

A LOGICAL BASIS FOR PROGRAMMING METHODOLOGY II

SATORU TAKASU

RESEARCH INSTITUTE FOR MATHEMATICAL SCIENCES, KYOTO UNIVERSITY

INTRODUCTION: In the previous paper ([45]), we have developed a uniform method for putting proofs of the verification conditions of a while-program together to form a proof of its quantified specification. In this paper, we extend the method to PASCAL programs and briefly describe how to use the method for program synthesis incorporating it with a proof-checker system having Gentzen's sequent calculus as its underlying logic.

1. MAIN CONSTRUCTION: Asserted Pascal Statements and Their
Templets of Proofs

(0) We assume that a many sorted theory $T=(L, AxiS)$ is given.

Let $P(x,y)$, $Q(x,y)$ and $R(x,y)$ stand for formulas in L , and x and y stand for the vectors of input and output variables respectively, possibly occurring in those formulas. No other variables are free.

(1) An asserted assignment statement is of the form

$$\{P(x,y)\} y:=F(x,y) \{Q(x,y)\}$$

and its verification condition is $P(x,y) \supset Q(x, F(x,y))$, where F is a vector of function symbols in L . The templet for this statement is

$$\frac{P(a,b), \Gamma \rightarrow Q(a, F(a,b))}{P(a,b), \Gamma \rightarrow \exists y. Q(a,y)}$$

(2) An asserted if-statement is of the form

$$\{P(x,y)\} \text{ if } t(x,y) \text{ then } B_1 \text{ else } B_2 \{Q(x,y)\}$$

where $t(x,y)$ is a quantifier-free formula in L , and B_1 and B_2 are statements, and its verification conditions are those of

$$\{P(x,y) \wedge t(x,y)\} B_1 \{Q(x,y)\} \text{ and } \{P(x,y) \wedge \neg t(x,y)\} B_2 \{Q(x,y)\}.$$

The templet for this statement is

$$\frac{\begin{array}{cc} t(a,b), P(a,b), \Gamma & \neg t(a,b), P(a,b), \Gamma \\ \rightarrow \exists y. Q(a,y) & ; \rightarrow \exists y. Q(a,y) \end{array}}{\begin{array}{c} \rightarrow t(a,b) \vee \neg t(a,b) ; t(a,b) \vee \neg t(a,b), P(a,b), \rightarrow \exists y. Q(a,y) \\ \hline P(a,b), \Gamma \rightarrow \exists y. Q(a,y) \end{array}}$$

(3) An asserted while-statement is of the form

$$\{P(x,y)\} \text{ while } \{Q(x,y)\} t(x,y) \text{ do } B \{R(x,y)\}$$

and its verification conditions are the formulas

$$P(x,y) \supset Q(x,y) \quad , \quad Q(x,y) \wedge \neg t(x,y) \supset R(x,y)$$

and the verification conditions of the asserted statement

$$\{Q(x,y) \wedge t(x,y)\} B \{Q(x,y)\}.$$

We assume a variable k expressing the number of visits to the loop occurs in Q without loss of generality.

The templet for this consists of the synthesis part:

$$\begin{array}{c}
 \frac{t(a,n,d), Q(a,n,d), \Delta \quad \neg t(a,n,d), Q(a,n,d), \Pi}{\rightarrow \exists w.Q(a,n+1,w) \quad \rightarrow \exists y.R(a,y)} \\
 \frac{t(a,n,d), Q(a,n,d), \Delta \quad \neg t(a,n,d), Q(a,n,d), \Pi}{\rightarrow \exists w.Q(a,n+1,w) \quad \rightarrow \exists w.Q(a,n+1,w)} \\
 \frac{\quad \vee \exists y.R(a,y) \quad \vee \exists y.R(a,y)}{\quad} \\
 \frac{\rightarrow t(a,n,d) \vee \neg t(a,n,d); \quad t(a,n,d) \vee \neg t(a,n,d), Q(a,n,d), \Delta, \Pi \quad \exists y.R(a,y) \rightarrow \exists y.R(a,y)}{\rightarrow \exists w.Q(a,n+1,w) \vee \exists y.R(a,y) \quad \exists y.R(a,y) \rightarrow} \\
 \frac{P(a,b), \Gamma \quad Q(a,n,d), \Delta, \Pi \rightarrow \exists w.Q(a,n+1,w) \vee \exists y.R(a,y) \quad \exists w.Q(a,n+1,w)}{\rightarrow \exists w.Q(a,n,w) \quad \exists w.Q(a,n,w), \Delta, \Pi \rightarrow \exists w.Q(a,n+1,w) \vee \exists y.R(a,y); \exists y.R(a,y)} \\
 \frac{P(a,b) \rightarrow \quad \exists w.Q(a,n,w) \vee \exists y.R(a,y), \Delta, \Pi \rightarrow \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)}{\exists y.R(a,y)} \\
 \frac{P(a,b), \Gamma, \Delta, \Pi \rightarrow \exists w.Q(a,c,w) \vee \exists y.R(a,y)}{P(a,b), \Gamma, \Delta, \Pi \rightarrow \forall k.(\exists w.Q(a,k,w) \vee \exists y.R(a,y))}
 \end{array}$$

and combined with the following right part called the termination part:

$$\begin{array}{c}
 \frac{Q(a,K,e), \theta \rightarrow \exists y.R(a,y)}{\exists w.Q(a,K,w), \theta \rightarrow \exists y.R(a,y) \quad ; \exists y.R(a,y) \rightarrow \exists y.R(a,y)} \\
 \frac{\exists w.Q(a,K,w) \vee \exists y.R(a,y), \theta \rightarrow \exists y.R(a,y)}{\text{Synthesis}} \\
 \text{Part} \quad ; \forall k.(\exists w.Q(a,k,w) \vee \exists y.R(a,y)), \theta \rightarrow \exists y.R(a,y)
 \end{array}$$

$$P(a,b), \Gamma, \Delta, \Pi, \theta \rightarrow \exists y.R(a,y)$$

(4) An asserted compound statement is of the form

$$\{P(x,y)\} \text{ begin } B_1; B_2; \dots; B_n \text{ end } \{R(x,y)\}$$

and its verification conditions are those of

$$\{P(x,y)\} \text{ begin } B_1; \dots; B_{n-1} \text{ end } \{Q(x,y)\} \text{ and } \{Q(x,y)\} B_n \{R(x,y)\}.$$

Its templet is

$$\begin{array}{c}
 \frac{Q(a,c), \Pi \rightarrow \exists y.R(a,y)}{P(a,b), \Gamma \rightarrow \exists w.Q(a,w) \quad ; \quad \exists w.Q(a,w), \Pi \rightarrow \exists y.R(a,y)} \\
 \frac{P(a,b), \Gamma, \Pi \rightarrow \exists y.R(a,y)}{P(a,b), \Gamma, \Pi \rightarrow \exists y.R(a,y)}
 \end{array}$$

These (1)-(4) has been studied in Takasu (1981). The following examples exhibit how to use (1)-(4) for the synthesis of programs and also give a preliminary observation for the study of procedure statement and procedure declaration:

Example 1:(Quotient and Remainder)

Specification: For a pair of natural numbers a and non zero b , there exist the natural numbers q and r such that $a=b*q+r$ and $b>r$.

To prove this specification, we use the following lemma:

Lemma: If $b>0$, then there holds for any k either (case 1) there exists R such that $a=b*k+R$, or (case 2) there exist q and r such that $a=b*q+r$ and $b>r$.

The specification follows from the lemma (Termination Part):

To prove this, it suffices proving that case 1 implies the specification: If we take $a+1$ for k , then $a=b*(a+1)+R$ never holds since $b>0$ implies $b*(a+1)>a$. This falsehood implies the specification.

(The enough number of visits $a+1$, makes the loop invariant (case 1) false and the specification is attained by case 2; this is the termination.)

To prove the lemma, we use induction on k (Synthesis Part):

Initial step ($k:=0$;): We take a for R ($R:=a$;) so that case 1 holds for $k=0$.

Inductive step: (a) Case 2 for k implies case 2 for $k+1$. (b)

Assume case 1 for k , namely $a=b*k+R$: We then consider the subcases with respect to $R>b$ v $b>R$. (Case $R>b$) (while $R>b$ do begin $k:=k+1$;))

Take R of $k+1$ -th step to be $R-b$ ($R:=R-b$) so that $a=b*k+R$ implies $a=b*(k+1)+R-b$. (Case $b>R$) (end;) If $b>R$, then case 2 can be proved by taking k for q ($q:=k$;) and R for r ($r:=R$;)).

We observe the relation between the above proof of the specifica-

tion and the following program:

```

begin k:=0; R:=a; while R>b do begin K:=k+1; R:=R-b end;
      q:=k; r:=R
end

```

Example 2: (GCD) To specify a program to compute the gcd of two natural numbers, we define the following predicates: $y|x \equiv \exists z. x=y*z$ and $GCD(a,b,z) = z|a \wedge z|b \wedge \forall y. (y|a \wedge y|b \supset y|z)$. We also assume the following propositions:

Proposition: (i) $\forall u. \forall v. (GCD(a,b,u) \wedge GCD(a,b,v) \supset u=v)$.

(ii) $\forall u. (GCD(b, a \bmod b, u) \supset GCD(a,b,u))$.

(iii) $GCD(a,0,a)$.

Specification: For a positive number a and non negative number b , there exists z such that $GCD(a,b,z)$.

Lemma: If a is positive and b is non negative, then for any non negative k , either (case 1) there exist a positive number x and a non negative number y such that $\max(a,b) - k \geq y$ and $GCD(x,y,u)$ implies $GCD(a,b,u)$ for any u , or (case 2) there exists z such that $GCD(a,b,z)$.

The specification follows from the lemma: First we take $\max(a,b)$ for k . (case 1) We take x for z of the specification and also x for u . Then we have $y=0$ which implies $GCD(x,y,x)$. So $GCD(a,b,z)$ in case 1 implies $GCD(a,b,z)$ of the specification. Note that $\max(a,b)$ is the enough number of visits to the loop and the loop invariant exactly implies $GCD(a,b,z)$.

We prove the lemma by induction on k :

Initial step : We take a for x and b for y in the lemma in the case of $k=0$ ($x:=a; y:=b$). Then the lemma is clear since case 1 trivially holds.

Inductive step: The case 2 for k and the case 2 for $k+1$ are

identical statements. Now we assume the case 1 for k . Then we make the case analysis with respect to $y=0 \vee y \neq 0$:

(Case $y \neq 0$) (while $y \neq 0$ do begin) : For this case, we prove the case 1 for $k+1$ from case 1 for k : Let x and y denote themselves for k -th step and X and Y for $k+1$ -th step. Now we take y for Y and $x \bmod y$ for Y ($r := x \bmod y$; $x := y$; $y := r$ end;). Then we verify

- i) $x > 0$, $y \geq 0$ and $y \neq 0$ implies $y > 0$ and $x \bmod y \geq 0$.
- ii) $\max(a, b) - k \geq y > x \bmod y$ implies $\max(a, b) - (k+1) \geq x \bmod y$.
- iii) $\text{GCD}(y, x \bmod y, u)$ implies $\text{GCD}(x, y, u)$.

(Case $y = 0$) For this case we prove that the case 1 for k implies the case 2 for $k+1$. We take x for z in case 2, then to be proved is $\text{GCD}(x, 0, x)$ ($z := x$;).

Then we have the following program:

```

begin      x:=a; y:=b;
           while  $y \neq 0$  do begin  $r := x \bmod y$ ;  $x := y$ ;  $y := r$ 
                               end ;  $z := x$ ;
end

```

Example 3: (Factorial) We consider to compute the factorial of an non negative integer as follows:

Axiom: (i) $\text{isfact}(0, 1)$. (ii) For any non negative integer u , v , and x , $\text{isfact}(x, v)$ and $u = (x+1) * v$ imply $\text{isfact}(x+1, u)$.

Specification: If n is non negative, then there exists r such that $\text{isfact}(n, r)$.

Lemma: For any non negative k , either (Case 1) there exists a non negative m such that i) $n \geq m = n - k \geq 0$ and ii) for any u , $\text{isfact}(m, u)$ implies there exists r satisfying $\text{isfact}(n, r * u)$, or (Case 2) there exists z such that $\text{isfact}(n, z)$.

The specification follows from the lemma: We take $n+1$ for k , then the false-hood of $m > 0$ implies the specification and Case 2 implies also the specification.

We prove the lemma by induction on k :

Initial step: We prove Case 1 for $k=0$. We take n for m and 1 for r , then $\text{isfact}(n, u)$ implies $\text{isfact}(n, 1 * u)$ (begin $M:=n$; $r:=1$;).

Inductive step: We separate the cases with respect to $m=0 \vee m > 0$ where this m is the one in k -th step.

$(m > 0)$: (while $m > 0$ do begin) We prove Case 1 of the $k+1$ -th step. We take $m-1$ for m of the $k+1$ -th step ($m:=m-1$;). Then $m > 0$ and $n-k=m$ imply $n-(k+1)=m-1$. We take $m*u$ for u of the k -th step, then $\text{isfact}(m-1, u)$ implies $\text{isfact}(m, m*u)$ by Axiom (ii). So there remains to prove that if $\text{isfact}(n, r*m*u)$, then there exists r such that $\text{isfact}(n, r*u)$. We take $r*m$ for this r ($r:=r*m$ end).

$(m=0)$: We prove Case 2 of the $k+1$ -th step. We take 1 for u and r for z of the $k+1$ -th step ($z:=r$;). Then $m=0$ and $\text{isfact}(0, 1)$ imply $\text{isfact}(m, 1)$, and $\text{isfact}(n, r*1)$ implies $\text{isfact}(n, r)$.

Thus obtained program is

begin $m:=n$; $r:=1$; while $m > 0$ do begin $r:=m*r$; $m:=m-1$ end end;

(5) An asserted procedure statement is of the form

$\{P(w, e)\} \quad q(w, e) \quad \{R(w, e)\}$

and its verification condition is

$P(w, e) \supset U(w, e) \wedge \forall u. (W(u, e) \supset R(u, e))$ (Adaptation)

$\{U(u, v)\} \quad \underline{\text{procedure}} \quad q(u; v); \quad B \quad \{W(u, v)\}$

is the corresponding procedure declaration, u is the vector of the non-local variables which subject to change in the body, v is the vector of the rest of the non-local variables, w is a part of the program variables and e is a vector of expressions.

(5-1) Non-recursive case: The templet for non-recursive procedure statement is the following:

$$\begin{array}{c}
 \frac{R(d,e) \rightarrow R(d,e)}{W(d,e) \rightarrow W(d,e) ; R(d,e) \rightarrow \exists w.R(w,e)} \\
 \frac{W(d,e), W(d,e) \supset R(d,e) \rightarrow \exists w.R(d,e)}{W(d,e), \forall u.(W(u,e) \supset R(u,e)) \rightarrow \exists w.R(w,e)} \\
 \frac{U(c,e) \rightarrow U(c,e) ; \exists u.W(u,e), \forall u.(W(u,e) \supset R(u,e)) \rightarrow \exists w.R(w,e)}{\text{Specification of the procedure declaration}} \\
 \frac{\begin{array}{l} \Gamma, U(c,r) \rightarrow \exists u.W(u,r) \\ \Gamma \rightarrow U(c,r) \supset \exists u.W(u,r) \end{array} \quad \begin{array}{l} U(c,e) \supset \exists u.W(u,e), U(c,e), \\ \forall u.(W(u,e) \supset R(u,e)) \rightarrow \exists w.R(w,e) \end{array}}{\Gamma \rightarrow \forall v.(U(c,v) \supset \exists u.W(u,v)) ; \forall u.(W(u,e) \supset R(u,e)) \rightarrow \exists w.R(w,e)} \\
 \frac{\Gamma \rightarrow \forall v.(U(c,v) \supset \exists u.W(u,v)) ; \forall u.(W(u,e) \supset R(u,e)) \rightarrow \exists w.R(w,e)}{U(c,e), \forall u.(W(u,e) \supset R(u,e)), \Gamma \rightarrow \exists w.R(w,e)} \\
 \text{Verification condition} \\
 \frac{\begin{array}{l} P(c,e), \Gamma \\ \rightarrow U(c,e) \wedge \forall u.(W(u,e) \supset R(u,e)); \end{array} \quad \begin{array}{l} U(c,e) \wedge \forall u.(W(u,e) \supset R(u,e)), \\ \Gamma \rightarrow \exists w.R(w,e) \end{array}}{P(c,e), \Gamma \rightarrow \exists w.R(w,e)}
 \end{array}$$

(5-2) Recursive case:

Let K be an integer variable not appearing in the procedure body B . We put a virtual assignment statement " $K:=K+1$;" in front of the body B and the initial value of K is set to be -1 when the procedure is called up at a virtual place outside of the declaration. The value of K is specific for that body for each recursive call, namely, if the

control is in the body for the first time, then $K=0$ and the recursive call in this body invokes another copy of the body and we have $K=1$ in this copy. The value of K is called the recursion depth of each stage.

Let u_K and v_K be the values of parameters at the time when the virtual assignment statement $K:=K+1$ is executed, and $g(u_K, v_K)$ a non-negative integer valued function satisfying $g(u_K, v_K) > g(u_{K+1}, v_{K+1})$ (strictly decreasing function with respect to the recursion depth).

Now we consider

$$\text{SPEC} \equiv \forall u. \forall v. (U(u, v) \wedge k = g(u, v) \geq 0 \supset \exists z. W(z, v))$$

as the specification of the recursive procedure declaration. To prove the specification, we first apply the course-of-value induction, namely we may assume

$$\text{COURSE} \equiv \forall h. (k > h \supset \forall u. \forall v. (U(u, v) \wedge h = g(u, v) \geq 0 \supset \exists z. W(z, v)))$$

always during the proving of the specification. Therefore, at the outset we have

$$\begin{array}{l} \frac{U(c, d), k = g(c, d) \geq 0, \text{COURSE} \rightarrow \exists z. W(z, d)}{\text{COURSE} \rightarrow U(c, d) \wedge k = g(c, d) \geq 0 \supset \exists z. W(z, d)} \\ \hline \text{COURSE} \rightarrow \forall u. \forall v. (U(u, v) \wedge k = g(u, v) \geq 0 \supset \exists z. W(z, v)). \end{array}$$

Now, the templet for recursive procedure statements is as follows:

$$\begin{array}{l} \frac{\frac{\frac{R(\alpha, e) \rightarrow R(\alpha, e)}{W(\alpha, e) \rightarrow W(\alpha, e); R(\alpha, e) \rightarrow \exists w. R(w, e)} \quad \frac{W(\alpha, e), W(\alpha, e) \supset R(\alpha, e) \rightarrow \exists w. R(w, e)}{W(\alpha, e), \forall u. (W(u, e) \supset R(u, e)) \rightarrow \exists w. R(w, e)}}{U(a, e), k_0 = g(a, e) \geq 0 \rightarrow U(a, e) \wedge k_0 = g(a, e) \geq 0; \exists z. W(z, e), \forall u. (W(u, e) \supset R(u, e)) \rightarrow \exists w. R(w, e)} \\ \hline U(a, e) \wedge k_0 = g(a, e) \geq 0 \supset \exists z. W(z, e), U(a, e), k_0 = g(a, e) \geq 0, \\ \forall u. (W(u, e) \supset R(u, e)) \rightarrow \exists w. R(w, e) \\ \hline \frac{k > k_0 = g(a, e) \quad \forall u. \forall v. (U(u, v) \wedge k_0 = g(u, v) \geq 0 \supset \exists z. W(z, e)), k_0 = g(a, e) \geq 0, \\ + k > k_0 \quad U(a, e), \forall u. (W(u, e) \supset R(u, e)) \rightarrow \exists w. R(w, e)}{\quad} \end{array}$$

(continues to the next page)

$$\begin{array}{l}
\frac{k > k_0 \supset \forall u. \forall v. (U(u, v) \wedge k_0 = g(u, v) \geq 0 \supset \exists z. W(z, v)), U(a, e), \\
k > k_0 = g(a, e) \geq 0, \forall u. (W(u, e) \supset R(u, e)) \rightarrow \exists w. R(w, e)}{U(a, e) \wedge k > k_0 = g(a, e) \geq 0 \wedge \forall u. (W(u, e) \supset R(u, e)), \\
\text{Verification Condition} \quad \forall h. (k > h \supset \forall u. \forall v. (U(u, v) \wedge h = g(u, v) \geq 0 \supset \exists z. W(z, v))) \rightarrow \exists w. R(w, e)} \\
P(a, e), k = g(c, d) \geq 0 \quad \exists k_0. (U(a, e) \wedge k > k_0 = g(a, e) \geq 0 \wedge \forall u. (W(u, e) \supset R(u, e))), \\
\rightarrow \exists k_0. (U(a, e) \wedge \forall h. (k > h \supset \forall u. \forall v. (U(u, v) \wedge h = g(u, v) \geq 0 \supset \exists z. W(z, v)))) \\
k > k_0 = g(a, e) \geq 0 \wedge \rightarrow \exists w. R(w, e) \\
\forall u. (W(u, e) \supset R(u, e)); \\
\hline
P(a, e), k = g(c, d) \geq 0, \forall h. (k > h \supset \forall u. \forall v. (U(u, v) \wedge h = g(u, v) \geq 0 \supset \exists z. W(z, v))) \\
\rightarrow \exists w. R(w, e)
\end{array}$$

Example 4: We consider a procedure to compute the gcd of x and y:

Specification: For any y and x, if x is positive and y is non negative, then there exists z such that GCD(x, y, z) holds.

In this specification, we regard y as g(x, y, z). Therefore, we may assume that for any h less than y, if x is positive and h is non negative, then there exists z such that GCD(x, h, z) holds, applying the course-of-value induction on y. This assumption is called the inductive assumption.

Now we separate the cases with respect to $y=0 \vee y>0$ (if $y=0$ then).

($y=0$): We take x for z in the conclusion ($z:=x$ else). Then the conclusion is clear from Proposition (iii) in Example 2.

($y>0$): For this case, we use Proposition (ii) as the adaptation (gcd(y, x mod y, z)). So we take x mod y for h in the inductive hypothesis so that $y > x \bmod y$ holds. Furthermore, we take y for x in the inductive hypothesis so that we may assume GCD(y, x mod y, z) holds. Thus we can now use Proposition (ii) to prove the conclusion.

This proof corresponds to the following procedure declaration:

```

procedure gcd(x, y: integer; var z: integer);
begin if y=0 then z:=x else gcd(y, x mod y, z) end;

```

Example 5: We consider a procedure to compute $m!$.

Specification: For any non negative integer m , there exists z such that $\text{isfact}(m,z)$ holds.

Now we regard m as $g(m,z)$ so that we have to prove:

Lemma 0: Assumption: (i) m is a non negative integer. (ii) For any non negative integer h less than m , there exists Z such that $\text{isfact}(h,Z)$ holds. Conclusion: There exists z such that $\text{isfact}(m,z)$ holds.

To prove Lemma 0, we separate the cases with respect to $m=0 \vee m>0$ (if $m=0$ then).

($m=0$): Combining the axiom $\text{isfact}(0,1)$ and the fact $m=0$, we have $\text{isfact}(m,1)$ taking 1 for z ($z:=1$ else) in the conclusion.

($m>0$): Under $m>0$, we may take $m-1$ for h in the assumption (ii). Therefore we must prove that $\text{isfact}(m-1,Z)$ and $m>0$ imply the existence of z satisfying $\text{isfact}(m,z)$. To do this, we prove the following lemmas:

Lemma 1: (Adaptation) If $m>0$, $\text{isfact}(m-1,Z)$ and $m-1 \geq 0$ and for any t , $\text{isfact}(m-1,t)$ implies $\text{isfact}(m,m*t)$, then there exists w such that $\text{isfact}(m,m*w)$.

Lemma 2: If there exists w such that $\text{isfact}(m,m*w)$, then there exists z such that $\text{isfact}(m,z)$.

Lemma 3: If $m>0$ and for any non negative u , v and x , $\text{isfact}(x,v)$ and $u=(x+1)*v$ imply ($\text{isfact}(x+1,u)$), then for any t , $\text{isfact}(m-1,t)$ implies $\text{isfact}(m,m*t)$.

Lemma 1 is clear if we take Z for t and w (begin $\text{fact}(m-1,w)$;end).

Lemma 2 is clear if we take $m*w$ for z ($z:=m*w$ end).

Lemma 3 is clear if we take $m*t$ for u , t for v and $m-1$ for x .

Thus we have the following procedure declaration:

```

procedure fact(m:integer; var z:integer);
var w:integer;
begin if m=0 then z:=1
      else begin fact(m-1,w); z:=m*w end
end

```

Example 6:(Quicksort) We consider the following procedure declaration:

```

procedure quicksort(var A:array [1..N] of integer;
                    m,n:integer);
var i,j:integer;
begin if n>m then begin partition(A,i,j,m,n);
                  quicksort(A,m,j);
                  quicksort(A,i,n)
                end
end

```

The verification of this declaration was given by M. Foley and C.A.R. Hoare (1971). This is specified as

$$\text{Concl}d(A, A_0, m, n) = \text{Sorted}(A, m, n) \wedge \text{Perm}(A, A_0, m, n)$$

where

$$\text{Sorted}(A, m, n) = \forall p. \forall q. (n \geq q \geq p \geq m \supset A[q] \geq A[p])$$

$$\text{Perm}(A, A_0, m, n) = \exists \begin{pmatrix} m & m+1 & \dots & n \\ i_1 & i_2 & \dots & i_{n-m} \end{pmatrix}. A = A_0 \begin{pmatrix} m & m+1 & \dots & n \\ i_1 & i_2 & \dots & i_{n-m} \end{pmatrix}$$

and the procedure "partition" is specified as

$$\text{Part}d(A, i, j, m, n) \wedge \text{Perm}(A, A_0, m, n)$$

where

$$\text{Part}d(A, i, j, m, n) \equiv i > j \wedge \forall p. \forall q. (i > p \geq m \wedge n \geq q > j \supset A[q] \geq A[p]).$$

It is also known (ibid.) that to verify the body of quicksort, we have the following only one verification condition:

$$\begin{aligned}
 & A=A_0 \supset \text{if } n>m \text{ then} \\
 & \exists B. (A=B \wedge n>m \wedge \forall C. \forall i. \forall j. (\text{Partd}(C, i, j, m, n) \wedge \text{Perm}(C, B, m, n) \\
 & \supset \exists D. (C=D \wedge \forall E. (\text{Sorted}(E, m, j) \wedge \text{Perm}(E, D, m, j) \\
 & \quad \exists F. (E=F \wedge \forall G. (\text{Sorted}(G, i, n) \wedge \text{Perm}(G, F, i, n) \\
 & \quad \supset \text{Sorted}(G, m, n) \wedge \text{Perm}(G, B, m, n) \quad))) \quad)) \\
 & \text{else } \text{Sorted}(A, m, n) \wedge \text{Perm}(A, A_0, m, n)
 \end{aligned}$$

which is easily proved.

Now our quantified specification is

$$\forall k. \forall A_0. \forall m. \forall n. (k=n-m \geq 0 \supset \exists A. (\text{Sorted}(A, m, n) \wedge \text{Perm}(A, A_0, m, n)))$$

where $g(m, n, A_0)$ is $n-m$.

To prove the above quantified specification, we use the course-of-value induction, namely we are going to prove

$$\begin{aligned}
 & \forall h. (k>h \supset \forall A'_0. \forall m^*. \forall n^*. (h=n^*-m^* \geq 0 \supset \exists A. (\text{Sorted}(A, m^*, n^*) \wedge \text{Perm}(A, A'_0, m^*, n^*))) \\
 & \rightarrow \forall A_0. \forall m. \forall n. (k=n-m \geq 0 \supset \exists A. (\text{Sorted}(A, m, n) \wedge \text{Perm}(A, A_0, m, n))).
 \end{aligned}$$

First, we make the case analysis of $k=n-m>0 \vee k=n-m=0$ (if $n>m$ then).

($n>m$): To explain the detail, we set the following abbreviation:

$$\begin{aligned}
 & L7 \equiv E=F \wedge \forall G. (\text{Concl}(G, F, i, n) \supset \text{Concl}(G, B, m, n)) \\
 & L5 \equiv C=D \wedge \forall E. (\text{Concl}(E, D, m, j) \supset \exists F. L7) \\
 & L2 \equiv A_0=B \wedge n>m \wedge \forall C. \forall i. \forall j. (\text{Partd}(C, i, j, m, n) \wedge \text{Perm}(C, B, m, n) \supset \exists D. L5) \\
 & \text{Course}(h, k) \equiv k>h \supset \forall A'_0. \forall m^*. \forall n^*. (h=n^*-m^* \geq 0 \supset \exists A. \text{Concl}(A, A'_0, m^*, n^*)).
 \end{aligned}$$

Using the cut inference with $\exists B. L2$, we introduce $L2(B/B)$ to the assumption, since $\exists B. L2$ can be proved easily. So we have

$$k=n-m>0, L2(B/B), \forall h. \text{Course}(h, k) \rightarrow \exists A. \text{Concl}(A, A_0, m, n).$$

Since

$$n>m \rightarrow \exists C. \exists i. \exists j. (\text{Partd}(C, i, j, m, n) \wedge \text{Perm}(C, A_0, m, n) \wedge n \geq i > j \geq m)$$

is the specification of the procedure "partition", we may introduce $\text{Partd}(\underline{C}, \underline{i}, \underline{j}, m, n) \text{ Perm}(\underline{C}, A_0, m, n)$ to the assumption so that the same formula in $L2(\underline{B}/\underline{B})$ can be eliminated (begin partition(A, i, j, m, n);). We have

$$\exists D. L5(\underline{B}/\underline{B}, \underline{C}/\underline{C}, \underline{i}/\underline{i}, \underline{j}/\underline{j}), \forall h. \text{Course}(h, k), A_0 = B, n \geq \underline{i} > \underline{j} \geq m, n - m = k > 0 \\ \rightarrow \exists A. \text{Concl}d(A, A_0, m, n).$$

We make two copies of $\forall h. \text{Course}(h, k)$ by contraction inference for the purpose to process L5 and L7. Now to process L5, we take $j - m$ for h .

Then

$$n \geq \underline{i} > \underline{j} \geq m, k = n - m \rightarrow k > j - m$$

so we may take m for m^* , $\underline{C}$ for A_0' , and $\underline{j}$ for n^* . Since $j - m \geq j - m \geq 0$ is clear, we introduce $\underline{A}_1$ for A in $\text{Course}(\underline{j}, k)$. We introduce $\underline{D}$ for D and we take $\underline{A}_1$ for E in $L5(\underline{B}/\underline{B}, \underline{C}/\underline{C}, \underline{i}/\underline{i}, \underline{j}/\underline{j})$ so we have

$$\text{Concl}d(\underline{A}_1, \underline{C}, m, \underline{j}), \text{Concl}d(\underline{A}_1, \underline{D}, m, \underline{j}) \supset \exists F. L7(\underline{B}/\underline{B}, \underline{i}/\underline{i}, \underline{A}_1/E), \underline{D} = \underline{C}, A_0 = \underline{B}, \\ \forall h. \text{Course}(h, k), n \geq \underline{i} > \underline{j} \geq m, n - m = k \rightarrow \exists A. \text{Concl}d(A, A_0, m, n)$$

Now we can cancel $\text{Concl}d(\underline{A}_1, \underline{D}, m, \underline{j})$ by $\text{Concl}d(\underline{A}_1, \underline{C}, m, \underline{j})$ and $\underline{D} = \underline{C}$ (the adaptation is cancelled: $\text{quicksort}(A, m, j)$);).

Similarly, to process L7, we take $n - \underline{i}$ for h , $\underline{F}$ for F and A_0' , $\underline{i}$ for m^* and n for n^* and we introduce $\underline{A}_2$ for A in $\text{Course}(n - \underline{i}, k)$. So we have

$$\text{Sorted}(\underline{A}_2, \underline{i}, n) \wedge \text{Perm}(\underline{A}_2, \underline{F}, \underline{i}, n) \supset \text{Sorted}(\underline{A}_2, m, n) \text{ Perm}(\underline{A}_2, \underline{B}, m, n),$$

$$\text{Sorted}(\underline{A}_2, \underline{i}, n) \wedge \text{Perm}(\underline{A}_2, \underline{F}, \underline{i}, n), A_0 = B, \underline{A}_1 = \underline{F} \rightarrow \exists A. \text{Concl}d(A, A_0, m, n)$$

Now we may cancel $\text{Sorted}(\underline{A}_2, \underline{i}, n) \wedge \text{Perm}(\underline{A}_2, \underline{F}, \underline{i}, n)$ ($\text{quicksort}(A, i, n)$ end) and we arrive at the stage to determine A in the conclusion to be $\underline{A}_2$.

This example is very typical for the non recursive procedure statement and recursive procedure statements.

($n = m$): For this case we take A_0 for A in the conclusion, so nothing to do for computation.

(6) An asserted for-statement is of the form

(i) $\{P\} \text{ for } i:=m \text{ to } n \text{ do } B \quad \{Q([m..n])\}$

or

(ii) $\{P\} \text{ for } i:=n \text{ down to } m \text{ do } B \quad \{Q([m..n])\}$

and the corresponding verification conditions are

(i) $P \supset Q([m..m])$ and those of

$$\{n \geq i \geq m \wedge Q([m..i])\} \quad B \quad \{Q([m..i])\},$$

(ii) $P \supset Q([n..n])$ and those of

$$\{n \geq i \geq m \wedge Q([i..n])\} \quad B \quad \{Q([i..n])\}$$

respectively. The corresponding templets are the followings:

$$\begin{array}{l} \text{(i)} \quad \frac{\Gamma, Q(a, [m..m+k], d), n \geq m+k+1 \rightarrow \exists y. Q(a, [m..m+k+1], y)}{n \geq m+k+1 \rightarrow n \geq m+k; \quad \Gamma, \exists y. Q(a, [m..m+k], y), n \geq m+k+1 \rightarrow \exists y. Q(a, [m..m+k+1], y)} \\ \frac{P(a, c), \Gamma \rightarrow \quad \Gamma, n \geq m+k+1, n \geq m+k \supset \exists y. Q(a, [m..m+k], y) \rightarrow \exists y. Q(a, [m..m+k+1], y)}{n \geq m \supset \exists y. Q(a, [m..m], y); \quad \Gamma, n \geq m+k \supset \exists y. Q(a, [m..m+k], y) \rightarrow n \geq m+k+1 \supset \exists y. Q(a, [m..m+k+1], y)} \\ \hline \Gamma, P(a, c) \rightarrow n \geq m+(n-m) \supset \exists y. Q(a, [m..m+(n-m)], y) \end{array}$$

$$\begin{array}{l} \text{(ii)} \quad \frac{\Gamma, Q(a, [n-k..n], d), n-(k+1) \geq m \rightarrow \exists y. Q(a, [n-(k+1)..n], y)}{n-(k+1) \geq m+n-k \geq m; \Gamma, \exists y. Q(a, [n-k..n], y), n-(k+1) \geq m \rightarrow \exists y. Q(a, [n-(k+1)..n], y)} \\ \frac{\Gamma, n-(k+1) \geq m, n-k \geq m \supset \exists y. Q(a, [n-k..n], y)}{\quad \rightarrow \exists y. Q(a, [n-(k+1)..n], y)} \\ \hline P(a, c), \Gamma \rightarrow \quad \Gamma, n-k \geq m \supset \exists y. Q(a, [n-k..n], y) \\ n \geq m \supset \exists y. Q(a, [n..n], y); \quad + \quad n-(k+1) \geq m \supset \exists y. Q(a, [n-(k+1)..n], y) \\ \hline \Gamma, P(a, c) \rightarrow n-(n-m) \geq m \supset \exists y. Q(a, [n-(n-m)..n], y). \end{array}$$

Example 7: We consider the following program:

```
var i, h: m..n; max: T; A: array of T;
begin h:=m; max:=A[m];
```

```

for i:=m to n do if A[i]>max then
    begin max:=A[i] ; h:=i end
end

```

We set

$$Q([m..m+k], h, \max) \equiv n \geq h \geq m \wedge \max = A[h] \wedge \forall j. (m+k > j \geq m \supset \max \geq A[j]);$$

then the specification of the above program can be expressed as

$$\exists h. \exists \max. Q([m..n], h, \max) .$$

To prove this specification, we use the induction on k:

Initial step: To prove $\exists h. \exists \max. Q([m..m], h, \max)$, we take m for h and A[m] for max (begin h:=m ; max:=A[m] ;). Then $Q([m..m], m, A[m])$ is clear.

Inductive step: To prove

$$Q([m..m+k], h, \max), n \geq m+k+1 \rightarrow \exists h. \exists \max. Q([m..m+k+1], h, \max)$$

we make the case analysis of $A[m+k+1] > \max \vee \max \geq A[m+k+1]$.

($A[m+k+1] > \max$: (if A[i]>max then begin): For this case, we take m+k+1 for h and A[m+k+1] for max. Then $Q([m..m+k+1], m+k+1, A[m+k+1])$ can be proved easily from $Q([m..m+k], h, \max)$ by making the case analysis of $j=m+k+1 \vee m+k > j \geq m$. (h:=i; max:=A[i] end).

($\max \geq A[m+k+1]$): For this case, we take h for h and max for max.

(Here we have no computation so that we have no else-part (end).)

(7) An asserted case-statement is of the form

$$\{P \wedge (x \in \{k_1, \dots, k_n\})\} \text{ case } x \text{ of } k_1:S_1; \dots; k_n:S_n \text{ end } \{Q\}$$

where x is the selector, k_i ($n \geq i \geq 1$) a label and S_i a statement. The verification conditions for this statement are those of

$$\{P \wedge x=k_i\} S_i \quad \{Q\} \quad \text{for } n \geq i \geq 1.$$

Its templet is

$$\frac{x=k_{n-1}, P(a, c) \rightarrow \exists y. Q(a, y); x=k_n, P(a, c) \rightarrow \exists y. Q(a, y)}{x=k_{n-1} \vee x=k_n, P(a, c) \rightarrow \exists y. Q(a, c)}$$

$$\dots$$

$$\frac{x=k_1, P(a, c) \rightarrow \exists y. Q(a, y); x=k_2 \vee \dots \vee x=k_n, P(a, c) \rightarrow \exists y. Q(a, y)}{x=k_1 \vee x=k_2 \vee \dots \vee x=k_n, P(a, c) \rightarrow \exists y. Q(a, y)}$$

(8) An asserted repeat-until-statement is of the form

$$\{P\} \text{ repeat } B \text{ until } t \quad \{Q \wedge t\}$$

where B is a statement and t is a quantifier-free formula in L . Its verification conditions are

$$Q \wedge \neg t \supset P$$

and those of

$$\{P\} \quad B \quad \{Q\}.$$

Since the above statement is equivalent to the statement

$$\{P\} \text{ begin } B; \text{ while } \{Q\} \quad \neg t \text{ do } B \{Q \wedge t\},$$

its templet can be similar to the templet for while-statement. But, the initial step of the induction used in the templet of while-statement must determine the same statement B as in the case of $\neg t$ at the inductive step. This implies that our system must check whether two parts of the proof are identical or not. By this reason, we exclude the repeat-until statements in our system together with the go-to-statements.

2. AN INTERACTIVE PROGRAM SYNTHESIZER:

Our proof-checker system interactively constructs proofs in many-sorted formal systems having a common syntactic rules to form terms and formulas and a common set of inference rules LJ+Induction. Each user must declare first a language (consisting of parameters, variables, function symbols and predicate symbols to be used) and a set of axioms. These materials are used to construct Pascal declarations automatically.

The declared function symbols and predicate symbols consist of those built in Pascal and/or those with the previously proved specifications being declared as axioms. The latter kind of axioms are associated with Pascal function and procedure declarations stored in a file.

After the language and axiom declarations, the interactive sessions take place for constructing a proof. At the outset, a sequent to be proved is supplied from the user, together with the informations whether the user regards the sequent as a specification of a program or of a procedure and which bound variables are regarded as output variables (or variable parameters of the procedure). Then the construction of the proof starts with this sequent upwards and right most first (with respect to the proof-tree), receiving some inference commands. Here we think, for each command, "to prove the conclusion of the inference, it suffices to prove the premisses" . The inference commands consist of, not only the primitive inferences in LJ+Induction, but also some compound inference commands. Each of these compound inference commands is associated with a templet of a proof which corresponds to a Pascal statement as we describe these in Section 1. Each of these compound inference commands constructs a part of the proof according to the corresponding templet and also it constructs a part of the program forming the corresponding statement

in a top-down manner using some stacks and in the manner of Dijkstra's structured programming.

Besides the usage of the compound inference commands, the system keeps the track of term dependence for the purpose to detect the input variables (or value parameters) and the program variables (or local variables). If a term t is selected for an existentially quantified bound variable z (by an inference $\rightarrow E$), then the bound variables whose eigen parameters (introduced by $\exists \rightarrow$ or $\rightarrow V$) are occurring in t , are said to be dependent on the target variable z . Such dependent variables in the right branch of a cut or induction inference may be depended by some other variables in the left branch. So we may extend transitively the dependence relation over the entire proof. After the completion of the proof, the variables, dependent on the output variables and not having dependent variables, are input variables (or value parameters). The variables, dependent on the output variables and not being input variables, are program variables (or local variables).

At the end of the proving of a quantified specification, an executable Pascal program or a function or procedure declaration will remain in a file. If it is an executable program, then it can be fed into a Pascal compiler. The constructed program is always totally correct provided that the quantified specification was the right one.

REFERENCES

- [1] Ackerman, W.: Zur Widerspruchsfreiheit der Zahlentheorie,
Math. Ann. 117(1940)162-194.
- [2] Alagic, S. & Arbib, M.A.: The design of Well-Structured and Correct
Programs, Springer-Verlag 1978.
- [3] Bibel, W. & Schreiber, J.: Proof search in a Gentzen-like system
of first-order logic, In E. Gelenbe & D. Potier ed.,
International Computing Symposium 1975 Amsterdam, North-
Holland.
- [4] Bledsoe, W.W.: A man-machine theorem proving system, Artificial
Intelligence 5(1974)51-72.
- [5] Bledsoe, W.W.: Non-resolution theorem proving, Artificial Intelli-
gence 9(1977)1-35.
- [6] Bledsoe, W.W. & Tyson, M.: The UT interactive theorem prover, Memo.
ATP-17a, The Univ. of Texas at Austin, Math. Dept. 1978.
- [7] Dahl, O.J. & Hoare, C.A.R.: Hierarchical program structures, In O.J.
Dahl, E.W. Dijkstra & Hoare: Structured Programming, Academic
Press (1972)197-220.
- [8] Darlington, J.: Application of program transformation to program
synthesis, In Colloques IRIA on Proving and Improving Programs,
Arc et Senans, (1975)133-144.
- [9] Darlington, J.: Synthesis of several sorting algorithms, Acta
Informatica 11(1978)1-30
- [10] Dijkstra, E.W.: A Discipline of Programming, Prentice Hall, 1976.
- [11] van Emden, M.H. & Kowalsky, R.A.: The semantics of predicate logic
as a programming language, JACM 23(1976)23-32.
- [12] Floyd, R.W.: Assigning meaning to programs, In Proc. of the Symp.
in Appl. Math. vol. 19, AMS Providence (1967)19-32.

- [13] Floyd,R.W.: Toward interactive design of correct programs,Proc. of IFIP Congress (1971)7-10, North-Holland.
- [14] Foley,M. & Hoare,C.A.R.: Proof of a recursive program: Quicksort, Computer J. 14(1971)391-395.
- [15] Gentzen,G.: Untersuchungen über das logische Schliessen, I, II, Math. Z. 39(1935)176-210.
- [16] Gödel,K.: Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes, Dialectica 12(1958)76-82.
- [17] Green,C.: The design of PSI program synthesis system, In Proc. of 2nd Int. Joint Conference on Software Engi.(1976) San Fransisco.
- [18] Hoare,C.A.R.: Algorithm 65: FIND CACM 4(1961)321.
- [19] Hoare,C.A.R.: The Axiomatic Basis of Computer Programming, CACM 12(1969)576-583.
- [20] Hoare,C.A.R.: Proof of a program FIND, CACM 14(1971)39-45.
- [21] Hoare,C.A.R.: Procedures and Parameters; an axiomatic approach, Symp. on Semantics of Algorithmic Language, (E. Engeler edi.) Lec. Note in CS 188 (1971).
- [22] Hoare,C.A.R.: A note on the for statement,BIT 12(1972)334-341.
- [23] Hoare,C.A.R. & Wirth,N.: An axiomatic definition of PASCAL, Acta Informatica 2(1973)335-355.
- [24] Igarashi,S. et al.: Automatic program verification I: Logical basis and its implementation, Acta Informatica 4(1975)145-182.
- [25] Kleene,S.C.: Introduction to Metamathematics, van Nostrand 1952.
- [26] Kowalski,R.: Predicate logic as a programming language,Proc. of IFIP Congress (1974)569-574, Amsterdam, North-Holland.
- [27] Kowalski,R.: Logic for Problem Solving, 1979 North-Holland.
- [28] Kreisel,G.: Some uses of proof theory for finding computer programs, Colloque Int. des Logique (1977)123-134, Paris.
- [29] Luckhum,D. & Nilsson,N.J.: Extracting Information from resolution

proof trees, Artificial Intelligence 2(1971)

- [30] Luckhurn D.C.: Program Verification and Verification-Oriented Programming, Proc. of IFIP Congress (1977) 783-793, North-Holland.
- [31] Manna, Z. & Waldinger, R.: Toward automatic program synthesis, CACM 14(1971) 151-165.
- [32] Manna, Z. & Waldinger, R.: Knowledge and reasoning in program synthesis, Artificial Intelligence 6(1975) 175-208.
- [33] Manna, Z. & Waldinger, R.: Studies in Automatic Programming Logic, North-Holland 1977.
- [34] Manna, Z. & Waldinger, R.: Synthesis: Dreams - Programs, IEEE Trans. on Software Engineering 5(1979) 294-328.
- [35] Manna, Z. & Waldinger, R.: A deductive approach to program synthesis, Proc. of 6-th IJICAI (1979) 542-551 Tokyo.
- [36] McCarthy, J.: Problems in the theory of computation, Proc. of IFIP Congress (1965) 219-222, W.A. Kalenich ed., Spartan Book Co.
- [37] Mendelson, E.: Introduction to Mathematical Logic, D. van Nostrand, 1964.
- [38] Nauer, P.: Proofs of algorithms by general snapshots BIT (1966) 310-316.
- [39] Sato, M.: Toward a mathematical theory of program synthesis, Proc. of the 6-th IJICAI (1979) 575-762.
- [40] Schütte, K.: Proof Theory, Springer-Verlag, 1977.
- [41] Smullyan, R.: First-Order Logic, Springer-Verlag 1968.
- [42] Suzuki, N.: Automatic verification of programs with complex data structures, Ph.D. Thesis 1976 Stanford Univ., published also in Outstanding Dissertations in the Computer Science, Garland Publ. Co. 1980.
- [43] Szabo, M.E. ed.: The collected papers of Gerhard Gentzen, North-Holland Publ. Co.

- [44] Takasu, S.: Proofs and Programs, in Proc. of the 3rd IBM Symp. on Math. Foundations of Computer Science 1978, IBM Japan.
- [45] Takasu, S. & Kawabata, S.: A logical basis for programming methodology, Theoretical Computer Science 16(1981)43-60.
- [46] Takeuti, G.: Proof Theory, North-Holland 1975.
- [47] Troelstra, A.S.: Metamathematical Investigation of Intuitionistic Arithmetic and Analysis, Lec. Note in Math. No.344, 1973.
- [48] Tyugu, E.H.: A programming system with automatic program synthesis, Lec. Note in Computer Science No. 47, Methods of Algorithmic Language Implementation, (1977)251-267.
- [49] Tyugu, E.H.: Towards Practical Synthesis of Programs, Proc. of IFIP Congress (1980) S.H. Levington ed., North-Holland Publ. Co.
- [50] Waldinger, R. & R.C.T. Lee: PROW: A step toward automatic program writing, In Proc. of IJICAI (1969)241-252.