

REALIZING AN EQUATIONAL SPECIFICATION

by

A. Pnueli, R. Zarhi

Department of Applied Mathematics

The Weizmann Institute of Science

Rehovot 76100, Israel

1. INTRODUCTION

In this work we address ourselves to the problem of translating an equational specification into an equivalent working program.

An equational specification is given by a set of equations of the form:

$$A_1[I_1, \dots, I_{d_1}] = f_1(A_1[E_1, \dots, E_{d_1}], \dots, A_m[E_1^S, \dots, E_{d_m}^S])$$

$$\vdots$$

$$A_m[I_1, \dots, I_{d_m}] = f_m(A_1[E_1^P, \dots, E_{d_1}^P], \dots, A_m[E_1^t, \dots, E_{d_m}^t]),$$

where I_1, \dots, I_{d_j} are free indices ranging over the nonnegative integers and E_j^i are integer valued subscript expressions. This set of equations defines therefore (recursively) the values of the elements of the possibly infinite arrays A_1, \dots, A_m .

Our interest in this problem stems from the thesis that an equational specification represents a higher level, non-procedural statement of the problem as compared to a program in a procedural language such as PL/I, Pascal, etc. for the same task.

This approach is taken by several non-procedural assignment-less high level languages such as the family of Data flow languages [AG], LUCID [AW] and MODEL [PPS]. On the other hand it seems a special case of a more general functional definition as is present in functional languages such as LISP, POP, etc.

The main task of a compiler for such a language is to analyze a specification for being consistent and complete, and construct a program in a more conventional language for the computation of the values of the array elements as defined by the specification. The algorithms described here were developed as part of a compiling processor for the language MODEL. Closely related questions have to be solved for any compiler of a non-procedural language, and in fact, our results form an extension of the algorithms developed for the compilation of LUCID programs [HOF].

There are several reasons for restricting our recursive equations to arrays i.e. functions over the natural numbers and imposing further limitations on the forms that the subscript expressions may assume.

The areas of applications that the MODEL system addresses are data processing and economic modelling. In both these areas, arrays and structures in general play an important role and the direct specification of relations between

input and output structures seems both natural and adequate.

The restrictions imposed on the syntax of a specification enable us to perform a much deeper analysis of the specification and derive a conclusive result which is presented here. In one sense our results can be interpreted as another case of recursion removal since a recursive definition is replaced by an equivalent iterative program. Moreover the terms we allow on the right are definitely nonlinear which in the general case implies the impossibility of such a translation. Consequently the restriction to integer arguments of restricted form introduces another family of recursive functions for which recursion elimination is feasible and effective.

A preliminary investigation of translatability of equational specification is reported in [PPZ]. There, all the right-hand subscript expressions were restricted to the form $I - c$ with $c \geq 0$. The results presented there gave sufficient conditions for translatability.

In this work we extend the framework by allowing much more general subscript expressions. The results presented here give a set of sufficient and necessary conditions for translatability.

The framework in which we study the problem of translatability is schematic and somewhat abstract in order to ensure solutions which do not depend on specific properties of particular functions.

A strongly related, even though differently motivated work is the one presented in [KMW]. They are interested in the question of whether recurrence relations arising in differential and difference equations can be scheduled to yield an orderly evaluation scheme of the elements defined. Based on the alternative theorem for linear equalities, they also conclude that an evaluation is possible iff the recurrence relations are noncircular. However they did not investigate the existence of a compact loop program for carrying out the computation in a reasonable way, a problem which is the main theme of our work.

Equational Specifications - Syntax and Semantics

A specification is given by a set of m simultaneously recursive equations for the array variables A_1, \dots, A_m of the form:

$$\begin{aligned} A_1[I_1, \dots, I_{d_1}] &= F_1(\dots I_j, \dots, A_1[E_1^l, \dots, E_{d_1}^l] \dots A_m[E_1^s, \dots, E_{d_m}^s]) \\ &\vdots \\ A_m[I_1, \dots, I_{d_m}] &= F_m(\dots I_\ell, \dots, A_1[E_1^t, \dots, E_{d_1}^t] \dots A_m[E_1^u, \dots, E_{d_m}^u]) \end{aligned}$$

The variables I_1, \dots, I_{d_i} are called free indices and range over the nonnegative integers. The E_j^i are subscript expressions which assume the following forms:

$$\begin{aligned} I_\ell \pm c \quad c \geq 0 \quad \text{where } I_\ell \text{ is a free index in the equation in which the} \\ E = \{ \text{expression appears} \\ g(I_1, \dots, I_{d_t}, A_1[E_1^l, \dots, E_{d_1}^l], \dots, A_m[E_1^s, \dots, E_{d_m}^s]) \}. \end{aligned}$$

The F_i 's are function symbols standing for functions into the domain over which the A_i 's vary.

The g 's appearing in subscript expressions are function symbols standing for integer functions. Their arguments, as shown in the definition, may be free indices as well as subscripted variables.

The free indices which appear on the left-hand side are assumed to be all distinct.

Following is an example of a schematic representation of Ackerman's function as a specification:

$$S_1: A[I, J] = F(I, J, A[I-1, J+1], A[I-1, g(A[I, J-1])]).$$

An interpretation I for a specification consists of:

1. A domain D over which the array elements will vary.
2. An assignment of concrete D functions to the symbols $\{F_i\}$, i.e. $I[F_i]: (Z)^{i \times (D^+)^k} \rightarrow D^+$ where F_i accepts d_i integer arguments and k, D arguments. Here $D^+ = D \cup \{1\}$
3. An assignment of concrete integer valued functions to the symbols $\{g_j\}$, i.e. $I[g_j]: (Z^+)^{d_j \times (D^+)^l} \rightarrow Z^+$.

Computability: Given a specification S and an interpretation I , a computation of S under I is a finite sequence of instances of the equations of S

$$\begin{array}{l} 1 \\ \vdots \\ m \quad A_i[v_1, \dots, v_{d_i}] = F_i \begin{bmatrix} I_1 \dots I_{d_i} \\ v_1 \dots v_{d_i} \end{bmatrix} \\ \vdots \\ n \end{array}$$

Here in the m -th line we have an instance of the equation for A_i obtained by substituting nonnegative integers v_1, \dots, v_{d_i} for the free indices, I_1 to I_{d_i} .

After substitution we evaluate the right-hand side as follows:

1. Replace every $u \neq c$ by its value (u being an integer).
2. Replace every F_i or g_i all of whose arguments are known, by its value.
3. Replace every $A[\dots u, \dots]$ with $u < 0$ or $A[\dots 1 \dots]$ by 1 .
4. Replace every $A[u_1, \dots, u_d]$, $u_1, \dots, u_d \geq 0$ which appears on the left-hand side of an earlier line, by its value.

A sequence is called a computation if every element $A[u_1, \dots, u_d]$, $u_1, \dots, u_d \geq 0$ arising in the evaluation of line m appears on the left-hand side of an earlier line.

Consider the following interpretation for S_1 :

```

D -- Nonegative integers,
F(I,J,d1,d2) =
    if I=0 then J+1
    else if J=0 then d1 else d2
g(d)=d .

```

Under this interpretation S_1 is identical to the definition of Ackerman's function. Below is a computation of S_1 under this interpretation:

1. $A(0,1) = 2$
2. $A(0,2) = 3$
3. $A(0,3) = 4$
4. $A(1,0) = A(0,1) = 2$
5. $A(1,1) = A(0,A(1,0)) = A(0,2) = 3$
6. $A(1,2) = A(0,A(1,1)) = A(0,3) = 4$

A computation under I whose last line is $A[v_1, \dots, v_d] = \delta \in D^+$ is called an I -computation for $A[v_1, \dots, v_d]$. We write $\text{Comp}\langle S, I, A[v_1, \dots, v_d] \rangle = \delta$. If there is not I -computation for $A[v_1, \dots, v_d]$ we write $\text{Comp}\langle S, I, A[v_1, \dots, v_d] \rangle = \perp$. An element $A[v_1, \dots, v_d]$ is said to be computable under I if it has an I -computation.

$A[v_1, \dots, v_d]$ is said to be computable if it is computable under every interpretation.

A specification is said to be complete if every element $A_i[v_1, \dots, v_{d_i}]$ for every $i = 1, \dots, m$ and $v_1, \dots, v_{d_i} \geq 0$ is computable.

The following specification is incomplete:

$$A[I] = F(A[I+1]) .$$

Loop Programs

Having discussed the source language for our translation, namely specifications, we proceed to define our object language.

A loop program is recursively defined as a list of statements, where a statement may be of the following types:

1. Assignment statement --

$$A[E_1, \dots, E_d] : = F(\dots L \dots B[G_1, \dots G_s] \dots)$$

where E_1, \dots, E_d , G_1, \dots, G_s are subscript expressions which may be nonnegative constants, L is a loop variable.

2. For loop --

FOR $I \geq E_1$ [TO E_2] DO

$S_1; S_2; \dots S_n$

END I

where S_1, \dots, S_n are statements, E_1 and E_2 are integer expressions. which may

only depend on loop variables. The upper limit 'TO E_2 ' is optional and when it is absent we refer to this as an infinite loop.

The only variables in the program are array variables and loop variables which may be combined in expressions. Following is an example of a loop program:

```

P1: FOR J ≥ 0 DO
    A[0,J]: =F(0,J,A[-1,J+1],A[-1,g(A[0,J-1])])
END J ;
FOR I ≥ 1 DO
    A[I,0]: =F(I,0,A[I-1,1],A[I-1,g(A[I,-1])]) ;
FOR J ≥ 1 DO
    A[I,J]: =F(I,J,A[I-1,J+1],A[I-1,g(A[I,J-1])])
END J ;
END I ;

```

An interpretation I of a loop program consists as before of a domain D and assignment to the function symbols $\{F_i\}$ and $\{g_i\}$.

Given an interpreted program P^I , we define the N-truncation of P^I , P_N^I as the program obtained by replacing every finite loop by an N-bounded loop:

FOR $I \geq E_1$ DO ... \Rightarrow FOR $I \geq E_1$ TO N DO .

P_N^I can be executed in a conventional way where expressions are evaluated using the following rules:

1. The initial values of all $A[v_1, \dots, v_d] = 1$.
2. Every $A[\dots u \dots]$ for $u < 0$ or $A[\dots 1 \dots]$ is evaluated to 1.

Obviously the execution of P_N^I must terminate since the loop bound expressions depend only on constants and other loop variables. We denote the value of $A[v_1, \dots, v_d]$ after P_N^I has terminated by

$$\text{Val} \langle P_N^I, I, A[v_1, \dots, v_d] \rangle > \quad (\text{for } v_1, \dots, v_d \geq 0) .$$

Obviously for $N \leq N'$ $\text{Val} \langle P_N^I, I, A[v_1, \dots, v_d] \rangle \subseteq \text{Val} \langle P_{N'}^I, I, A[v_1, \dots, v_d] \rangle$ where \subseteq is the partial order over D^+ by which

$$a \subseteq b \iff a = 1 \text{ or } a = b .$$

We may thus define

$$\text{Val} \langle P^I, I, A[v_1, \dots, v_d] \rangle = \text{lub}_N \text{Val} \langle P_N^I, I, A[v_1, \dots, v_d] \rangle .$$

That is, the value of an element computed by the infinite program is defined to be the least upper bound of the sequence (chain) of values computed by the N-truncations of the program.

A program P is said to realize a specification S if they are defined over the same array variables, and for every common interpretation I

$$\text{Comp} \langle S, I, A[v_1, \dots, v_d] \rangle = \text{Val} \langle P^I, I, A[v_1, \dots, v_d] \rangle$$

for every element $A[v_1, \dots, v_d]$, $v_1, \dots, v_d \geq 0$.

The basic question investigated in this paper is : Given a specification,

does there exist a loop program realizing it?

To this question we give an affirmative answer, i.e. every specification can be realized by a program. Furthermore, we give an algorithm for constructing the program which realizes the given specification.

Without loss of generality we will deal only with complete specifications.

Specifications in Normal Form

A further restriction placed on the form of admissible specification is:

Every two subscript expressions of the form $I_j \pm c$, $I_k \pm d$ appearing in different subscript positions of the same variable instance must be disjoint, i.e. $j \neq k$.

A specification is in normal form if:

1. All variables have the same dimension, d .
2. Every subscript expression of the form $I_k \pm c$ appearing in subscript position j of a variable must satisfy $k = j$.

The important point in 2. is that the same free index occupies the same position in all the variables in an equation.

Claim: Every specification can be brought to normal form.

In every specification which does not satisfy 1. we can extend the dimensions of every variable to a common maximum d . The extension is performed by adding extra dimensions which are arbitrarily subscripted by free indices not occupying the previous positions.

To handle inconsistent subscripting consider the following specification:

$$A[I, J] = F(I, J, A[I-1, J], A[J-1, I]) .$$

Define a new array B by

$$B[I, J] = A[J, I] .$$

Then the following is a consistently subscripted specification which extends the original one.

$$A[I, J] = F(I, J, A[I-1, J], B[I, J-1])$$

$$B[I, J] = F(J, I, B[I, J-1], A[I-1, J]) .$$

This transformation can be applied to the general case to produce a subscript consistent specifications.

Representation of a Specification by a Dependency Graph

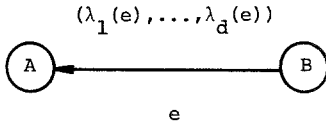
A key concept in the analysis of specifications is the dependency graph. For a given specification S we define the dependency graph $G_S = (V_S, E_S, \lambda_S)$ which is an edge labelled directed graph as follows:

V_S -- The set of nodes, having a node for each array variable.

For every dependency of the form:

$$A[I_1, \dots, I_d] = F(\dots B[E_1, \dots, E_d] \dots) .$$

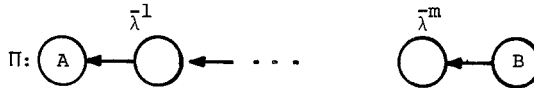
We draw a labelled edge e



The label $\bar{\lambda}(e) = (\lambda_1(e), \dots, \lambda_d(e))$ is defined by

$$\lambda_k(e) = \begin{cases} -c & \text{if } E_k = I_k + c \\ -\infty & \text{if } E_k = g(\dots) \end{cases}$$

Clearly a path Π in the graph



with $\bar{\lambda}(\Pi) = \bar{\lambda}^{-1} + \dots + \bar{\lambda}^m$ and $\lambda(\Pi) = (\ell_1, \dots, \ell_d)$ represents a dependency:

$$A[I_1, \dots, I_d] \leftarrow B[I_1 - \ell_1, \dots, I_d - \ell_d]$$

which is implied by the specification. A $-\infty$ component implies a dependency on an arbitrary higher value of a subscript.

In particular if Π is a cycle, $\bar{\lambda}(\Pi)$ represents a self dependency. A cycle C is called nonpositive if $\lambda_i(c) \leq 0$ for $i = 1, \dots, d$.

Claim: The existence of a nonpositive cycle implies an incomplete specification.

A nonpositive cycle implies a self dependency which can be extended into an infinite chain:

$$A[\bar{I}] \leftarrow A[\bar{I} - \bar{\lambda}] \leftarrow A[\bar{I} - 2\bar{\lambda}] \dots$$

This claim prevents any computation for $A[0, 0, \dots, 0]$.

Realizing by Scheduling

Our approach to constructing a program which realizes a specification is by scheduling. In this approach we attempt to construct a program out of the equations of the specification which are properly sequenced and combined into an appropriate loop structure, where we clearly must have a loop variable identified with each free index. The constraints in sequencing the equations as assignment statement are the dependency constraints represented in the dependency graph. An equation computing the value of an array element must be executed prior to an equation which needs this value.

As a first stage in the construction we decompose the dependency graph $G(s)$ into its maximal strongly connected components. Furthermore it is possible to sort these components into a linear order:

$$G(s) = G_1, G_2, \dots, G_p$$

such that any edge from G_i to G_j must satisfy $i \leq j$.

Let us denote the program realizing a specification represented by a dependency graph G by $P(G)$. Then if we have realizing program $P(G_1), \dots, P(G_p)$, then a realizing program for G is:

$$P(G) = P(G_1); \dots; P(G_p) .$$

This is so because every value needed in G_j which is not defined in G_j is defined in G_i , $i < j$ and is already computed by $P(G_i)$.

Consequently it is sufficient to consider strongly connected specifications.

A Positive Edge Algorithm

Assume that one subscript position, the first say, is such that $\lambda_1(e) \geq 0$ for all edges in G . Then, we claim a program for G may be derived which is of the form:

```
FOR  $I_1 \geq 0$  DO
   $P(G')$ 
END  $I_1$ 
```

G' is a modified form of G obtained by deleting from G all edges e such that $\lambda_1(e) > 0$, and then retaining on the remaining edges the labels $\lambda_2, \lambda_3, \dots, \lambda_d$.

Let us explain the reason for this modification. By enclosing the program in a loop on I_1 we have decided to compute all the elements defined in G by layers corresponding to ascending values of I_1 . The fact that $\lambda_1(e) \geq 0$ for all edges ensured us that no element of an I_1 -th layer may depend on an element from $I_1 + c$, $c > 0$ layer. This implies that if the elements are computable, they are computable in order of increasing I_1 layers. On the other hand, any dependency of the form $A[I_1, \dots] \leftarrow \dots B[I_1 - c, \dots]$, $\dots, c > 0$ which corresponds to an edge e with $\lambda_1(e) = c > 0$, will be automatically fulfilled by an ascending I_1 loop so that we may drop it from the graph and not consider it any longer.

The modified graph G' may now be further decomposable into strongly connected components which we may proceed to analyze and program.

This leads to the following algorithm:

Algorithm E - Construct a program $P(G)$ for a given specification G .

1. Decompose G into SCC (strongly connected components) topologically sorted G_1, \dots, G_p . Apply the following steps to construct $P(G_1), \dots, P(G_p)$ and then combine into $P(G) = P(G_1); \dots; P(G_p)$.
2. Here we are presented with a strongly connected component which for convenience we will call G . Locate a position i , $1 \leq i \leq d$ such that for all edges e in G , $\lambda_i(e) \geq 0$.
3. Construct the modified graph G^i obtained from G by deleting all edges e with $\lambda_i(e) > 0$.
4. Construct the program:

```

FOR  $I_i \geq 0$  DO
  P( $G^i$ )
END  $I_i$ 

```

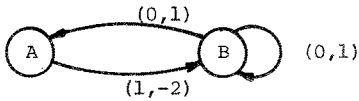
In order to obtain $P(G^i)$ we apply algorithm E recursively to G^i . Consider the specification:

```

A[I,J] = F(B[I,J-1])
B[I,J] = H(B[I,J-1], A[I-1,J+2]).

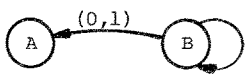


```

Its dependency graph is given by



The steps in the construction are:

```

FOR  $I \geq 0$  DO
  P {  } ;
END I
  ↓
FOR  $I \geq 0$  DO
  P {  } ;
  P {  } ;
END I
  ↓
FOR  $I \geq 0$  DO
  FOR  $J \geq 0$  DO
    B[I,J]: = H(B[I,J-1], A[I-1,J+2])
  END J ;
  FOR  $J \geq 0$  DO
    A[I,J]: = F(B[I,J-1])
  END J ;
END I ;

```

A vector $(\lambda_1, \dots, \lambda_d)$ is defined to be lexicographically positive if for some i , $1 \leq i \leq d$, $\lambda_1 = \dots = \lambda_{i-1} = 0$ and $\lambda_i > 0$.

The edge matrix of a dependency graph G is the matrix whose rows are all the vectors $\bar{\lambda}(e)$ for all the edges e in G ,

$$E(G) = \begin{bmatrix} \bar{\lambda}(e_1) \\ \vdots \\ \bar{\lambda}(e_k) \end{bmatrix}.$$

Thus we obtain:

Theorem 1: A specification S whose edge matrix is lexicographically positive, i.e. all rows are lex-positive, is realizable.

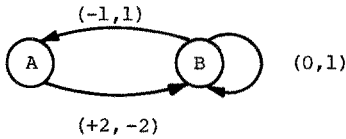
This is easily extended to a specification such that a column permutation of its edge matrix is lex-positive. In fact, the condition for success of algorithm E is more general than that. It requires only the existence of a nonnegative column in each encountered strongly connected component. Once the graph has been split into components, the permutations achieving positivity may be different for the different components.

A Positive Cycle Algorithm

Consider the following specification:

$$\begin{aligned} A[I, J] &= F(B[I+1, J-1]) \\ B[I, J] &= G(B[I, J-1], A[I-2, J+2]) . \end{aligned}$$

It has the dependency graph:



Algorithm E fails to schedule this specification because each subscript position has an edge with a negative label in this position.

However by defining a transformed array

$$A[I, J] = A'[I+1, J]$$

we obtain the specification

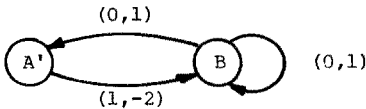
$$\begin{aligned} A'[I+1, J] &= F(B[I+1, J-1]) \\ B[I, J] &= G(B[I, J-1], A'[I-1, J+2]) . \end{aligned}$$

Now by $I' = I+1$ we get:

$$\begin{aligned} A'[I', J] &= F'(B[I', J-1]) \\ B[I, J] &= G(B[I, J-1], A'[I-1, J+2]) \end{aligned}$$

where F' is F where F is defined and 1 otherwise ($I' = 0$).

Consequently the graph representation of the transformed specification is



This specification is now lex-positive and hence schedulable. Trying to generalize the transformation above, and concentrating on the first subscript position, we look for a general transformation

$$A_i[I_1, \dots, I_d] = A'_i[I_1 + k_1^i, I_2, \dots, I_d], \quad k_1^i \geq 0.$$

The transformation should be such that the labels on all the edges of the modified graph should have nonnegative first component.

Let

$$A_i[I_1, \dots, I_d] \leftarrow \dots A_j[I_1 - \lambda_1(e), \dots]$$

be any dependency in the old graph corresponding to an edge e . In the new graph this will be transformed to

$$(I'_1 = I_1 + k_1^i)$$

$$A'_i[I'_1, \dots, I'_d] \leftarrow \dots A'_j[I'_1 - k_1^i + k_1^j - \lambda_1(e), \dots] \dots$$

Thus the new label of e is given by

$$\lambda'_1(e) = \lambda_1(e) + (k_1^i - k_1^j).$$

The k_1^i 's we are looking for should therefore satisfy

$$k_1^j - k_1^i \leq \lambda_1(e).$$

for all edges e : $A_i \leftarrow A_j$.

By [FF] a necessary and sufficient condition for the existence of such $k_1^i \geq 0$ is that for every cycle in the graph $C = e_1, e_2, \dots, e_m$, $\lambda_1(C) = \lambda_1(e_1) + \dots + \lambda_1(e_m) \geq 0$. Motivated by this we consider an extended edge matrix. Let $e_1 \dots e_m$ be the edges of G .

$$\text{Let } E(G) = \begin{bmatrix} \lambda(e_1) \\ \vdots \\ \lambda(e_m) \end{bmatrix} \quad R(G) = \begin{bmatrix} \delta(e_1) \\ \vdots \\ \delta(e_m) \end{bmatrix}$$

where $\lambda(e_i)$ is the label for edge $e_i: u_j \leftarrow u_\ell$, and $\delta(e_i)_j = 1$, $\delta(e_i)_\ell = -1$, $\delta(e_i)_t = 0 \quad t \neq j, \ell$.

We define $M_E(G)$ the extended matrix to be

$$M_E(G) = [E(G); R(G)]$$

$M_C(G)$ the cycle matrix of G is defined by

$$M_C(G) = \begin{bmatrix} \bar{\lambda}(c_1) \\ \vdots \\ \bar{\lambda}(c_s) \end{bmatrix} \quad \text{where } \bar{\lambda}(c_i) \text{ is the label of the cycle } c_i \text{ of } G, \text{ and}$$

$c_1 \dots c_s$ are all the basic cycles of G .

A specification S is called cycle-lexicographically positive (c-lex-positive for short) if the matrix $M_C(G)$ is lex-positive. By [FF] if S is c-lex-positive then there exists a t -unit-vector $v = (0 \dots 0, 1 \dots 0)$ and constants $k_t^i \geq 0 \quad 1 \leq i \leq n$, s.t.

$$M_E(G) \cdot \begin{bmatrix} \bar{v} \\ \bar{k} \end{bmatrix} \geq 0$$

Algorithm C -

1. Decompose G into a sorted list of strongly connected components: $G = G_1, \dots, G_p$. Apply the following steps to each component which for brevity, we denote by G .

2. Find a position t and nonnegative constants k_i for each array variable A_i such that for each edge e in G , $e: A_i \xrightarrow{\lambda(e)} A_j$ the following holds

$$\lambda_t(e) + k_i^i - k_j^j \geq 0$$

3. Modify the specification by substituting:

$$A_i[I_1, \dots, I_d] = A'_i[\dots, I_t + k_t^i, \dots],$$

The modified specification would now have

$$\lambda'_t(e) \geq 0$$

for every edge in the graph.

4. Construct the program

```
FOR  $I_t \geq 0$  DO
  P( $G^i$ )
END  $I_t$ 
```

$P(G^i)$ is the program constructed for the graph G^i by a recursive application of algorithm C. G^i is obtained from G by deleting all edges e such that $\lambda'_t(e) > 0$.

Theorem 2: Every specification which is cycle-lex-positive is realizable.

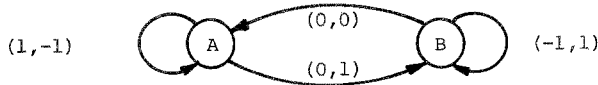
Again, the actual condition required is less restrictive than lex-positivity even after transformation.

The General Case

Consider the following specification

$$\begin{aligned} A[I, J] &= F(A[I-1, J+1], B[I, J]) \\ B[I, J] &= G(B[I+1, J-1], A[I, J-1]) \end{aligned}$$

Its graph representation is:



Here neither the first nor the second label is nonnegative for each cycle. Consequently algorithm C will fail to schedule this specification. However by the following substitution:

$$\begin{aligned} A[I, J] &= A'[I+J, I] \\ B[I, J] &= B'[I+J, J] \end{aligned}$$

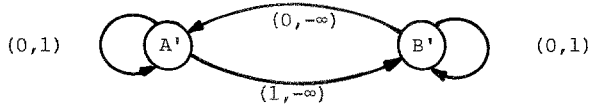
We obtain a modified specification:

$$\begin{aligned} A'[I+J, I] &= F(A'[I+J, I-1], B'[I+J, J]) \\ B'[I+J, J] &= G(B'[I+J, J-1], A'[I+J-1, I]) \end{aligned}$$

Resubstitute $I+J = I'$ and $I = J'$ in the first equation and $I+J = I'$ and $J = J'$ in the second equation to obtain

$$\begin{aligned} A'[I', J'] &= F(A'[I', J'-1], B'[I', I'-J']) \quad I' \geq J' \geq 0, \\ B'[I', J'] &= G(B'[I', J'-1], A'[I'-1, I'-J']) \quad I' \geq J' \geq 0. \end{aligned}$$

Treating the expression $I' - J'$ as $g(I', J')$ we obtain the following dependency graph:



with the extended edge matrix

$$M_E(G) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\infty & 1 & -1 \\ 1 & -\infty & -1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The new specification is c -lex-positive and therefore schedulable by Algorithm C.

Generalizing this example we will look for a linear $d \times d$ transformation T^i , one for each variable A_i such that:

- The entries $T^i_{k,\ell} \geq 0$ all are integers.
- T^i is unimodular, i.e. $\det(T^i) = \pm 1$.
- Perform the following:

- Substitute $A_i[\bar{I}] = A_i[\bar{I} \cdot T^i]$ for each variable A_i ;
- substitute $\bar{I} \cdot T^i = \bar{I}'$ in the equation for A_i ;
- substitute each dependency $A_i[\bar{I}] \leftarrow A_j[\bar{I} - \bar{\lambda}]$ by $A_i[\bar{I}'] \leftarrow A_j[\bar{I}'(T^i)^{-1} - \bar{\lambda}] T^j$

Then the resulting specification is c -lex-positive.

The transformations T^i are constructed incrementally by columns. For two variables A and B , T^A and T^B will be identical in the first j columns if A and B stayed in the same strongly connected component for the first j steps.

The basic construction step is that of finding columns $\bar{v} \geq 0$ and $\bar{k} \geq 0$ such that $M(\bar{v}, \bar{k}) \geq 0$ where M is the current extended edge matrix. In this we refer to the basic alternative theorem $[T]$ which is the theorem underlying the duality theorem for linear programming:

For a given matrix M , there either exists a nonnegative column $\bar{v} \geq 0$ such that $M\bar{v} \geq 0$ and $M\bar{v} \neq 0$ or there exists an all positive row $\bar{u} > 0$ such that $\bar{u}M \leq 0$.

We show that the second alternative implies the existence of a non-positive cycle which as shown above is impossible in a complete specification.

Suppose there exists a $\bar{u} > 0$ such that $\bar{u}M = \bar{u}[E; R] \leq 0$, in particular $\bar{u}R \leq 0$, by the definition of R the sum of it's columns is $\bar{0}$, this implies $\bar{u}R = 0$. Let v^x be a vertex of G and let $E_{\bar{I}}$ be the set of edges leading into \bar{v} and $E_{\bar{O}}$ the set of edges leading out of \bar{v} . Then $\bar{u}R = 0$ implies $\sum_{i \in E_{\bar{I}}} u_i = \sum_{i \in E_{\bar{O}}} u_i$. \bar{u} is a combination of all the edges of G ($\bar{u} > 0$) with the incoming weight of each vertex (number of incoming edges counting repetitions) equal to the outgoing weight.

This means that the combination \bar{u} of edges is a circuit in G , and $\bar{u}E \leq 0$. Thus the combination \bar{u} defines a nonpositive circuit.

Algorithm T - for scheduling a complete specification.

Let G be the dependency graph.

The algorithm operates in two phases. In the first phase we find for each array variable A_i a list of transformation vectors $L_i = (\begin{smallmatrix} z_1 \\ v_1 \end{smallmatrix}, \begin{smallmatrix} z_2 \\ v_2 \end{smallmatrix}, \dots)$. The list will be used in the second phase to construct for each array an individual unimodular nonnegative transformation such that the transformed specification will be c-lex-positive.

Initially $L_i = \emptyset$ for every variable A_i .

Phase I

Denote $G^t = G$ and repeatedly perform steps 1-4, for $t=1, 2, \dots$ until the exit condition described in step 1 is fulfilled.

1. Decompose G^t into strongly connected components:

$$G^t = C_1^t, \dots, C_{q_t}^t.$$

If all the components are singular, i.e. contain no edges then exit this phase.

Otherwise, perform steps 2, 3, 4 for each nonsingular component C_j^t .

2. Find column vectors $\bar{v} \geq 0, \bar{k} \geq 0$ such that

$$E_j^t \bar{v} + R_j^t \bar{k} \geq 0, \quad E_j^t \bar{v} + R_j^t \bar{k} \neq 0$$

where $[E_j^t, R_j^t]$ is the extended edge matrix for the nonsingular component C_j^t .

3. Append the obtained vector \bar{v} to the vector list L_i of each variable $A_i \in C_j^t$.
4. For each edge e connecting $A_i \xrightarrow{e} A_l$ where both $A_i, A_l \in C_j^t$ compute the weight:

$$w(e) = \bar{\lambda}(e) \cdot \bar{v} + k_l - k_i$$

Here $\bar{\lambda}(e)$ is the row in E_i^t corresponding to the edge e .

Delete e from $C_j^t(G^t)$ if $w(e) > 0$.

The graph obtained from G^t by the deletions performed in step 4 is G^{t+1} .

Phase II

1. For each array variable A_i use the list L_i to construct a unimodular nonnegative dxd transformation T^i (the construction is described below).
2. Substitute each dependency

$$\begin{aligned} A_i(\bar{I}) &\leftarrow A_j(\bar{I} - \bar{\lambda}) \text{ by:} \\ A^i(\bar{I}') &\leftarrow A_j((I'(T^i)^{-1} - \bar{\lambda})T^j) \end{aligned}$$

to get an equivalent C-lex-positive specification S' .

3. Apply algorithm C to S' .

Below we describe the construction of T^i (step 1 or phase II)

Basic Lemma

Let \bar{v} be a vector of length d ,

and $\gcd(v_1, \dots, v_d) = 1$, then there exists a dxd matrix M such that:

1. $\det(M) = \pm 1$.

2. the first column of M is \bar{v} .

Proof

by induction on d .

For $d=1$ - obvious.

Assuming that the lemma is true for a certain $d \geq 1$, consider $d+1$: Denote $g = \gcd(v_2, \dots, v_{d+1})$ and $\lambda_i = v_i/g$ $2 \leq i \leq d$. Obviously $\gcd(\lambda_2, \dots, \lambda_{d+1}) = 1$ and by the induction hypothesis there is a $d \times d$ unimodular matrix M_λ whose first column is $[\lambda_2, \dots, \lambda_{d+1}]^T$. $\gcd(v_2, \dots, v_{d+1}) = 1$ implies $\gcd(v_1, g) = 1$. Consequently there exist integers α, β such that

$$\alpha v_1 - \beta g = 1$$

We define M by

$$M = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & & \\ 0 & & & \end{bmatrix} \times \begin{bmatrix} v_1 & \beta & 0 & \dots & 0 \\ g & \alpha & 0 & \dots & 0 \\ 0 & & & & \\ \vdots & & & & \\ 0 & 0 & & & \end{bmatrix} \begin{bmatrix} & & & & \\ & M_\lambda & & & \\ & & & I & \\ & & & & \end{bmatrix}$$

Clearly: $\det(M) = \det(M_\lambda) \cdot (v_1 \alpha - \beta g) = \pm 1$

and the first column of M is $[v_1, g\bar{\lambda}]^T = \bar{v}$.

Consider a list L of vectors $L = (\bar{v}^1, \dots, \bar{v}^p)$ corresponding to an array variable

A. Let $\bar{k}^1 \dots \bar{k}^p$ be the corresponding vectors found in step 2. We denote by $C^1 \dots C^p$ the sequence of strongly connected components generated by the algorithm, such that C^{i+1} is a subgraph of C^i , \bar{v}^i is the transformation vector found for the component C^i and A is contained in each C^i . Associating with each C^i its extended edge matrix $[E^i, R^i]$, the \bar{v}^i 's, \bar{k}^i 's and E^i 's, R^i 's are related by

$$E^{j-i} \bar{v}^i + R^{j-i} \bar{k}^i = 0 \text{ for every } 1 \leq i < j \leq p.$$

$$\bar{v}^i, \bar{k}^i \geq 0 \quad E^{i-i} \bar{v}^i + R^{i-i} \bar{k}^i \geq 0, \quad E^{i-i} \bar{v}^i + R^{i-i} \bar{k}^i \neq 0 \text{ for } 1 \leq i \leq p$$

Our aim is to construct a unimodular nonnegative transformation $T = [\bar{u}^1, \dots, \bar{u}^p, \dots, \bar{u}^d]$ such that the \bar{u}^i 's with a corresponding $\bar{b}^1, \dots, \bar{b}^p$ satisfy the same requirements as the \bar{v}^i 's, \bar{k}^i 's above, i.e.:

$$(Q) \quad \begin{aligned} E^{j-i} \bar{u}^i + R^{j-i} \bar{b}^i &= 0 \text{ for every } 1 \leq i < j \leq p \\ E^{i-i} \bar{u}^i + R^{i-i} \bar{b}^i &\geq 0, \quad E^{i-i} \bar{u}^i + R^{i-i} \bar{b}^i \neq 0. \quad 1 \leq i \leq p \end{aligned}$$

The constructed T defines the transformation on the subscripts of all the variables that are contained in C^p .

By (Q) the modified specification constructed in step 2, phase II, of the algorithm is C-lex-positive and algorithm C can be applied.

We now describe the construction of T .

Unfortunately we cannot use the given \bar{v}^i 's as the desired \bar{u}^i 's and will have to use instead:

$$\bar{u}^i = h_{ii} \bar{v}^i + \sum_{j < i} h_{ij} \bar{v}^j \quad h_{ii} > 0 \quad i = 1, \dots, p$$

for some coefficients h_{ij} . Note that for every such combination.

1. For every cycle c in C^j , $j > i$ with the label $\bar{\lambda}(c)$, $\bar{\lambda}(c)u^i = 0$.
2. For every cycle c in C^i $\bar{\lambda}(c)u^i \geq 0$ and there exists a cycle c_0 in C^i such that $\bar{\lambda}(c_0)u^i \neq 0$.

Then by [FF] there exists $\bar{b}^i \geq 0$ such that \bar{u}^i, \bar{b}^i satisfy equations (Q). Consequently every such combination solves the system Q.

The particular h_{ij} and $\bar{u}^{p+1}, \dots, \bar{u}^d$ will be chosen so as to make $T = [\bar{u}^1 \dots \bar{u}^d]$ nonnegative and unimodular.

Without loss of generality we can preorder the subscript positions so that the sequence $\bar{v}^1, \dots, \bar{v}^p$ is graded. By this we mean that for every subsequent $\bar{v}^1, \dots, \bar{v}^t$ $1 \leq t \leq p$ there exists an $s=s(t)$ such that

For every $1 \leq i \leq t$, $j > s$, $\bar{v}_j^i = 0$

and for every $1 \leq j \leq s$ there exists an $i \leq t$ such that $\bar{v}_j^i > 0$.

This implies that in adding \bar{v}^{t+1} to the set $\bar{v}^1 \dots \bar{v}^t$ there is an $s'=s(t+1) \geq s(t)$ such that $\bar{v}_{s'+1}^{t+1} \dots \bar{v}_s^{t+1} > 0$ and $\bar{v}_k^{t+1} = 0$ for all $k > s'$.

The equations (Q) satisfied by the \bar{v}^i 's show that every subset $\bar{v}^1, \dots, \bar{v}^t$ is linearly independent. This implies that for every $t \leq p$ $s(t) \geq t$. We will construct the \bar{u}^i and T by successive construction of $T^0 = I$, $T^1, \dots, T^p = T$ each step adding a new \bar{u}^{t+1} to the already constructed $\bar{u}^1 \dots \bar{u}^t$.

Throughout the construction T^t will maintain the following properties:

- a) T^t has the following partitioned form:

$$T^t = \begin{bmatrix} X_{(s)}^t & 0 \\ 0 & I_{d-s} \end{bmatrix}$$

where $s=s(t)$ is the grade number associated with $\bar{v}^1 \dots \bar{v}^t$. $X_{(s)}^t$ is an $s \times s$ matrix. I_{d-s} is the $(d-s) \times (d-s)$ unit matrix.

- b) T^t is unimodular and nonnegative.

c) $T^t = [\bar{u}^1 \dots \bar{u}^t, \bar{u}^{t+1}, \dots, \bar{u}^d]$ where $\bar{u}^1 \dots \bar{u}^t$ satisfy (with the corresponding \bar{b}^i 's) equations (Q).

Obviously $T^0 = I$ satisfies all the above requirements for $t=0$.

Consider now the general step of going from T^t , $t < p$ to T^{t+1} . Let now $\bar{v}^{t+1} = [v_1 \dots v_t, v_{t+1}, \dots, v_d]^T$. The vector \bar{v}^{t+1} can be represented as

$$\bar{v}^{t+1} = T^t ((T^t)^{-1} \bar{v}^{t+1}) = T^t \bar{w} = \sum_{i=1}^d \bar{u}^i w_i$$

where $\bar{w} = (T^t)^{-1} \bar{v}^{t+1}$

Note that because of the partitioned structure of T^t $w_i = v_i$ for $i > s$

Define $\bar{x} = [0 \dots 0, w_{t+1}, \dots, w_d]^T / g$ where $g = \gcd(w_{t+1}, \dots, w_d)$

Let $\bar{y} = T^t \bar{x}$, then $\bar{y} = \frac{1}{g} \sum_{i=t+1}^d \bar{u}^i w_i$ and $\bar{v}^{t+1} = \sum_{i=1}^t \bar{u}^i w_i + g \bar{y}$.

Consequently for every cycle c in C^j $j > t+1$, $\bar{\lambda}(c) \bar{y} = 0$ and for every cycle c in C^{t+1} $\bar{\lambda}(c) \bar{y} = \frac{1}{g} \bar{\lambda}(c) \bar{v}^{t+1} \geq 0$ and there exists some cycle c_0 in C^{t+1} such that $\bar{\lambda}(c_0) \bar{y} \neq 0$, then by [FF] there exists a \bar{b}^{t+1} s.t. \bar{y}, \bar{b}^{t+1} satisfy equation (Q).

Note also that for $i > s$ $y_i = v_i^{t+1}/g \geq 0$. Let $s' = s(t+1)$. By the basic lemma

there exists an $(s'-t) \times (s'-t)$ unimodular matrix V whose first column is

$$\frac{1}{g} [w_{t+1} \dots w_{s'}]^T$$

Let U be:

$$U = \begin{bmatrix} V & 0 \\ 0 & I_{d-s'} \end{bmatrix} \quad \text{Obviously } U \text{ is unimodular.}$$

Consider now:

$$T' = T^t \begin{bmatrix} I_t & 0 & 0 \\ 0 & V & 0 \\ 0 & 0 & I_{d-s'} \end{bmatrix} = \begin{bmatrix} y_{(s')} & 0 \\ 0 & I_{d-s'} \end{bmatrix}$$

$$\text{Here } y_{(s')} = \begin{bmatrix} x_{(s)}^t & 0 \\ 0 & I_{s'-t} \end{bmatrix} \times \begin{bmatrix} I_t & 0 \\ 0 & V \end{bmatrix}$$

Obviously T' is unimodular.

T' has the following column structure:

$$T' = [\bar{u}^1, \dots, \bar{u}^t, \bar{y}, \bar{u}^{t+2}, \dots, \bar{u}^d]$$

for some $\bar{u}^{t+2}, \dots, \bar{u}^{t+2}, \dots, \bar{u}^d$. To show this we note first that in the transformation of T^t the first block was I_t which preserves the first t columns. The $(t+1)$ 'st column in T' is formed by multiplying T^t by the column \bar{x} , i.e. $T^t \bar{x} = \bar{y}$.

To finalize our construction of T^{t+1} we only need to ensure the nonnegativity of the elements. This is accomplished by a transformation of T' which consists of addition of the first $t+1$ columns to columns $t+1, \dots, s'$. Such a transformation is obviously expressible as a unimodular transformation which preserves the unimodularity of T^{t+1} .

We start by ensuring that the $(t+1)$ 'st column is nonnegative. For $i > s$ we have already observed that $y_i = v_i^{t+1}/g \geq 0$. For $i \leq s$ there is always some $j \leq t$ such that $u_i^j > 0$ and we add the j 'th column sufficiently many times to the $(t+1)$ 'st column until u_i^{t+1} becomes positive. After these additions $u_i^{t+1} > 0$ for every $i=1, \dots, s'$. By adding the $(t+1)$ 'st column to the subsequent columns we can now ensure a nonnegative unimodular matrix

$$T^{t+1} = \begin{bmatrix} x_{(s')}^{t+1} & 0 \\ 0 & I_{d-s'} \end{bmatrix}$$

where $T^{t+1} = [\bar{u}^1, \dots, \bar{u}^t, \bar{u}^{t+1}, \dots]$ and \bar{u}^{t+1} is a combination of the form $\bar{y} + \sum_{i=1}^t h_i \bar{u}^i$.

Since \bar{y}, \bar{u}^{t+1} satisfy equations (Q) for $i = t+1$ we can find $\bar{b}^{t+1} \geq 0$ such that $\bar{u}^{t+1}, \bar{b}^{t+1}$ satisfy equations (Q) for $t+1$.

This completes the construction step, and for $t=p$ we have the desired $T=T^p$.

Theorem 3: Every complete specification can be scheduled.

By preliminary analysis of an arbitrary specification we can detect incomplete parts which always form whole strongly connected components on which no complete part depend. A program for the complete part can be constructed by Algorithm T.

Any program of the form:

$$A[I_1, \dots, I_d] = A[I_1, \dots, I_d]$$

will do for the incomplete parts.

Therefore we have,

Theorem 4: Every specification can be realized.

Extensions and Discussion

The presented analysis completely resolves the problem of realizability of specifications which fit into the syntactic framework studied here. Many possible extensions suggest themselves as relaxation of some of the restrictions. One possible extension is to allow constant subscripts as well as having several equations for disjoint ranges of a single variable. An example of such extended specification is:

$$\begin{aligned} A[0, J] &= F(J) \\ A[I+1, 0] &= G(A[I, 1]) \\ A[I+1, J+1] &= H(A[I, G(A[I+1, j])]) \end{aligned}$$

The problem of deciding whether a given specification in the extended form is complete is undecidable. It is not difficult to see that for an arbitrary program P in a simple programming language $[M]$ we can define a specification S_P in our extended form such that:

1. P halts for each input if and only if S_P is complete;
2. P halts for input $(a_1 \dots a_d)$ if and only if $A(a_1, \dots, a_d)$ (A an array variable of S_P) is computable.

Thus it is undecidable even for a single element whether the element is computable.

Another possible extension is to allow constant subscripts only on the right hand side of the equations and to require a single equation for each array variable.

Consider for example the following specification:

$$\begin{aligned} \underline{S} \\ A[I, J] &= F(A[I+1, J-1], B[I, J]) \\ B[I, J] &= G(B[I-2, J+2], C[I-1, J]) \\ C[I, J] &= H(C[I+3, J-1], A[0, J]) \end{aligned}$$

The extended edge matrix is:

$$M_E = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 2 & -2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & -1 \\ -3 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Algorithm T will fail to find \bar{V}^1 because the sum of the rows is $[-1, 0, 0, 0, 0]$.

But it can be verified that there is no cycle.

$$A(\bar{I}_0) \leftarrow \dots \leftarrow A(\bar{I}_p)$$

such that $\bar{I}_p \geq \bar{I}_0$ and $\bar{I}_j \geq 0$ for $1 \leq j \leq p$.

One can easily see that it is also impossible to define $A(0,J)$ by another array variable to get two scc such that each component is schedulable by algorithm T.

By defining A', B', C' - transformed array variable for A, B, C :

$$\begin{aligned} A'[I+J, J] &= A[I, J] \\ B'[I+J, I] &= B[I, J] \\ C'[J, I] &= C[I, J] \end{aligned}$$

We get the following loop program for S:

```

FOR I' ≥ 0 DO
  FOR J' ≥ 0 TO I' DO
    B[I', J'] = G(B'[I', J'-2] , C'[I'-J', J'-1])
  END J';
  FOR J' ≥ 0 TO I' DO
    A'[I', J'] = F(A'[I', J'-1] , B'[I', I'-J'])
  END J';
  FOR J' ≥ 0 DO
    C'[I', J'] = H(C[I'-1, J'+3] , A'[I', I'])
  END J';
END I';

```

The example shows that allowing constant subscripts on the right hand side of the equations makes the scheduling problem much more difficult, and that the criteria and methods presented here are inadequate to deal with such specifications.

The algorithms presented here will be incorporated in the next version of the MODEL translator system.

REFERENCES

- [AG] Arvind, Gostelow, "Dataflow computer Architecture: Research and Goals", Technical Report No. 113, Department of CIS, University of California, Irvine.
- [AW] Ashcroft, E.A. and Wadge, W., "LUCID, A nonprocedural Language with Iterations", CACM 20, No. 7, pp. 519,526.
- [EF] Ford, I.R. and Fulkerson, D.R., Flows in Networks, Princeton University Press, 1962.
- [HOF] Hoffman, C.M., "Design and Correctness of a Compiler for a Nonprocedural Language", Acta Informatica 9, pp. 217-241, 1978.
- [KMW] Karp, M.R., Miller, E.R., Winograd, S., "The Organization of Computations for Uniform Recurrence Equations", ACM Journal, Vol, 14, No. 3, 1967, pp. 563-590.
- [M] Minsky, M.L., "Computation Finite and Infinite Machines", Prentice-Hall, 1967.
- [PMM] Prywes, N.S., "Model II - Automatic Program Generator, User Manual", Office of Planning and Research, Internal Revenue Service, TIR-77-41, July 1978., Available from CIS Department, University of Pennsylvania.
- [PPS] Prywes, N.S., Pnueli, A. and Shastri, S., "Use of a Nonprocedural Specification Language and Associated Program Generator in Software Development", to appear in ACM TOPLAS.
- [PPZ] Pnueli, A., Prywes, N. and Zarhi, R., "Scheduling an Equational Specification", International Workshop on Program Construction, Chateau de Bonas, France, September 1980; (ed. Inria).
- [SHA] Shastri, S., Pnueli, A. and Prywes, N.S., "Basic Algorithms Used in the MODEL System for Design of Programs", Moore School Report, CIS Department, University of Pennsylvania.
- [T] Tucker, A.W., "Dual Systems of Homogeneous Linear Relations", Linear Inequalities and Related Systems, H.W. Kuhn and A.W. Tucker, Princeton University Press, 1956, pp. 3-18.
- [WAD] Wadge, W., "An Extentional Treatment of Dataflow Deadlocks", Proceedings: Semantics of Concurrent Computation, Evian, France, 1979, Springer-Verlag.