boxed{\begin{array}{c} \vdots \\ P_n \end{array}} \\ P_1 \wedge \ldots \wedge P_n \qquad (\wedge\mathcal{I}) \end{array}$$

- $\vee\mathcal{E}$

  $$P_1 \vee \ldots \vee P_n \quad \begin{array}{|c|c|c|} \hline P_1 & \cdots & P_n \\ \vdots & & \vdots \\ C & & C \\ \hline \end{array}$$

  $$C \qquad (\vee\mathcal{E})$$

- $\vee\mathcal{I}$

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

  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

- $\rightarrow\mathcal{I}$

$$\begin{array}{|l|}\hline \quad P \\ \quad \vdots \\ \quad Q \\ \hline \end{array}$$
$$P \rightarrow Q \qquad (\rightarrow\mathcal{I})$$

- $\rightarrow\mathcal{E}$

$$\frac{P \qquad P \rightarrow Q}{Q} \ (\rightarrow\mathcal{E})$$

- $\neg\mathcal{I}$

$$\begin{array}{|l|}\hline \quad P \\ \quad \vdots \\ \quad \bot \\ \hline \end{array}$$
$$\neg P \qquad (\neg\mathcal{I})$$

- $\neg\mathcal{E}$

$$\frac{P \quad \neg P}{\bot} \ (\neg\mathcal{E})$$

- $\neg\neg$

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

Equality rules

- *eqsub*

$$\frac{a = b \quad S[a]}{S[b]} \ (eqsub)$$

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*

$$\frac{}{a = a} \ (reflex)$$

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 $\forall\mathcal{E}$

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

- $\forall\mathcal{I}$

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

  where c must be new to the current context.

- typed $\forall\mathcal{I}$

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

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

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

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 $\exists\mathcal{I}$

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

- $\exists\mathcal{E}$

$$\begin{array}{ll} & \exists x.\ P[x] \\ \hline \multicolumn{1}{|l}{c\exists\mathcal{E}} & \multicolumn{1}{l|}{P[c]} \\ \multicolumn{1}{|l}{} & \multicolumn{1}{l|}{\vdots} \\ \multicolumn{1}{|l}{} & \multicolumn{1}{l|}{Q} \\ \hline & Q \qquad (\exists\mathcal{E}) \end{array}$$

where c is new to the current context.

- typed $\exists\mathcal{E}$

$$\begin{array}{ll} & \exists x : t.\ P[x] \\ \hline \multicolumn{1}{|l}{c\exists\mathcal{E}} & \multicolumn{1}{l|}{P[c]} \\ \multicolumn{1}{|l}{} & \multicolumn{1}{l|}{\mathit{is\text{-}t}(c)} \\ \multicolumn{1}{|l}{} & \multicolumn{1}{l|}{\vdots} \\ \multicolumn{1}{|l}{} & \multicolumn{1}{l|}{Q} \\ \hline & Q \qquad (\exists\mathcal{E}) \end{array}$$

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