Analytic differentiation using a syntax-directed compiler*

By Herbert Schorr†

The syntactic definition of a language used for analytic differentiation is presented in this paper. The definition is given in Backus normal form. A translation is then added to this definition so that the derivative of any algebraic expression written in the language can be obtained by using the syntax-directed compiler developed by E. T. Irons.

The mathematically inelegant form of the derivatives obtained leads to changes in the original syntactic definition of the language. The language is simultaneously extended in order to obtain: (1) derivatives of higher order, (2) partial derivatives, (3) the derivatives of implicit functions and (4) the derivatives of any number of functions at the same time.

The output of the compiler is the solution of the original problem and not an intermediate-language program to be assembled and executed. This, it is felt, represents a new use of compilers. Also, using Backus normal form rather than assembly language facilitates the programming, checking out and changing of programs. The principle of extending the syntax of any language in order to obtain a more efficient translation is discussed.

The method here presented for performing analytic differentiation differs from other methods (J. W. Hanson *et al.*, 1962) in that it makes use, not of a special language for symbol-manipulation, but instead, of the syntax-directed compiler (E. T. Irons, 1963a).

This compiler requires (1) a specification in a modified Backus normal form (BNF) of the source language and (2) a specification of the translation process to be carried out to convert from the source string to the object string.

Normally the object string is in some form of assembly language, but in the present instance it is the derivative of the source expression.

The Backus normal form specification

Algebraic expressions can be written using only constants, numbers, independent variables, dependent variables and the normal arithmetic operators. In addition, for convenience, parentheses and certain standard functions, e.g. sine and cosine, are introduced. The BNF description of algebraic expressions using only these last two functions is given in Table 1.

The language there defined is the simplest that can be used to differentiate *any* algebraic expression. However, because of the inelegance (mathematically) of the results the table has to be extended to include more standard functions and certain other modifications.

Obtaining derivatives

To obtain, via a syntax-directed compiler, the derivative of an algebraic expression, a "translation" must be added to the above BNF specification. The way of providing such a translation is indicated in the literature (Irons, 1963a, b).

In this case the particular translation to be combined with the BNF specification can be obtained by using the rules of differentiation and observing that the derivative of any algebraic expression is formed from:

(1) the components (or sub-expressions) of an expression, and
(2) the derivatives of these expressions.

For example, the algebraic expression

$$A * B$$

has, as its sub-expressions A and B and its derivative is given by

$$A * B' + B * A'$$

where A′ and B′ are the derivatives of A and B, respectively.

Each sub-expression corresponds to an ⟨expression⟩ in Table 1, and therefore two outputs, the original expression and its derivative, must be associated with each expression defined in that table.

For example, if $C * x \uparrow 2$ is an ⟨expression⟩ the two outputs are (1) the original expression $C * x \uparrow 2$ and

(2) its derivative $2 * C * x$

whilst, for $\mathbf{sin}\,(C * x \uparrow 2)$ the outputs are $\mathbf{sin}\,(C * x \uparrow 2)$ and $2 * C * x * \mathbf{cos}\,(C * x \uparrow 2)$.

The outputs can be obtained by adding the definition

$$\{\mathbf{sin}\,(\rho_{2\cdot1}) \mid \rho_{2\cdot2} * \mathbf{cos}\,(\rho_{2\cdot1})\}$$

to the formation rule

$$\mathbf{sin}\,(\langle\text{expression}\rangle) = :: \langle\text{expression}\rangle$$

The adding of the appropriate definitions to the formation rules of Table 1 leads to Table 2.

The "diagramming" and formation sequence diagram for the input $y = 1/x + \mathbf{sin}\,(a + x \uparrow 2)$, using Table 2, is given in Tables 3*a* and 3*b*, respectively.

* *First submitted September* 1963

† *Now with IBM, P.O. Box* 218, *Yorktown Heights, New York.*

Table 1

Backus normal form specification of algebraic expressions to be differentiated

⟨constant⟩ ::= $a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|z$
⟨independent variable⟩ ::= x
⟨dependent variable⟩ ::= y

⟨digit⟩ ::= 0|1|2|3|4|5|6|7|8|9|
⟨number⟩ ::= ⟨digit⟩ | ⟨number⟩ ⟨digit⟩ |
⟨number⟩ . ⟨number⟩ | . ⟨number⟩

⟨expression⟩ ::= ⟨constant⟩ | ⟨independent variable⟩ |
⟨dependent variable⟩ | ⟨number⟩ |
⟨expression⟩ ↑ ⟨expression⟩ | − ⟨expression⟩ |
+ ⟨expression⟩ | ⟨expression⟩ * ⟨expression⟩ |
⟨expression⟩ / ⟨expression⟩ |
⟨expression⟩ + ⟨expression⟩ |
⟨expression⟩ − ⟨expression⟩ | (⟨expression⟩) |
sin (⟨expression⟩) | **cos** (⟨expression⟩)

⟨statement⟩ ::= ⟨expression⟩ = ⟨expression⟩

⟨program⟩ ::= ⟨statement⟩

Table 2

Syntax table for differentiation

⟨statement⟩ =:: ⟨program⟩ $\{\rho_1\}$
⟨expression⟩ = ⟨expression⟩ ::= ⟨statement⟩ $\{\rho_{3\cdot2} = \rho_{1\cdot2}\}$

sin (⟨expression⟩) =:: ⟨expression⟩ {**sin** $(\rho_2)|(\rho_{2\cdot2})$* **cos** (ρ_2)}
cos (⟨expression⟩) =:: ⟨expression⟩ {**cos** $(\rho_2)| - (\rho_{2\cdot2})$* **sin** (ρ_2)}
⟨expression⟩ ↑ ⟨expression⟩ =:: ⟨expression⟩ $\{\rho_3 \uparrow \rho_1|(\rho_3 \uparrow \rho_1)^*((\rho_1{}^*\rho_{3\cdot2})/\rho_3 + \rho_{1\cdot2}{}^*$ **ln** $(\rho_3))\}$
−⟨expression⟩ =:: ⟨expression⟩ $\{-\rho_1|-\rho_{1\cdot2}\}$
⟨expression⟩ / ⟨expression⟩ =:: ⟨expression⟩ $\{\rho_3/\rho_1|(\rho_{3\cdot2}{}^*\rho_1 - \rho_3{}^*\rho_{3\cdot2})/((\rho_1) \uparrow 2)\}$
⟨expression⟩ * ⟨expression⟩ =:: ⟨expression⟩ $\{\rho_3{}^*\rho_1|\rho_{3\cdot2}{}^*\rho_1 + \rho_3{}^*\rho_{1\cdot2}\}$
⟨expression⟩ − ⟨expression⟩ =:: ⟨expression⟩ $\{\rho_3 - \rho_1|\rho_{3\cdot2} - \rho_{1\cdot2}\}$
⟨expression⟩ + ⟨expression⟩ =:: ⟨expression⟩ $\{\rho_3 + \rho_1|\rho_{3\cdot2} + \rho_{1\cdot2}\}$
+ ⟨expression⟩ =:: ⟨expression⟩ $\{+\rho_1| + \rho_{1\cdot2}\}$
(⟨expression⟩) =:: ⟨expression⟩ $\{(\rho_2)|(\rho_{2\cdot2})\}$
⟨independent variable⟩ =:: ⟨expression⟩ $\{\rho_1|1\}$
⟨dependent variable⟩ =:: ⟨expression⟩ $\{\rho_1|\rho_1'\}$
⟨constant⟩ =:: ⟨expression⟩ $\{\rho_1|0\}$
⟨number⟩ =:: ⟨expression⟩ $\{\rho_1|0\}$
⟨number⟩ =:: ⟨number⟩ $\{\cdot\rho_1\}$
⟨number⟩ . ⟨number⟩ =:: ⟨number⟩ $\{\rho_3\cdot\rho_1\}$
⟨number⟩ ⟨digit⟩ =:: ⟨number⟩ $\{\rho_2\rho_1\}$
⟨digit⟩ =:: ⟨number⟩ $\{\rho_1\}$

x =:: ⟨independent variable⟩ $\{x\}$ $\quad$ y =:: ⟨dependent variable⟩ $\{y\}$

0 =:: ⟨digit⟩ {0} $\quad$ a =:: ⟨constant⟩ $\{a\}$
1 =:: ⟨digit⟩ {1} $\quad$ b =:: ⟨constant⟩ $\{b\}$
. $\quad$.
. $\quad$.
. $\quad$.
9 =:: ⟨digit⟩ {9} $\quad$ z =:: ⟨constant⟩ $\{z\}$

Improvement of output

Although the output obtained by using Table 2 will be the correct derivative, it is inelegant and can be improved by the omission of superfluous terms of the form $0 * x$.

Some of these arise because the derivative of a constant is zero, and these can be eliminated by first defining a ⟨constant expression⟩ as any arithmetic formula that only contains constants. The definition of an ⟨expression⟩ can then be expanded to include as special cases all those formulae in which a ⟨constant expression⟩ appears on one side of a binary arithmetic connective.

If appropriate definitions are appended to each of these expanded formation rules, zero and other superfluous terms will be eliminated.

For example, if

⟨expression⟩ ↑ ⟨constant expression⟩ = :: ⟨expression⟩

$$\{\rho_3 \uparrow \rho_1 | \rho_{3\cdot 2} * \rho_1 * \rho_3 \uparrow (\rho_1 - 1)\}$$

is added to Table 2, the output for

$$(\mathbf{sin}\ (2\ x)) \uparrow b$$

is $$(2 * \mathbf{cos}\ (2 * x)) * b * (\mathbf{sin}\ (2 * x)) \uparrow (b - 1)$$

rather than

$$(\mathbf{sin}\ (2 * x)) \uparrow b\ (b * 2 * \mathbf{cos}\ (2 * x)/$$

$$(\mathbf{sin}\ (2* x)) + 0 * \mathbf{ln}\ (\mathbf{sin}\ (2 * x))$$

In the Appendix an expanded version of the syntax of Table 1 is given which allows, as well as the above,

(A) an increase in the number of standard functions permitted;
(B) the inclusion of **comments**, which are ignored by the compiler;
(C) the differentiation of an implicit function;
(D) the representation of differentials in the input expression as y', y'' etc;
(E) the differentiation of more than one formula at a time;
(F) a better output, by the inclusion of other special cases of the differentiation formulas

and

(G) the use of declarations.

Changes (A) and (B) are self-explanatory, whilst changes (C), (D) and (E) will be discussed in the next section; changes (F) and (G) are discussed below.

Some special cases of the differentiation formulas included, besides those already discussed, are:

⟨independent variable⟩ ↑ 1 =:: ⟨expression⟩ $\{\rho_3|1\}$
⟨independent variable⟩ ↑ ⟨number⟩=:: ⟨expression⟩
$\{\rho_3 \uparrow \rho_1|\rho_1\rho_3 \uparrow \rho_{1\cdot 2}\}$

These sentences, given the inputs $x \uparrow 1$ and $x \uparrow n$, respectively, produce outputs 1 and $nx \uparrow m$ where $m = n - 1$ (for $n = 3$ the output is $3x \uparrow 2$), respectively.

In order to obtain the output $nx\uparrow m$, two definition strings must be associated with every number. The first definition string is the number itself, while the second is the number minus one. Thus, in the definition of the second sentence above, $\rho_{1\cdot 2}$ stands for one less than the number represented by ρ_1. The syntactic definition of ⟨number⟩ which provides the outputs required is given and discussed in section 1·3 of the Appendix. The improved output obtained by using the syntax tables corresponding to the BNF description given in the Appendix, rather than Table 2, is illustrated in Table 4. This table is a formation sequence diagram for the example of Table 3.

Change (G), the use of declarations, permits any ⟨identifier⟩ to stand for, or identify, any variable or constant. Furthermore, unless the concatenation of two identifiers is declared as a third identifier, the compiler recognizes that, for example,

$$xy$$

is the juxtaposition of the two identifiers x and y. If the juxtaposition of two identifiers is understood to mean the multiplication of the two quantities represented by the identifiers, then the compiler can accept as an input the expression

$$\mathbf{sin}\ cdx \uparrow 2y + 2x$$

which corresponds to

$$\mathbf{sin}\ (c*d*x \uparrow (2*y)) + 2*x$$

in the language presented in Table 1. Any product of variables and constants in which the multiplication operator * is omitted is defined (syntactically) as a ⟨simple expression⟩. The syntactic definition of ⟨simple expression⟩ is given in section 2.2 of the Appendix.

Functions whose derivatives can be obtained

As mentioned above, (C), this program can be used to obtain the derivative of an implicit function. For such a function, the output of the program is an algebraic expression from which the derivative can easily be obtained. For example, for

$$x^3 + x^2y^2 + 1 = x + 2y$$

the program output is

$$3x^2 + 2xy^2 + x^2 2yy' + 0 = 1 + 2y'$$

from which

$$y' = \frac{3x^2 + 2xy^2 - 1}{2 - 2xy^2}$$

can be obtained. A syntax directed compiler program to obtain y' can be written.

Derivatives of higher order can be obtained using the differentiation program. If a derivative of order n is desired, the program is used n times with the previous output as its input. Hence, the program must allow differentials such as y', y'', etc. (change (D) above) to

(*text continued on page* 295)

Key for Tables 3 and 4

d.v.	—	dependent variable
i.v.	—	independent variable
num	—	number
cons	—	constant
exp	—	expression
digit	—	digit
statement	—	statement
program	—	program
s.e.	—	simple expression

Table 3

Differentiation of $y = 1 / x + \sin(a * x \uparrow 2)$

a. *Diagram of the formula*

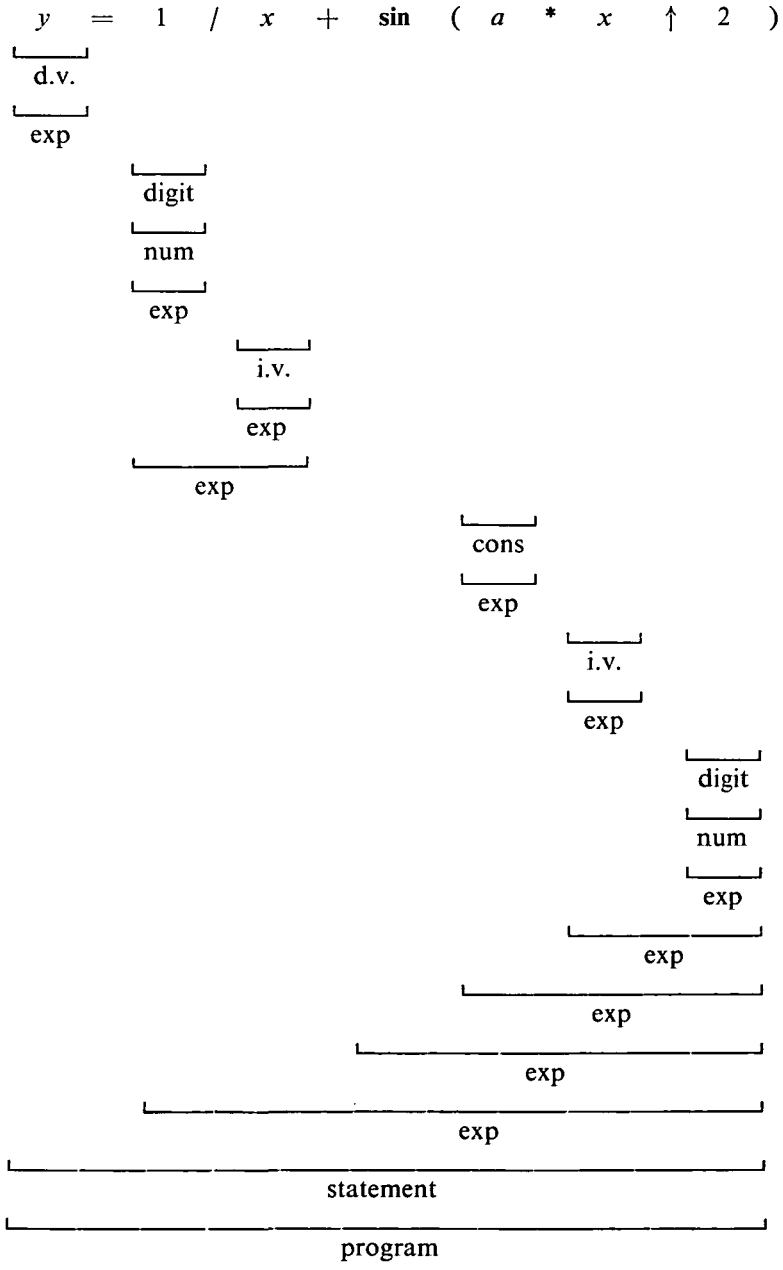

Table 3 (contd.)

b. *Formation sequence diagram*

symbol	y	y	1	1	1	x	x
subject of sentence	d.v.	exp	digit	num	exp	i.v.	exp
output	y	$y \mid y'$	1	1	1|0	x	$x\|1$

$1/x$	a	a	x	x	2
exp	cons	exp	i.v.	exp	digit
$1/x \mid (0*x-1*1)/((x)\uparrow 2)$	a	$a\|0$	x	$x\|1$	2

2	2	$x\uparrow 2$
num	exp	exp
2	2|0	$x\uparrow 2 \mid (x\uparrow 2)*((2*1)/x + 0*\ln(x))$

$a*x\uparrow 2$
exp
$a*x\uparrow 2 \mid 0*x\uparrow 2 + a*(x\uparrow 2)*((2*1)/x + 0*\ln(x))$

$\sin(a*x\uparrow 2)$
exp
$\sin(a*x\uparrow 2) \mid (0*x\uparrow 2 + a*(x\uparrow 2)*((2*1)/x + 0*\ln(x)))*\cos(a*x\uparrow 2)$

$1/x + \sin(a*x\uparrow 2)$
exp
$1/x + \sin(a*x\uparrow 2) \mid (0*x - 1*1)/((x)\uparrow 2) + (0*x\uparrow 2 + a*(x\uparrow 2)*((2*1)/x + 0*\ln(x)))*\cos(a*x\uparrow 2)$

$y = 1/x + \sin(a*x\uparrow 2)$
statement, program
$y' = (0*x - 1*1)/((x\uparrow 2) + (0*x\uparrow 2 + a*(x\uparrow 2)*((2*1)/x + 0*\ln(x)))*\cos(a*x\uparrow 2)$

Table 4

Formation sequence diagram of $y = 1/x + \sin(a^*x \uparrow 2)$ using the syntax given in the Appendix

symbol	y	y	y	y	1	1	x
subject of sentence	d.v.	var	s.e.	exp	num	cons	i.v.
output	$y \mid y'$	$y \mid y'$	$y \mid y'$	$y \mid y'$	1|0	1|0	x|1

$1 / x$	$1/x$	a	x	2	$x \uparrow 2$
s.e.	exp	cons	i.v.	num	s.e.
$1/x \mid - 1/x \uparrow 2$	$1/x \mid - 1/x \uparrow 2$	$a \mid 0$	$x \mid 1$	$2 \mid 1$	$x \uparrow 2 \mid 2x$

$a^*x \uparrow 2$	$(a^*x \uparrow 2)$	**sin** $(a^*x \uparrow 2)$
s.e.	s.e.	exp
$a^*x \uparrow 2 \mid a^*2x$	$a^*x \uparrow 2 \mid a^*2x$	**sin** $(a^*x \uparrow 2) \mid a^*2x$ **cos** $(a^*x \uparrow 2)$

$1/x +$ **sin** $(a^*x \uparrow 2)$
exp
$1/x +$ **sin** $(a^* x \uparrow 2) \mid - 1/x \uparrow 2 + a^*2x$ **cos** $(a^*x \uparrow 2)$

begin $y = 1/x +$ **sin** $(a^*x \uparrow 2)$ **end**
statement
$y' = - 1/x \uparrow 2 + a^*2x$ **cos** $(a^*x \uparrow 2)$

appear in the input expression. Also, the following rules of differentiation must be incorporated in the syntax table corresponding to the BNF description given in the Appendix:

$$\frac{d}{dy}(y') = y''$$

$$\frac{d}{dy}(y'') = y'''$$

etc.

Every expression to be differentiated is enclosed within the brackets **begin** and **end**. This permits (change (E)) more than one expression at a time to be differentiated (see section 3.1 of the Appendix).

The partial derivative of a function may also be obtained. If $z = f(x, y)$ then $\frac{\partial z}{\partial y}$ can be obtained by using the following program:

begin independent variable (y);
dependent variable (z);
constant $(x, a, b, c, \ldots)$;
begin $z = f(x, y)$ **end end**

The derivative $\frac{\partial z}{\partial y}$ of an implicit function such as

$$f(x, y, z) = g(x, y, z)$$

can be obtained by changing the last line of the above program to

begin $f(x, y, z) = g(x, y, z)$ **end end**

As above, the output of the program is an algebraic expression from which $\frac{\partial z}{\partial y}$ can easily be obtained.

If w is a complex function of z, and

$$w = f(z) = u(x, y) + iv(x, y)$$

then if the function has a derivative it obeys the Cauchy–Reimann conditions and

$$\frac{dw}{dz} = \frac{\partial u}{\partial x} + \frac{i\partial v}{\partial x} = \frac{\partial f}{\partial x}.$$

Thus, $\frac{dw}{dz}$ is obtained by the following program:

begin independent variable (x);
dependent variable (w);
constant $(y, i, a, b, c, \ldots)$;

begin $w = u(x, y) + iv(x, y)$ **end end**

Ambiguity and operator precedence

The BNF specification of a language is often ambiguous (Cantor, 1962, Corn, 1963). This is true of the specification given in the Appendix. The ambiguity is removed by specifying the order in which the sentences formed from the syntax are to be used in translating an expression. The method by which this order is specified is presented in detail elsewhere (Schorr, 1962); essentially the order in which the sentences are used corresponds with their order of appearance in a syntax table such as Table 2. Combined with the right-to-left scan employed by the compiler, the order given in Table 2 ensures that the rules of arithmetic precedence of the operators presented in section 2.2 of the Appendix hold.

Conclusions

A program for obtaining the derivatives of algebraic formulas has been presented. This program is written in Backus normal form as augmented by E. T. Irons for use in a syntax-directed compiler. For an ALGOL 60 program the output of this compiler is an equivalent assembly language program. This program has to be assembled and the resulting machine-code program executed before the solution of the original problem is achieved. In contrast, for the differentiation program, the output of the compiler is the solution of the original problem. Therefore, in addition to its use in writing compilers, augmented Backus normal form can be used as a programming language in its own right. Thus, the usefulness of the compiler, originally written to translate ALGOL 60 (or similar language) programs, has been extended.

It is shown in this paper that changes sometimes are required in the simplest syntactic definition of a language. These changes were necessary to improve the translation obtained. If a syntax-directed compiler is to be used, then these changes were probably necessary in the original syntactic definition of any language if an efficient or improved translation is to be obtained. This is especially true if the translation is to be made into the assembly language of a particular computer, and the computer's peculiarities are to be taken advantage of, or compensated for.

Besides illustrating the above syntax-directed compiler principles and uses, the program for analytic differentiation has the following advantages over other such programs.

(1) The program is not written in an assembly language but in a higher-level language. This facilitates programming and results in fewer errors.
(2) Since the syntax-directed compiler is an already tested program, only the syntax tables remain to be tested. For example, no input/output routines have to be tested.
(3) Any new function can be added to this program by just adding a sentence to the syntax tables derived from the Appendix.
(4) Any other desired changes in the program can be made without too much difficulty.
(5) Derivatives of higher order, partial derivatives, and the derivatives of implicit functions and complex functions can be obtained.
(6) Any identifier can be used to stand for any constant or variable desired.
(7) The derivative of more than one function at a time can be found.

Work on the syntax-directed compiler programs to do the following appears to be feasible and will be investigated.

(1) Simultaneously calculate all of the partial derivatives of a function.
(2) Calculate the Cauchy–Riemann equations for a complex function.
(3) Apply the chain rule to a change of independent variables.
(4) Solve for the derivative in the output expression when the input of the analytic differentiation program is an implicit function.

Such a set of programs should provide a complete package for analytic differentiation.

Acknowledgement

The author would like to thank Mr. Eric Nixon for his help in preparing the manuscript.

References

BACKUS, J. W. (1959). "The syntax and semantics of the proposed international algebraic language of the Zurich ACM–GAMM conference," *Proc. International Conf. Information Processing*, UNESCO, Paris, France, June 1959, pp. 125–132.

CANTOR, D. G. (1962). "On the ambiguity problem of Backus systems," *Jour. ACM*, Vol. 9, No. 4, pp. 477–479.

GORN, S. (1963). "Detection of generative ambiguities in context-free mechanical languages," *Jour. ACM*, Vol. 10, No. 2, pp. 196–208.

HANSON, J. W., CAVINESS, J. S., and JOSEPH, C. (1962). "Analytic differentiation by computer," *Comm. of the ACM*, Vol. 5, No. 6, pp. 349–355.

IRONS, E. T. (1963a). "The structure and use of the Syntax Directed Compiler," *Annual Review in Automatic Programming*, Vol. 3, pp. 207–227.

IRONS, E. T. (1963b). "Towards More Versatile Mechanical Translators," Proc. of Symposium in Applied Mathematics, *Amer. Math. Soc.*, Vol. XV, pp. 41–50.

NAUR, P. *et al.* (1960). "Report on the algorithmic language ALGOL 60," *Comm. of the ACM*, Vol. 3, No. 5, pp. 299–314, and *Computer Journal*, Vol. 5, p. 349, (Jan. 1963).

SCHORR, H. (1962). "A syntax directed translation procedure," TR No. 25, Dept. of Electrical Engineering, Digital Systems Lab., Princeton University.

Appendix

The syntax of a language for analytic differentiation

1. Basic symbols, identifiers and numbers

1.1 *Letters and delimiters*

⟨basic symbol⟩ ::= ⟨letter⟩ | ⟨digit⟩ | ⟨delimiter⟩ | ⟨standard function⟩

⟨letter⟩ ::= *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *J* | *K* | *L* | *M* | *N* | *O* | *P* | *Q* | *R* | *S* | *T* | *U* | *V* | *W* | *X* | *Y* | *Z* | *a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* | *j* | *k* | *l* | *m* | *n* | *o* | *p* | *q* | *r* | *s* | *t* | *u* | *v* | *w* | *x* | *y* | *z*

⟨delimiter⟩ ::= ⟨arithmetic operator⟩ | ⟨separator⟩ | ⟨bracket⟩ | ⟨declarator⟩

⟨arithmetic operator⟩ ::= + | − | * | / | ↑

⟨adding operator⟩ ::= + | −

⟨multiplying operator⟩ ::= * | /

⟨separator⟩ ::= , | . | = | ; | **comment**

⟨bracket⟩ ::= (|) | **begin** | **end** | [|]

⟨declarator⟩ ::= **independent variable** | **dependent variable** | **constant**

Discussion. The same "comment" conventions as in ALGOL 60 apply here. The juxtaposition of two identifiers to indicate multiplication is "defined" as both a multiplying operator and an arithmetic operator.

1.2 *Identifiers*

⟨identifier⟩ ::= ⟨letter⟩ | ⟨identifier⟩ ⟨letter⟩ | ⟨identifier⟩ ⟨digit⟩

Discussion. Identifiers have no inherent meaning but are used for the identification of variables and constants.

1.3 *Function names*

⟨function name⟩ ::= **exp**|**ln**|**log.** ⟨number⟩ . |**sin**|**cos**|**tan**|**cot**|**sec**|**csc**|**arcsin**|**arccos**|**arctan**|**arccot**|**arcsec**|**arccsc**|**sinh**|**cosh**|**tanh**|**coth**|**sech**|**csch**|**arcsinh**|**arccosh**|**arctanh**|**arccosh**|**arcsech**|**arccsch**

Discussion. The functions whose names are given above are, respectively, the exponential, natural logarithm, logarithm to the base ⟨number⟩, sine, cosine, tangent, cotangent, secant, cosecant, arcsine, arccosine, arctangent, arccotangent, arcsecant, arccosecant, hyperbolic sine, hyperbolic cosine, hyperbolic tangent, hyperbolic cotangent, hyperbolic secant, hyperbolic cosecant, hyperbolic arcsine, hyperbolic arccosine, hyperbolic arctangent, hyperbolic arccotangent, hyperbolic arcsecant, and hyperbolic arccosecant function.

1.4 *Numbers*

⟨digit⟩ ::= 1|2|3|4|5|6|7|8

⟨zero⟩ ::= 0

⟨nine⟩ ::= 9

⟨all zero⟩ ::= ⟨zero⟩ | ⟨all zero⟩ ⟨zero⟩

⟨integer⟩ ::= ⟨digit⟩ | ⟨integer⟩ ⟨digit⟩ | ⟨integer⟩ ⟨zero⟩ | ⟨integer⟩ ⟨nine⟩ | ⟨zero⟩ | ⟨nine⟩

⟨number⟩ ::= ⟨integer⟩ | ⟨integer⟩ . ⟨integer⟩ | . ⟨integer⟩ | −. ⟨integer⟩ | − ⟨integer⟩ . ⟨integer⟩ | ⟨all zero⟩ . ⟨integer⟩ | + ⟨number⟩

Discussion. With each ⟨digit⟩, except 0 and 9, five definition strings are associated: they are, (1) the digit itself, (2) the digit minus 1, (3) the digit plus 1, (4) the ten's complement of the digit, and (5) the nine's complement of the digit. Zero and nine are treated as special cases. An integer is formed from a juxtaposition of digits. By properly combining the definition strings of each digit, five strings are associated with every integer; these strings are the same as the five above strings (with the word digit replaced by the word integer). Finally, each ⟨number⟩ is formed from one or two integers and has two definition strings associated with it: (1) the ⟨number⟩ itself, and (2) the ⟨number⟩ minus one. For example, the integer 523 has the following definition strings

523 | 522 | 524 | 477 | 476

and the definition strings of the numbers 523, .523, 523.523, −.523, −523.523 are, respectively:

523 | 522
.523 | −.477
523.523 | 522.523
−.523 | −1.523
−523.523 | −524.523

The two definition strings of a number m, are used to obtain the output mx^n, where $n = m - 1$, as the derivative of x^m. For example, the output $523x^{522}$ is obtained as the derivative of x^{523}.

2. Expressions

2.1 *Variables, constants, constant expressions and subscripts*

⟨independent variable⟩ :: = ⟨identifier⟩ | ⟨identifier⟩ [⟨subscript⟩]

⟨dependent variable⟩ :: = ⟨identifier⟩ | ⟨identifier⟩ [⟨subscript⟩]
⟨variable⟩ :: = ⟨independent variable⟩ | ⟨dependent variable⟩ | ⟨variable⟩
⟨constant⟩ :: = ⟨identifier⟩ | ⟨identifier⟩ [⟨subscript⟩] |
⟨number⟩ | ⟨constant⟩ * ⟨constant⟩ |
⟨constant constant⟩
⟨constant expression⟩ :: = ⟨constant⟩ | ⟨constant expression⟩ ⟨arithmetic operator⟩ ⟨constant expression⟩ | ⟨function name⟩ ⟨constant⟩| ⟨function name⟩ (⟨constant expression⟩) | (⟨constant expression⟩)
⟨subscript⟩ :: = ⟨identifier⟩ | ⟨integer⟩ | ⟨identifier⟩+ ⟨integer⟩ | ⟨identifier⟩ − ⟨integer⟩ | ⟨integer⟩ + ⟨identifier⟩ | ⟨integer⟩ − ⟨identifier⟩

Examples

x
y
$x[1]$
$C[1] * C[2]$
$C[1]\ C[2]$
$x[i + 1]$

Discussion. $C[1]\ C[2]$ represents the product of the constants $C[1]$ and $C[2]$; the multiplication sign * has been omitted.

2.2 *Expressions*

⟨simple expression⟩ :: = ⟨variable⟩ | ⟨constant⟩ |
⟨variable⟩ ↑ 1 | ⟨variable⟩ ↑ 2 | ⟨variable⟩ ↑ ⟨number⟩ | ⟨variable⟩ ↑ ⟨constant⟩ |
⟨constant⟩ ⟨multiplying operator⟩ ⟨simple expression⟩ | ⟨simple expression⟩ ⟨multiplying operator⟩ ⟨constant⟩ | ⟨simple expression⟩ ⟨multiplying operator⟩ ⟨simple expression⟩ | (⟨simple expression⟩)
⟨expression⟩ :: = ⟨constant expression⟩ | ⟨simple expression⟩ | ⟨expression⟩ ↑ 1 | ⟨expression⟩ ↑ 2 | ⟨expression⟩ ↑ ⟨number⟩ | ⟨expression⟩ ⟨arithmetic operator⟩ ⟨constant expression⟩ | ⟨constant expression⟩ ⟨arithmetic operator⟩ ⟨constant expression⟩ | ⟨expression⟩ ⟨arithmetic operator⟩ ⟨expression⟩ | (⟨expression⟩) | − ⟨expression⟩ | + ⟨expression⟩
⟨function⟩ ⟨variable⟩ | ⟨function name⟩ ⟨simple expression⟩ | ⟨function name⟩ (⟨expression⟩)

Examples

$ax \uparrow 2 + bx + \mathbf{sin}\ cx$
$a*x \uparrow 2 + b*x + \mathbf{sin}\ (c*x)$
$xy + \mathbf{exp}\ (xy \uparrow 2 + \mathbf{sin}\ y) / y \uparrow (2x + 3)$

Discussion. A ⟨simple expression⟩ is a product of variables, powers of variables and constants; an ⟨expression⟩ is any algebraic formula.

The following rules of precedence hold:—

first: standard functions
second: exponentiation
third: unary minus
fourth: division
fifth: multiplication
sixth: subtraction
seventh: addition

3. Statements, declarations, blocks and programs

3.1 *Statements*

⟨statement⟩ :: = **begin** ⟨expression⟩ = ⟨expression⟩ **end** |
⟨statement⟩ ⟨statement⟩

Examples

begin $y = ax \uparrow 2 + bx + sin\ cx$ **end**
begin $ay \uparrow 2 + cos\ (yx + 10) = xy + exp$
$(xy \uparrow 2 + sin\ y) / y \uparrow (2x + 3)$ **end**

Discussion. This program may be used to obtain the derivative of more than one expression at a time. Each expression whose derivative is to be obtained corresponds to a ⟨statement⟩. The derivative of an implicit function can also be obtained using this program.

3.2 *Declarations*

⟨identifier list⟩ :: = ⟨identifier⟩ | ⟨identifier⟩ [⟨subscript⟩] | ⟨identifier list⟩, ⟨identifier list⟩
⟨declaration⟩ :: = **independent variable** (⟨identifier⟩); **dependent variable** (⟨identifier⟩); **constant** (⟨identifier list⟩);

Example

independent variable (x);
dependent variable (y);
constant (a, b, c);

Discussion. A declaration serves to identify the independent variable, the dependent variable and constants of a statement to be differentiated.

3.3 *Blocks*

⟨block⟩ ::= **begin** ⟨declaration⟩ ⟨statement⟩ **end** | **begin** ⟨declaration⟩ ⟨block⟩ **end** | ⟨block⟩ ⟨block⟩

Example

begin independent variable (x);
dependent variable (y);
constant (a, b, c);
begin $y = ax \uparrow 2$ **sin** $cx + bx$ / **cos** x **end end**

Discussion. Any identifier in a declaration appearing within a block is valid only for the block. This permits an identifier, for example x, to be an independent variable within one block, and a constant in another block.

3.4 *Program*

⟨program⟩ ::= ⟨block⟩