Program Construction Using State Refinement Rules

Program Construction Using State Refinement Rules

Yoneharu FUJITA* and Fujio NISHIDA*

(Received June 15, 1982)

This paper presents an approach to program construction using state-refinement rules in a top-down manner from specifications of a program.

Both the domain dependent and independent knowledges necessary to various program construction are stored in a form of the state-refinement rules. The system synthesizes specified programs by combining these rules through state unification and furthermore, optimizes them.

1. Introduction

This paper presents a method of knowledge-based semiautomatic program construction[5] with the aid of a program library. The given specifications written in state expressions are checked whether they meet a causality relation called the precedence condition. Subsequently it is examined whether the procedures that satisfy the given specification can be constructed by using expansion and linkage rules in various fields stored in a program library. These rules take a kind of form of production rules and are called transformation rules (abbreviated to TRs). Though the specifications and TRs are written on the basis of state expressions for convenience to link various specified elements, procedural expressions $F(x)$ can be also used by interpreting $\{F(x)\}$ as the state brought after the procedure has been applied. Each synthesized part of programs using TRs is optimized and more precise specifications are given by the operation parts of TRs and refined until programs written in a specified target language are obtained.

On the other hand, if the refinement process cannot proceed, the system requires more precise specifications to the user.

2. Specifications of Programs

2.1 State Expression

Basic specifications of a program are sometimes given in relation-oriented forms by a predicate formula. This formula takes one of the following forms:

$$Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \quad (1), \qquad (P \rightarrow Q_1) \wedge (\neg P \rightarrow Q_2) \quad (2), \qquad (P \rightarrow Q_1) \wedge (\neg P \rightarrow T) \quad (3)$$

where P is a conjunction of several literals and denotes a test condition, Q_1, Q_2, ... and Q_n called post conditions are literals or formulas consisting of some combination of (1), (2) and (3), and denote some relations brought by a certain operation. T denotes the value '*true*' and means no operation needed for the output condition. Each formula can take some quantifiers.

* Department of Electrical Engineering, College of Engineering.

There are many cases where procedural specifications are more convenient and suitable than declarative specifications. In these cases, procedural expressions enclosed with braces are used for the description of relations brought by the procedures. For example, $\{y:=x\}$ means an equality relation $y=x$.

A state is characterized by some relations which hold at a certain point of flowchart diagram of a program and represented by a predicate formula enclosed by braces { }.

A primed variable is introduced as a term of the value immediately after a certain operation has been applied on the same variable, for instance, as $y'=f(x, y)$.

An abbreviated description $X(1..n)$ for an ordered set is also introduced for $(X(i)|i=1..n)$. For example, $Y(1..n)=X(1..n)$ represents $\forall i\epsilon(1..n)\ Y(i)=X(i)$. Furthermore, the expression $\{\forall i\epsilon(1..n)\ S(i)\}$ denotes a state brought by iterations of an operation $S(i)$ for i from 1 to n.

2.2 Precedence Condition

Specifications of a program are given by a sequence of states which involves the input state, some intermediate states and the output state. Let $\{Q_i\}$ $(i=1, 2, \ldots)$ denote states. Then, the basic form of specifications is written as

$$\{Q_1\}\,;\ \{Q_2\}\,;\ldots\,;\ \{Q_n\}. \tag{4}$$

Each adjacent pair $\{Q_i\}\,;\ \{Q_{i+1}\}$ means that, if the system is in a state $\{Q_i\}$ it moves to the next state $\{Q_{i+1}\}$.

Furthermore, a form of repetition

$$\{*(P\rightarrow Q_1)\wedge(\neg P\rightarrow T)\} \tag{5}$$

is introduced to represent a sequence of repetition of a state $\{(P\rightarrow Q_1)\wedge(\neg P\rightarrow T)\}$ which continues so long as the condition P holds. Expression (5) can be also interpreted as a macro-state if the final state brought after repetition is taken as the state.

In expression (4), each state can be divided into several substrates by partitioning the parts of the involved logical formula ΛQ_i and they can be arranged in a form of expression (4) again.

Using the partitioned form of the predicate formula of a state together with an iterative form expression (5), users can describe the program specifications as precisely as possible if necessary.

In these specifications each variable of any state must satisfy the following precedence conditions.

(1) Any non-primed variable of the argument of a function or a test condition of a logical formula in a state S must be defined on the range of the index of a loop or at the post condition of a logical formula in a state on every path which leads to the state S.

(2) For any primed variable y', the non-primed variable y must be defined in advance.

In addition to the above state expressions, some specifications and declarations must be given in advance with respect to (a) target languages, (b) names of input and output variables and (c) types and structures of variables.

[Example 1] This example shows specifications of a program which selects passers in an entrance examination. The input data is a table named as *SCORE* which contains the identification number of every examinee and his scores of three subjects in each row. The output data are the sorted table of *SCORE* in descending order of total scores of three subjects and the mean value of the total scores. The number of examinees is M. The specifications are given as follows:

Target-L(PASCAL)
IN-OUT: *IN* (*SCORE* (1..*M*, 1..4)); *OUT* (*SCORE* (1..*M*, 1..5), *MEAN*);
TYPE: *INTEGER* (*SCORE*); *STRUCT*: *ARRAY* (*SCORE* (1..*M*, 1..5));
PROG: {*SCORE* (1..*M*, 5)=*SCORE* (1..*M*, 2)+*SCORE* (1..*M*, 3)
+*SCORE* (1..*M*, 4)
∧*AVRG*=*Mean* (*SCORE* (1..*M*, 5))}; {*SCORE'* (1..*M*, 1..5)=
sort (*SCORE* (1..*M*, 1..5), *key*=*SCORE* (\$, 5), *rel*:≧)}
where *Mean* (*X*(1..*n*)): Σ(*x*(1..*n*))/*n*
S=Σ(*X*(1..*n*)): *S*=*X*(1)∧{∀*i*∈(2..*n*)*S'*=+(*S*, *X*(*i*))}.

In the above specifications, the functions *Mean* and Σ are assumed to be non-primitive functions and defined in the 'where' part of the specifications, whereas, '*sort*' is hierarchically defined by TRs in a database as shown in Table 1.

3. Transformation Rules

3.1 The Structure of Transformation Rules

The TRs are represented on a state representation basis for convenience to link various specification entities as follows:

$$\{P(X)\}\,;\, Q(X,Y) \quad (OP\text{: } op\text{-}part),$$

where X and Y are input and output variable vectors necessary for the second-order unification and $P(X)$ and $Q(X, Y)$ are called the precondition and the postcondition of the TR respectively.

If the precondition of a TR does not contain any expression of a result of a computation to be obtained in advance, the precondition of the TR is generally omitted in this paper for simplicity. TRs with and without preconditions are called linkage and expansion types respectively.

The operation part (abbreviated to OP) suggests some procedures, state-sequences or supporting TRs which support and help refining the state transformation rule. The procedures are written in a target language such as ALGOL or LISP and prefixed by the symbol '*p*:'. The state-sequences are described in the same form as that of the specification and prefixed by the symbol '*s*:'. Likewise, the names of supporting TRs are prefixed by the symbol '*t*:'.

3.2 Classes of TRs

The TRs are classified into three classes and are attached the prefixed class numbers. The first of them contains program-oriented TRs which generate the fundamental program structures. The second consists of data-oriented TRs related

to specific data structures. The third consists of problem-oriented TRs which belong to specific problem fields such as symbol manipulation or database management system. Many TRs of the first and second classes belong to the expansion type. Table 1 shows some examples of TRs, where x, y and z are used as input, intermediate and output variables. TR 1-① and TR 1-② are TRs which give assignment and conditional statements respectively, where $\bar{S}_1$ and $\bar{S}_2$ denote some procedures and S_1 and S_2 are the relations brought by the procedures. TR 1-③ shows an iterative application of a twoplace function f to a set $X(1..n)$. TR 1-⑤ shows an iterative state expression similar to TR 1-③ in a recursive form for a certain initial value of y where $d(x)$ is a primitive function. TR 1-③ and TR 1-⑤ have a convenient form to designate an iterative expression using a higher level language specification[4], while TR 1-⑥ and ⑦ have forms directly applicable to program construction where some mathematical induction[2] needs to be applied in planning of program construction for specifications as shown in [Example 4]. Furthermore, TR 1-⑥ and TR 1-⑦ postulate relations of the form $<\{A\}\,;\,\{B\}>$ which asserts the transitivity from the state $\{A\}$ to $\{B\}$ as a precondition.

Next, a part of the class of data-oriented TRs is shown. TR 2-① represents an exchange of contents of two vector variables. TR 2-② represents the function which moves the maximum of a set to the head of the set using TR 2-①. TR 2-③ represents the sorting of a set of vectors in descending order of a certain function value of vector components as a key using the above TR 2-②, where '$' denotes an arbitrary value in $(1..n)$. As seen in the above, they constitute a hierarchical structure to some extent and are used as basic TRs of data management and symbolic manipulation.

Finally, a part of TRs associated with linear algebra are shown as an example of the class of problem-oriented TRs. TR 3-m1 and TR 3-m2 say that the dimension of a triangular and a unit submatrix contained in a matrix is extended by one respectively. TR 3-m3 assures that any matrix can be transformed into a triangular matrix and TR 3-m4 says that a triangular matrix is transformable to a unit matrix, where "*Matrix* $(A(1..k,\ 1..k+1))$", "*Partialtriangular* $(A(1..k,\ 1..k+1),\ i..k)$" and "*Partialunit* $(A(1..k,\ 1..k+1),\ 1..i)$" for any i in $(1..k)$ are predicates which assert "$A(1..k,\ 1..k+1)$ is a $k\times(k+1)$ matrix", "$A(1..k,\ 1..k+1)$ is a $k\times(k+1)$ matrix containing a triangular sub-matrix from i to k in rows and from $i+1$ to $k+1$ in columns" and "$A(1..k,\ 1..k+1)$ is a $k\times(k+1)$ matrix containing a unit submatrix from 1 to i in rows and from 2 to $i+1$ in columns", respectively.

Table 1 TRs of classes 1, 2 and 3.

1-①	$\{z=f(x)\}$ $(p: z:=f(x))$
1-②	$\{(P(x)\rightarrow S_1(z,\ x))\wedge(\neg P(x)\rightarrow S_2(z,\ x))\}$ $(p:$ *if* $P(x)$ *then* $\bar{S}_1(z,\ x)$ *else* $\bar{S}_2(z,\ x))$
1-③	$\{\forall i\in(1..n)y'=f(y,\ X(i))\}$ $(p:$ *for* $i:=1$ *to* n *do* $y:=f(y,\ X(i)))$
1-④	$\{Y(1..m,\ 1..n)=X(1..m,\ 1..n)\}$ $(s:\forall i\in(1..m)Y(i,\ 1..n)=X(i,\ 1..n))$
1-⑤	$*\{(P(x)\rightarrow y'=f(y,\ x)\wedge x'=d(x))\wedge(\neg P(x)\rightarrow T)\}$
	$(p:$ *while* $P(x)$ *do* $y:=f(y,\ x);\ x:=d(x)$ *od*$)$
1-⑥	$<\{S(n)\}>\wedge<\{S(k)\}\,;\,\{S(k-1)\}>;\ \{S(1)\}$
	$(s:\{S(n)\}\,;*\{(k>1\rightarrow(\{S(k)\}\,;\,\{S(k-1)\}))\}\wedge(\neg k>1\rightarrow T)\})$
1-⑦	$<\{S(1)\}>\wedge<\{S(k)\}\,;\,\{S(k+1)\}>;\ \{S(n)\}$
	$(s:\{S(1)\}\,;*\{(k<n\rightarrow(\{S(k)\}\,;\,\{S(k+1)\}))\wedge(\neg k<n\rightarrow T)\})$

2-① $\{X'(1..m)=Y(1..m) \wedge Y'(1..m)=X(1..m)\}$
$(s:(X'(1..m),\ Y'(1..m))=exch\ (X(1..m),\ Y(1..m)))$

2-② $\{Y(1..n)=Max(Y(1..m,\ 1..n))\}$
$(s:\{\forall i \epsilon(1..m)\forall j \epsilon(i+1..m)\ (f(Y(i,\ 1..n)) \geqq f(Y(j,\ 1..n)) \rightarrow T)$
$\wedge(\neg f(Y(i,\ 1..n)) \geqq f(Y(j,\ 1..n)) \rightarrow (Y'(i,\ 1..n)=Y(j,\ 1..n)$
$\wedge Y'(j,\ 1..n)=Y(i,\ 1..n)))\})$

2-③ $\{Y'(1..m,\ 1..n)=sort(Y(1..m,\ 1..n),\ key=f(Y(\$,\ 1..n)),\ rel:\geqq)$
$(s:\ \{Y(1..m,\ 1..n)\ \epsilon array\ (1..m,\ 1..n)\};$
$\{\forall i \epsilon(1..m) f(Y'(i,\ 1..n))=Max(f(Y'(i..m,\ 1..n)))\};$
$Y'(1..m,\ 1..n)=sort\ (Y(1..m,\ 1..n),\ key=f(Y(\$,\ 1..n)),\ rel:\geqq)\})$

3-m1 $\{Partialtriangular\ (A(1..k,\ 1..k+1),\ i..k)\};$
$\{Partialtriangular\ (A'(1..k,\ 1..k+1),\ i-1..k)\}$
$(s:\ \{\forall l \epsilon(1..i)\ \forall j \epsilon(1..i+1)\ ((l=i \rightarrow A'(l,\ j)=A(l,\ j)/A(l,\ l+1))$
$\wedge(l<i\ A'(l,\ j)=A(l,\ j)-A(l,\ i+1)*A(i,\ j)/A(i,\ i+1)))\})$

3-m2 $\{Partialunit\ (A(1..k,\ 1..k+1),\ 1..i)\};$
$\{Partialunit\ (A'(1..k,\ 1..k+1),\ 1..i+1)\}$
$(s:\ \{\forall l \epsilon(1..i)\forall j \epsilon(1..i+1) A'(i+1,\ j)=A(i+1,\ j)-A(l,\ j)*A(i+1,\ l+1)\})$

3-m3 $\{Matrix\ (A(1..k,\ 1..k+1))\};$
$\{Partialtriangular\ (A(1..k,\ 1..k+1),\ 1..k)\}$
$(t:$ 3-m1, 1-⑥$)$

3-m4 $\{Partialtriangular\ (A(1..k,\ 1..k+1),\ 1..k)\};$
$\{Partialunit\ (A(1..k,\ 1..k+1),\ 1..k)\}$
$(t:$ 3-m2, 1-⑦$)$

3.3 Search of TRs

In order to have access to appropriate TRs quickly, two kinds of tables are implemented.

One of them is an indexing table which contains headings and pointers to the locations of the bodies of TRs for class 1 and 2 or contains those to the entries of tree-like files for TRs of class 3. A heading consists of some logical symbols such as *, $\forall$, $\rightarrow$ and procedural names for class 1 and 2, and some domain or attribute names for class 3. The system has access to the heading by the aids of characteristic expressions involved in the current state expression in the application of TRs.

The others are tree-like files. The TRs of type 3 are classified into several subclasses such as algebraic formulamanipulation. In each subclass a tree-like file is implemented so that the system is accessible to appropriate TRs by refering to detailed characteristics of states such as domain of data and the number of literals as seen in problem solving.[6]

4. Refinement of Specifications

4.1 Linkage and Expansion

Specifications by a user or the expanded parts by TRs are partitioned, arranged and optimized, then the resulted states on a branch are refined from the left. The system searches a unifiable post condition of a TR with a predicate formula contained in the next leftmost state in the branch using the indexing table and files.

The principle of unification procedures is based on the rules of projection, elimination and imitation in five rules of the general second order unification introduced by T. Pietrsukowski.[1]

If a unifiable TR is found and it is an expansion type, a pointer to the OP of the TR is recorded for the expansion to be done later. If the unifiable TR is a linkage type and the unified precondition of the TR is not subsumed by the predicate formula of the leftmost state of the branch, the precondition is set as an intermediate state and a pointer to the OP of the TR is recorded. This process continues until such a unified precondition of a TR is subsumed by the predicate formula in the leftmost state of the branch or it is not needed in the appearance of an expansion type TR. This operation is called linkage of states.

If the linkage is impossible due to lack of stored knowledge, the system requests some suggestions to the user. In such a way, the linkage proceeds from the left to the right in each branch.

After finishing linkage between states in each branch, each state is expanded by replacing it by the pointed OP according to the respective types of OP as follows:

Suppose the applied TR to a pair of states $\{Q_1\}$; $\{Q_3\}$ has a form of $\{Q_2\}$; $\{Q_3\}$ $(OP\colon R)$, then a sequence of expressions $\{Q_1\}$; $\{Q_2\}$; R; $\{Q_3\}$ is constructed, where R is a sequence of states or procedures.

In the above, if the state Q_1 leads to Q_2 and R also leads to the state Q_3 without interpolating any intermediate state, $\{Q_1\}$ and $\{Q_3\}$ can be deleted. However, for the later usage as the heading of this program section and others, they are sometimes left in forms of $[Q_1]$ and $[Q_3]$.

4.2 Arrangement of Substates and Optimization

As mentioned in section 2 and 3, the state expressions in the specifications given by users or in the expansion forms obtained by applying TRs sometimes involve logical conjunctive forms $Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n$, where each $Q_i (i=1, 2, \ldots, n)$ is a literal or a complementary implication form like expression (2) or (3). The system partitions these formulas into several parts and arranges them according to the precedence condition and constructs a sequence of substates $\{Q_1\}$; $\{Q_2\}$; ...; $\{Q_n\}$. Subsequently, a global optimization is tried over a sequence of substates on the branch.

The main optimizations are (a) factoring of loop-constant expressions from an iterative state expression, (b) loop fusion for several iterative state expressions and (c) elimination of redundant computations by moving a common part of computation appearing in branches to the front of them. In practical situation such common factors often appear in various modified forms. To facilitate to deal with various forms, several optimization procedures are implemented. They are called by the aid of the type of state expressions and optimize the state expressions.

The next example shows a loop fusion

[Example 2] The first state expression in [Example 1] is rewritten by substituting the 'where' part into the main part of the specifications as follows;

$$\{SCORE\,(1..M, 5)=SCORE\,(1..M, 2)+SCORE\,(M, 3)+SCORE\,(M, 4)$$
$$\wedge MEAN=S'/M \wedge S=SCORE\,(1, 5)$$
$$\wedge \forall i \epsilon\,(2..M) S'=+(S, SCORE\,(i, 5))\}. \qquad (6)$$

Then the second and third lines in expression (6) are arranged to satisfy the

precedence condition as follows:

$$S=SCORE\,(1,\,5)\wedge\forall i\epsilon(2..M)\;S'=+(S,\;SCORE\,(i,\,5))\wedge AVRG=S'/M. \quad (7)$$

The first line of expression (6) is transformed to the following form by TR 1-④;

$$\forall i\epsilon(1..M)\;SCORE(i,\,5)=SCORE(i,\,2)+SCORE(i,\,3)+SCORE(i,\,4). \quad (8)$$

By searching a common factor in iteration state expressions (7) and (8), expression (8) is at first partitioned into the following two expressions:

$$\{SCORE\,(1,\,5)=SCORE\,(1,\,2)+SCORE\,(1,\,3)+SCORE\,(1,\,4)\}\,; \quad (9)$$

$$\{\forall i\epsilon(2..M)\;SCORE(i,\,5)=SCORE(i,\,2)+SCORE(i,\,3)+SCORE(i,\,4)\} \quad (10)$$

and then two iteration expressions (7) and (10) are fused. In this way, expression (6) are refined and optimized to the following form:

$\{SCORE\,(1,\,5)=SCORE\,(1,\,2)+SCORE\,(1,\,3)+SCORE\,(1,\,4)\}$;
$\{S=SCORE\,(1,\,5)\}$;
$\{\forall i\epsilon\,(2..M)\;(SCORE\,(i,\,5)=SCORE\,(i,\,2)+SCORE\,(i,\,3)+SCORE\,(i,\,4)$
$\wedge S'=+(S,\;SCORE\,(i,\,5)))\}$; $\{AVRG=S'/M\}$.

If there remains any non-primitive expression in the target language after arrangements and optimizations, the system repeats the refinement process.

5. Example

Two examples are shown in this section. The first shows an expansion of a specification and the second involves a planning process of a program construction using some TRs of linkage type.

[Example 3] Construction of a sorted score table.

The refined specifications shown by expression (10) are expanded to the following program by substituting M and 5 for m and n in TR 1-③ respectively:

$S:=SCORE\,(1,\,5)$;
for $i:=2$ *to* M *do*
  $SCORE\,(i,\,5):=SCORE\,(i,\,2)+SCORE\,(i,\,3)+SCORE\,(i,\,4)$;
  $S':=S+SCORE\,(i,\,5)$ *od*;
$AVRG:=S/M$.

The part of a sorting program is constructed from the second state expression of [Example 1] by the aids of TR 2-③, 2-② and 2-①, and substitutions $Y(1..m,\,1..n)\leftarrow SCORE\,(1..M,\,1..5)$ and $f(Y(i,\,1..n))\leftarrow SCORE\,(i,\,5)$ as follows:

for $i:=1$ *to* M *do for* $j:=i+1$ *to* M *do*
  if $\neg SCORE(i,\,5)\geqq SCORE\,(j,\,5)$ *then*
    $(SCORE\,(i,\,1...5),\;SCORE\,(j,\,1...5))$
      $:=exch\;(SCORE\,(i,\,1...5),\;SCORE\,(j,\,1...5))$ *od od.*

An expansion of assignment statement of "*exch*" by TRs $\{(x',y')=exch\,(x,y)\}$ $(s:t=x\wedge x'=y\wedge y=t)$, TR 1-④ and TR 1-① yields the following remaining program part:

for $i1:=1$ *to* 5
  do $t(i1):=SCORE\,(i, i1)$ *od*;
for $i1:=1$ *to* 5
  do $SCORE\,(i, i1):=SCORE\,(j, i1)$ *od*;
for $i1:=1$ *to* 5
  do $SCORE\,(j, i1):=t(i1)$ *od*.

[Example 4] Conversion to a unit matrix.

This example shows a construction of a solving program of a simultaneous linear equation in terms of an array representation $(1..k, 1..k+1)$, where the first column denotes a constant vector corresponding to the 0th degree coefficients of variables and each pivot is assumed to be non-zero. The program part of the given specifications is

$$\{Matrix(A(1..k, 1..k+1))\};\\ \{Partialunit(A(1..k, 1..k+1), 1..k)\} \tag{11}$$

An application of TR 3–m4, which is found by using a tree-like index file of linear algebra, to expression (11) yields

$$\{Matrix(A(1..k, 1..k+1)\};\\ \{Partialtriangular(A(1..k, 1..k+1), 1..k)\};\\ \{Partialunit(A(1..k, 1..k+1), 1..k)\}. \tag{12}$$

Subsequently, it is found that TR 3–m3 links the second state of expression (12) with the first state and a linkage at the higher level is completed. The second state expression contained in expression (12) is found to be an applicable form of TR 1–⑥ of mathematical induction and the postulate of the induction is found to be assured by TR 3–m1, then the global transition by TR 3–m3 is refined to an iterative transition by TR 3–m1 after the substitution of k for n in $S(n)$ as follows:

$$\{Matrix(A(1..k, 1..k+1))\};\ i:=k;\ \textit{while}\ i>1\ \textit{do}\\ \{Partialtriangular(A(1..k, 1..k+1), i..k)\};\\ \{Partialtriangular(A(1..k, 1..k+1), i-1..k)\};\ i:=i-1\ \textit{od};\\ [Partialtriangular(A(1..k, 1..k+1), 1..k)]. \tag{13}$$

By a similar way to the above, the global transition by TR 3–m4 in expression (12) is also refined by the TR 3–m2 and 1–⑦ as follows:

$$[Partialtriangular(A(1..k, 1..k+1), 1..k)];\\ i:=1;\ \textit{while}\ i\ k\ \textit{do}\ \{Partialunit(A(1..k, 1..k+1), 1..i)\};\\ \{Partialunit(A(1..k, 1..k+1), 1..i+1)\};\ i:=i+1\ \textit{od};\\ \{Partialunit(A(1..k, 1..k+1), 1..k)\}. \tag{14}$$

The replacement of the transitions in the while-statement contained in expressions (13) and (14) by the OPs of TR 3–m1 and 3–m2 yields the following procedure:

```
i:=k; while i>1 do
  for l:=1 to i do for j:=1 to i+1 do
    if l=i then A(l, j):=A(l, j)/A(l, l+1)
           else A(l, j):=A(l, j)−A(l, i+1)*A(i, j)/A(i, i+1)
        od od;
      i:=i−1 od;
i:=1; while i<k do
  for l:=1 to i do for j:=1 to i+1 do
    A(i+1, j):=A(i+1, j)−A(l, j)*A(i+1, l) od od; i:=i+1 od
```

6. Conclusion

An experimantal system is constructed on a LISP system by using 33 *kcells*. The required time for generation of small programs including the program of [Example 3] is several seconds for each.

The future problems of this project are to increase the flexibility of specifications by introducing various specification forms and to apply this system to generations of system programs like compilers.

References

1) T. Pietrzykowski, Journal of ACM, **20**, 333 (1973).
2) Z. Manna and R. Waldinger, Communications of ACM, **14**, 151 (1971).
3) Y. Fujita and F. Nishida, IECE Trans., **J-61D**, 103 (1978).
4) J. T. Schwartz, Computer Languages, **1**, 161 (1975).
5) D. R. Barstow, Artificial Intelligence, **12**, 73 (1979).
6) F. Nishida and Y. Fujita, Proc. of IJCAI-**79**, 659 (1979).
7) H. Kuroki, F. Nishida and Y. Fujita, Proc. of **23**-th Conference of IPSJ, 399 (1981).