CERN - DATA HANDLING DIVISION
DD/74/20
I.M. Willers
July 1974

The syntax directed graph algorithm for the input of equations to the Taylor series system for solving ordinary differential equations.

Introduction

In Barton, Willers and Zahar [1] it is described how the Taylor series method for the solution of the initial value problem of ordinary differential equations may be implemented as a completely automatic procedure.

In this paper an alternative description of the compiling process and code generation stages of the implementation are given. As well as being easier to describe and to understand this new description leads to a simpler implementation of the algorithm. The numerical aspects are covered in Barton, Willers and Zahar 1970 [2]. The basic method is described in R.E. Moore 1966 [3] and outlined in the two papers by Barton, Willers and Zahar. In short, the unknown variables of the equations are replaced by truncated Taylor series. A recurrence relation between the Taylor series coefficients is established using the equations themselves. The equations are split up into a set of simple equations which involve one arithmetic operation and at least one unknown variable which may have been introduced at this stage. Each new variable has a Taylor series associated with it. Each one of these simple equations corresponds to a relationship between the respective Taylor coefficients. This relationship depends upon the arithmetic operation and may be simply stated. These simple relationships when ordered produce the recurrence relationship. The recurrence relationships and the equation's initial conditions may be used to obtain a normal truncated power series solution to the problem. Using suitable numerical control the dependent variables may now be evaluated at another point hence giving new initial conditions and the process repeated to give a step-by-step integration procedure.

This paper introduces the concept of a "Syntax Directed Graph" (S.D.G. for short) which is a generalisation of the syntax tree normally used when compiling expressions, see Graham 1964 [4].

The S.D.G. structure is utilised to establish that a problem may be well posed and secondly for the generation of the recurrence relationship in the form of code.

1. Construction of the Syntax Directed Graph

In the next two paragraphs a mapping is described which establishes the abstract directed graph. This is immediately replaced by the geometric directed graph which is more intuitively obvious.

Let X be a set whose elements are arithmetic operators so that, for example,

$$X = \left[\phi, +, -, /, \uparrow, ', *, \sin, \cos, \int \right]$$

where ϕ is the null operator. Let V be a set which contains X an infinite number of times. Let A be the set of all real numbers, the variables introduced by a problem statement, the independent variable and a set of variables, t_i, which can be used when required.

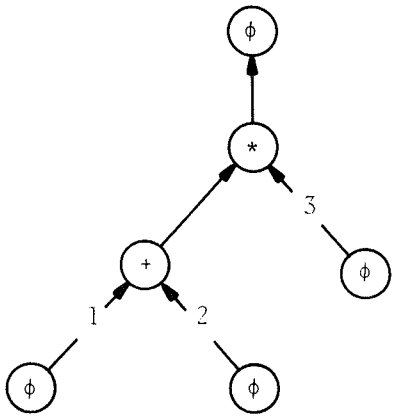
Let there be mappings of the set A onto the set V x V, so that a variable is the result of one operation and is used directly by the other operation. This is not a symmetric relationship therefore we associate an ordering with each pair of elements taken from V x V, and each pair is named by the element of A which maps onto that pair. Consider the

expression (1 + 2) * 3. We introduce the null operators ϕ_1, ϕ_2, ϕ_3, and ϕ_4. Now choosing the first operator so that the variable is the result of that operation it is possible to map

1 onto $[\phi_1, +]$, 2 onto $[\phi_2, +]$, 3 onto $[\phi_3, *]$, t_1 onto $[+, *]$

and t_2 onto $[*, \phi_4]$.

This mapping can be expressed in terms of a geometric graph. Let elements of A represent the arcs which are directed from the first towards the second element of the pairs chosen from V x V. The above example can be drawn as follows

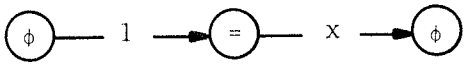


This structure is directly analogous to a syntax tree the construction of which is well understood, see Graham 1964 [4].

However in the Taylor series system the objective is to input sets of equations. Let the equals sign, =, be added to the set, X, and consider the equation

$$x = 1$$

mapping this onto an S.D.G gives

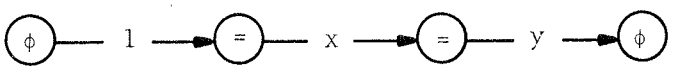


We can add another equation to the set

$$x = 1$$

$$y = x$$

which maps onto

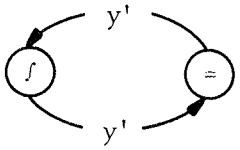


Thus two equations have mapped onto a single S.D.G. and the relationship between the two equations is clearly indicated. The process of adding an equation to a set may be seen to be a simple process which may involve the deletion of null vertices.

Consider the differential equation:

$$y' = y$$

This maps onto the S.D.G.



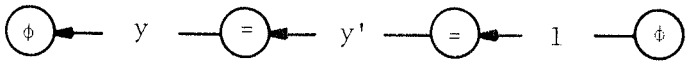
This introduces the interesting occurrence of a loop in the S.D.G. The structure of a syntax tree is completely lost and the relationship expressed in the equation cannot be expressed in a syntax tree. In this way a problem statement for the Taylor series system may be mapped onto two sets of S.D.G.'s, one set representing the differential system and the other set the initial values.

Consider the following three examples:

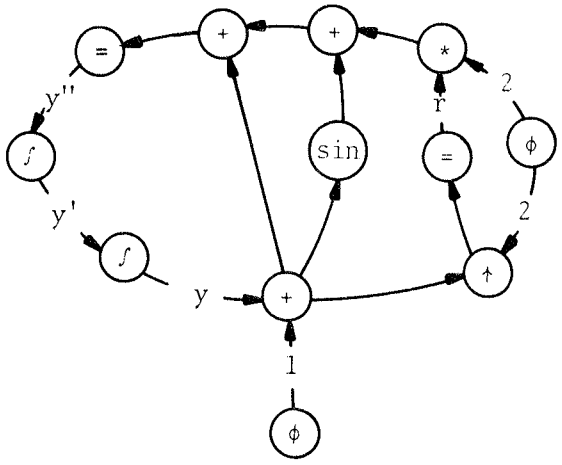
Example 1:

```
Integrate
y'' = 2 * r + sin (y + 1) + 1 + y
r = (1 + y) ↑ 2
With initial conditions
y = y'
y' = 1
```

The initial conditions map onto



The two equations form one connected S.D.G. which indicates how one of the equations must be used before solving the other. The differential equation and the identity map onto



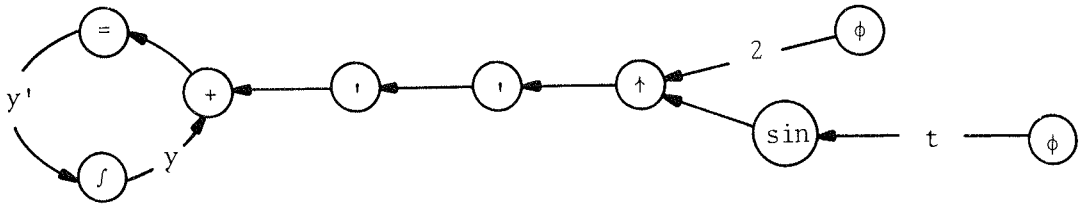
The S.D.G.'s can be optimised by a straightforward comparison of vertices and this process is made easier, as far as implementation is concerned, if the variables, or arcs, pointing into the vertices are ordered. The removal of common subexpressions is clearly demonstrated in these examples.

Each prime that appears on the left hand side of the equation leads to an integration of the right hand side. The looping properties of the S.D.G. are clearly shown.

Example 2:

Integrate
$y' = y + (\sin^2 t)''$
With initial conditions
$y = 1$
$t = 0$

The S.D.G. for the differential equation is

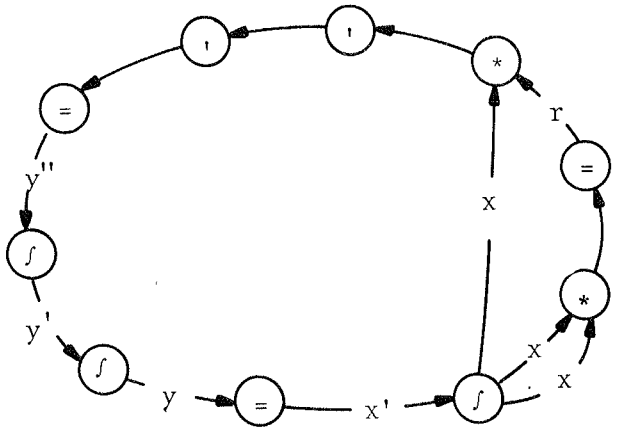


This shows a tree structure which joins a small loop. A well defined tree structure contains variables about which all information is known and it is possible to evaluate fully any such tree structure independently for a well defined set of equations.

Example 3:

Integrate
$y'' = (x * r)''$
$r = x * x$
$x' = y$
With initial conditions
$y = 1$
$y' = 0$
$x = 1$
$t = 0$

The S.D.G. for the differential equation set is



This example gives a complicated looping structure. There is no tree type structure present and nothing is known about the variables represented by the arcs. This S.D.G. indicates the close connection between the equations in a way which is not possible using a conventional syntax tree.

2. The Compiling Algorithm

One task of this algorithm is to determine whether a problem statement may be well defined for the Taylor series algorithm.

A typical problem statement has the form:

Integrate

$$y_i^{(n_i)} = f_i\ (t,\ y_1,\ y_2,\ldots,y_m) \qquad 1 \leq i \leq m$$

With initial conditions

$$y_1^{(j)} = h_i^j\ (t,\ y_1,\ y_2,\ldots y_m) \qquad 1 \leq i \leq m \text{ and } 0 \leq j \leq n_i$$

$$t = t_o$$

where (j) and (n_i) represent a non negative number of differentiations and t is the independent variable.

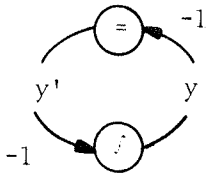
We express the two sets of equations as two sets of S.D.G.'s and define a labelling of the S.D.G. which enables the algorithm to be applied. Each variable is represented as a truncated Taylor series and each operation is a relationship acting on these series producing a single result which is a coefficient of a series. Consider such an operation on j Taylor series where the i-th Taylor series is known up to and including the coefficient of order m_i. Let this operation produce the coefficient of order n. Then there exists a general relationship of the form:

$$c \leq n \leq \min\ (m_1,\ m_2,\ldots m_j) + c \qquad (1)$$

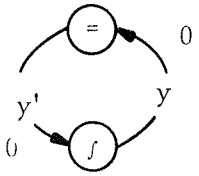
where c is a function of the operation so that $c = 1$ for $\int$, $c = -1$ for ' and $c = 0$ otherwise. For example if one wanted to add two series together and one series is known up to order 3 while the other is known to order 5 then the sum of these two series can be calculated up to order 3. Clearly for addition $c = 0$.

Let the value -1 and the orders of known coefficients be associated with each series. Each variable in a problem statement or equivalently each arc of a S.D.G. has associated with it a set of integers. Using the relationship deduced above the action of an operation can be simulated by assigning a set of integers with the set of arcs leaving a vertex. For a given vertex, which represents an operation, it is possible to associate the set $[c, c+1,\ldots,n]$ (defined by the relationship (1)) with the set-of arcs leaving the vertex. Let us now consider only sets of the type $[-1, 0, 1,\ldots,n]$ and characterise this set by the use of the integer n. An integer, bounded by the relationship (1), can be assigned to each set of arcs in a S.D.G.

Consider the real example of a differential equation y' = y.



The value -1 is given to each arc and will be referred to as the value of that arc. Relationship (1) allows the association of the value zero with y but $0 < c$ for integration thus this operation is not possible. Simulation of the operator '=' produces a similar situation. If, however, the initial value of arc y is made zero it is possible to simulate the equals operation to give the following picture.



Now the action of the $\int$ operation can be simulated to give the value 1 to arc y. This process may be continued so that with each arc is associated an integer which is as large as is desired. It should be realised that giving the value zero to arc y was equivalent to giving the differential equation an initial value without which it was impossible to solve.

Theorem.

If at the beginning it is possible to raise the value of each finite valued arc by one then it is possible to raise the values of each finite valued arc to an integer which is as large as is desired.

Proof

Consider an arc whose finite value can be raised by one. Let its value be raised by one to n which is bounded by the relationship (1)

$$c \leq n \leq \min (m_1, m_2, \ldots, m_j) + c$$

Now let the values of every other finite valued arc be raised by one then in particular the value of every finite m_i has changed. Now we have

$$c \leq n + 1 \leq \min (m_1 + 1, m_2 + 1, \ldots, m_j + 1) + c$$

thus it is possible to raise the value of the arc which is under consideration. Since this is true for every finite arc, they can all be raised to an integer which is as large as is desired.

Q.E.D.

The aim of the algorithm is to raise the value of each arc by one. The simulation of this particular sequence of operations corresponds to a simulation of a recurrence relationship.

The algorithm

The S.D.G.'s of the problem statement are drawn and the value -1 is assigned to all arcs except to those arcs which represent constants or the independent variable when known to order zero to which are assigned the value infinity. The algorithm is applied to the initial conditions. If this is successful the values of the arcs in the S.D.G. of the initial conditions are copied to similarly named arcs in the S.D.G. of the differential equations. Then the algorithm is repeated for the S.D.G. of the differential equations.

The nodal application of the algorithm is the act of raising the value of an arc by one to the value n using the relationship

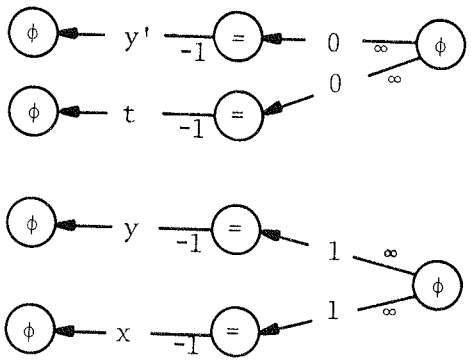
$$c \leq n \leq \min (m_1, m_2, \ldots, m_j) + c$$

A single application of the algorithm consists of a series of nodal applications. After a nodal application the arc to which a value is assigned is marked (by a tick, say) and subsequent nodal applications are made to unmarked arcs until this becomes impossible thus finishing a single application.

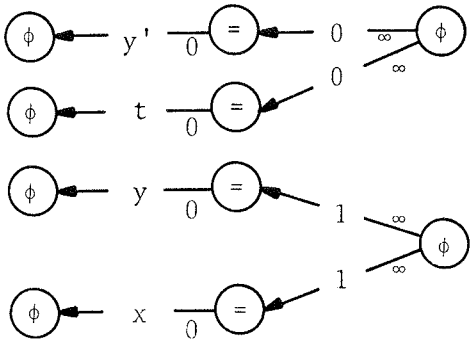
A single application is made and, if all arcs are ticked the set of equations may be well posed. If no previously unmarked arc was marked by a tick the set of equations is ill posed. Otherwise the arcs marked by ticks are remembered, the ticks removed and the algorithm repeated.

The algorithm halts in a finite number of steps since either on each single application an arc is marked for the first time, or else the process is halted, and there are a finite number of arcs.

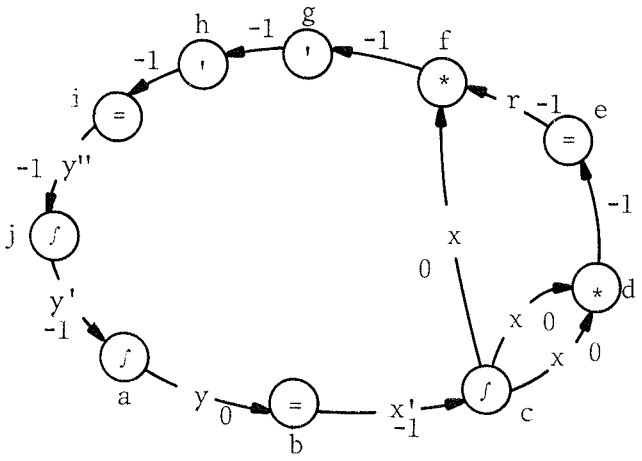
Consider the application of the algorithm to the third example. The S.D.G. for the initial conditions are drawn and labelled.



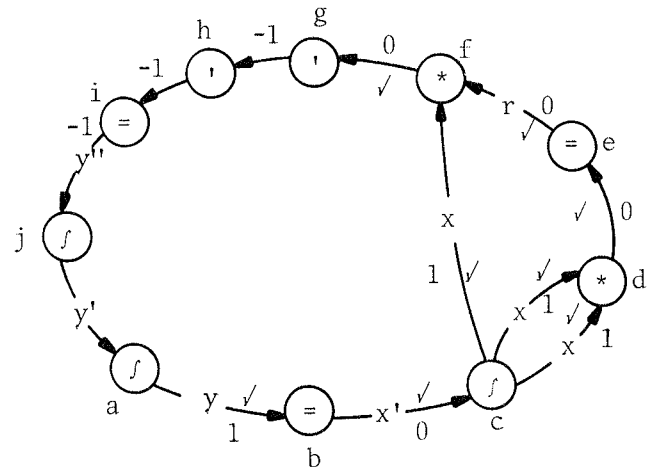
The nodal applications are made in any order giving



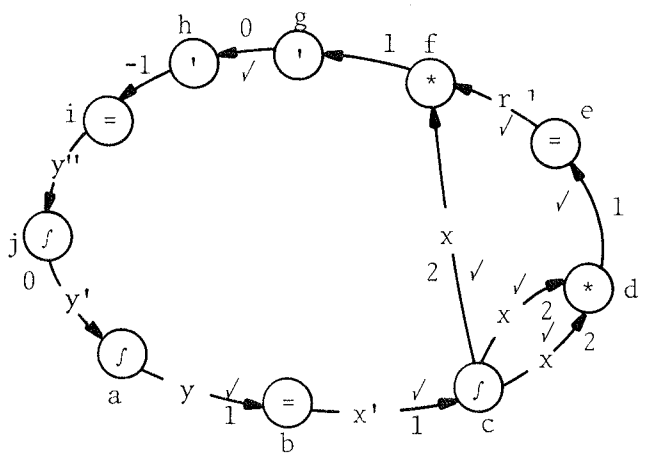
The initial conditions are well posed and so the S.D.G. for the differential equations is drawn and labelled.



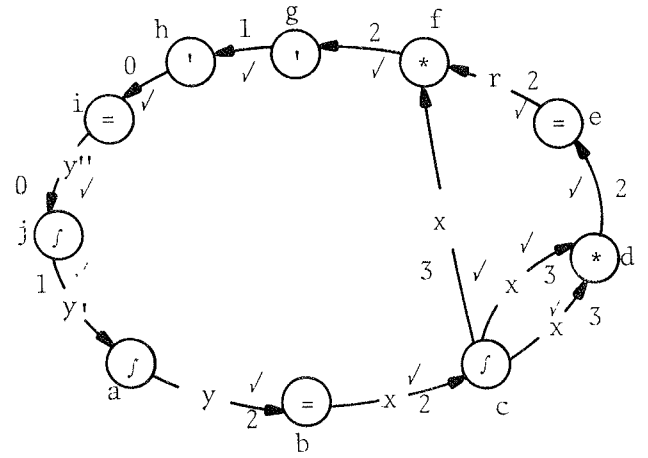
In this example it is interesting to note that all the arcs have finite values. Initially nodal applications can be made at the vertices (a), (b) or (d). Let the algorithm be applied to vertices (a), (b), (c), (d), (e) and (f) in that order.



Since it is impossible to continue this completes a single application. Again there is a choice about the order in which nodal applications can be made. The algorithm is applied to vertices (b), (c), (d), (e), (f) and (g).



The procedure is repeated but now the order of nodal applications is unique and they are applied.



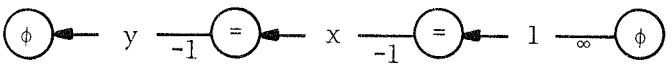
All arcs are marked by ticks and the problem may be well posed.

Consider this fourth example which is not well posed:

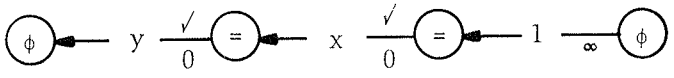
Example 4

Integrate
y' = x'
x = 1 + y
With initial conditions
y = x
x = 1

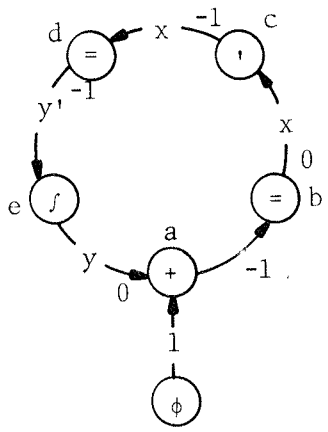
The labelled S.D.G. for the initial conditions is drawn.



The algorithm is applied successfully.



The S.D.G. for the differential equations is also drawn and labelled.



The only vertex where a nodal application can be made is (a). The second single application of the algorithm indicates that the equations are ill posed.

The Output of Code

Each nodal application of the algorithm can be thought of as a simulation of the action of the operational relationship at a vertex. Therefore as a nodal application is made the appropriate object code is generated. If the code that is being generated is from the last single application it is surrounded by a programmed loop. This code within the loop corresponds to a recurrence relationship and the whole loop is known as the recurrence loop. For example 3 the object code is written as a series of sentences which describe the operations. The operations act upon truncated series and produce a single result which is the coefficient of a series. The coefficient order refers to the order of the coefficients that are the results of operations. A series of comments are written down the right hand side of the page to indicate the corresponding nodal applications.

The subroutine for the initial conditions

```
Set coefficient order = 0

Set y' = 0
Set y  = 1
Set x  = 1
Set t  = 0
End
```

The differential equations lead to the code

Comment: beginning of first single application.

Set coefficient order = 0

Integrate y' to y (= x')	(a,b
Integrate x' to x	(c
Set t_1 = x * x (= r)	(d,e
Set t_2 = x * r	(f

Comment: beginning of second single application

Set coefficient order = 1

Integrate x' to x	(c
Set t_1 = x * x (= r)	(d,e
Set t_2 = x * r	(f
Differentiate t_2 to t_3	(g

Comment: beginning of third single application

Set coefficient order = 2

Loop commences:	Set t_1 = x * x (= r)	(d,e
	Set t_2 = x * r	(f
	Differentiate t_2 to t_3	(g

Decrement coefficient order

Differentiate t_3 to t_4 (= y'')	(h,i

Decrement coefficient order

Integrate y'' to y'	(j

Increment coefficient order

Integrate y' to y (= x')	(a,b

Increment coefficient order

Integrate x' to x	(c

Increment coefficient order and loop until required order

End

The algorithm has been coded and runs successfully on the CDC 6400 machine at CERN and produces output in the FORTRAN language. The resulting code has been used by a numerical control program to obtain, quite accurately, the step-by-step solutions to initial value problems of ordinary differential equations.

REFERENCES

[1] Barton, D., Willers, I.M., and Zahar, R.V.M. An implementation of the Taylor series method for ordinary differential equations. Compt. J. 14,3(1971), 243-248; also in The Best Computer Papers of 1971, (Ed.) Petrocelli, Auerbach Philadelphia, 1971, p.147-163.

[2] Barton, D., Willers, I.M., and Zahar, R.V.M. Taylor series methods for ordinary differential equations - an evaluation. Proc. Math. Software Symposium, Purdue U., Lafayette, Ind., 1970.

[3] Moore, R.E. Interval Analysis, Prentice-Hall, Englewood Cliffs, N.J., 1966.

[4] Graham, R.M. Bounded context translation, AFIPS, SJCC, Vol. 25, p.17-29, 1964.