Conditions for underflow and overflow of an arithmetic stack

D. T. Goodwin

Department of Computer Science, University of Keele, Keele, Staffordshire ST5 5BG

The behaviour of an arithmetic stack is formally described as the loading of an arbitrary string of a context free language symbol by symbol on to a stack. Instances of a special symbol in the string being loaded invoke an operation which removes the top cell of the stack in some undefined way. Necessary and sufficient conditions for stack length boundedness are stated and proved. One application of the results concerns the choice between compile time and run time checks for underflow and overflow. Another concerns the testing for applicability of a certain algorithm for inverting Metcalfe-Reeves translators.

(Received June 1975)

The formal device studied by this paper is a push-down store or stack on to which a string of symbols is being loaded, one symbol at a time. One special symbol **C** causes the top two cells of the stack to be replaced by one cell, in some undefined way. Similarly, other special symbols $P_1, P_2, \ldots P_R, \ldots$ may be used. Any instance of P_R in the string is not loaded, but instead causes a particular (non-identity) permutation of a fixed number r_R of the top cells of the stack. The only restriction on the input is that it is an arbitrary string of a fixed context free (CF) language, as defined briefly in the section on notation below.

Example

Let the only P_R be denoted by **X**, which interchanges the top two cells of the stack, and let **C** concatenate the top two cells of the stack. Then the string *cb***XC***a***XC** is processed from left to right as follows:

Stack contents (top cell at right)	*String to be loaded*
—	*cb***XC***a***XC**
c	*b***XC***a***XC**
c b	**XC***a***XC**
b c	**C***a***XC**
bc	*a***XC**
bc a	**XC**
a bc	**C**
abc	—

This device and the special symbols **C** and **X** of this example were used by Metcalfe (1964) and Reeves (1967) to edit the output from a syntax driven translation system. In Goodwin (1975) a translation system on the same lines was developed which allowed the input/output grammars or 'translators' to be inverted automatically under certain given conditions. These conditions were restrictions on the ways in which translators were able to form items on the edit stack, using **C**. However, no proofs were given of stated algorithms for determining for an arbitrary translator whether the conditions were true. These proofs are given here.

Another application of more general interest is the provision of compile time and run time checks on whether certain high level language programs will cause underflow or overflow of a run time arithmetic stack. FORTRAN programs whose translated form uses a hardware or software stack could be processed, and also and more generally, ALGOL 60 and certain POP2 programs, where the user can access the stack directly, and where recursive function calls are allowed. These possibilities are discussed further at the end of the paper.

In what follows only the number of cells on the stack is studied and not their contents. This explains how it is possible to leave **C** undefined except in so far as it removes the top cell. As seen above, it could have the additional effect of concatenating cell contents, or in the second application it might be a arithmetic operator, or a transfer to store instruction.

The theorems below are on the following loosely described topics:

Theorem 1 When the stack length is unbounded

Theorem 2 Uniqueness of the stack length

Theorem 3 Conditions for boundedness of the stack length *after* loading of the string

Theorem 4 Conditions for boundedness of the stack length *during* loading of the string

Theorem 5 Restrictions on stack length bounds during loading

Theorem 6 Disturbances caused by permutations of the stack cells.

The next two sections give the notation to be used in the discussion of context free languages and graphs, and include some simple lemmas. The treatment of the number of cells then proceeds.

1. Notation

A context free language is a language defined by a grammar G as follows. G consists of:

1. A finite alphabet of 'terminal symbols'. Here the unsuffixed letters a, b, c ... are used for these. A general terminal is denoted by t.
2. A finite alphabet of 'nonterminal symbols'. A general nonterminal is denoted by N, which is often suffixed. Sometimes $N_1, N_2, \ldots N_h, \ldots N_n$, are used to denote all the nonterminals of G.
3. A finite number of 'production rules' each of which is of the form

$$N -> C_1 C_2 \ldots C_j \ldots C_p \ ,$$

where each of the C_j may be either a single terminal or a single nonterminal. There may be a number of rules with N_h, say, on the left hand side. These are called 'the rules of N_h'. Any rule of G is identified as $R_i(N)$, the ith rule of N, where the rules of N are numbered in some arbitrary order. When all the rules of N are being considered at once, the ith rule is written

$$N -> C_{i1} C_{i2} \ldots C_{ij} \ldots C_{ip} \ .$$

(It is understood that in general the value of p, the index of the last symbol of a rule, will vary from rule to rule).

4. A special nonterminal S which is one of $\{N_1, N_2, \ldots N_n\}$.

The word 'string is now restricted to meaning an arbitrary concatenation of symbols, possibly empty, which unless other-

wise stated may be any out of the alphabets 1 and 2. A general string is denoted by u, v, or w, possibly suffixed. A general string of terminals only is denoted by s. x is used for a single symbol which is either a terminal or a nonterminal. u^n denotes the string $uu \ldots u$ where u is repeated n times.

If a string w_0 contains a nonterminal N_0 it can be 'expanded' by an application of a rule $R_i(N_0)$. Let $w_0 = u_0 N_0 v_0$. Having chosen i, this instance of N_0 is replaced in w_0 by

$$C_{i1} C_{i2} \ldots C_{ij} \ldots C_{ip} .$$

This operation is written

$$w_0 = u_0 N_0 v_0 => u_0 C_{i1} \ldots C_{ij} \ldots C_{ip} v_0 = w_1 \text{ (say)} .$$

Similarly if w_1 contains an instance of N_1 (say), not necessarily distinct from N_0, a rule of N_1 can be used to expand w_1 into w_2. This operation $w_k => w_{k+1}$ can be continued as long as w_k contains at least one nonterminal. From now on the mark $=>$ is used more generally to show that w_k has been expanded from w_0 in one or more steps, for any $k > 0$. $w_0 => w_k$ is a 'derivation' or a 'derivation of w_k from w_0'.

Notice that w_{k+1} contains symbols of two distinct origins. It contains $C_{i1} \ldots C_{ip}$ which appear because of the application of the N_k rule, and also other symbols which were present in w_k. In the derivation of w_{k+2} the nonterminal N_{k+1} may be chosen from either group of symbols in w_{k+1}. However, it will be useful to discuss $w_0 => w_1 => \ldots => w_k => \ldots w_q$ in whichfor each k, N_{k+1} is chosen only out of the $C_{i1} \ldots C_{ip}$ in w_{k+1} which arise from the expansion of N_k. Such a derivation $w_0 => w_q$ is here called a 'chained derivation' and is denoted by $w_0 \overset{c}{=>} w_q$. It follows that there exist $u_0, v_0, \ldots u_k, v_k \ldots$ such that

$$w_0 = u_0 N_0 v_0 => u_0 u_1 N_1 v_1 v_0 => u_0 u_1 u_2 N_2 v_2 v_1 v_0$$
$$=> \ldots => u_0 u_1 \ldots u_q N_q v_q \ldots v_1 v_0 = w_q .$$

A derivation $N = w_0 => w_k$ can be expressed as a 'generation tree', which is a tree whose nodes are (all the instances of) the symbols of $w_0, w_1, \ldots w_k$. Branches leave a node N_0 to arrive at the symbols into which N_0 is expanded by a rule application.

Example:
Given rules $N_0 -> N_1 N_2$, $N_1 -> ab$, $N_2 -> cN_3$, $N_3 -> dN_4 e$, then $N_0 => abcN_3$ has the generation tree shown in **Fig. 1.** The chained derivation $N_0 \overset{c}{=>} N_1 cdN_4 e$ has the generation tree in **Fig. 2,** which shows its characteristic linear sequence of rule applications.

The strings of the *CF* language determined by G are now defined as those (finite) terminal strings s such that $S => s$.

G is 'admissible' if for each N not the same as S, there exist some u, v such that $S => uNv$ and there exists s such that $uNv => s$. Only admissible grammars are considered below.

A 'recursive derivation' is a derivation of the form $N \overset{c}{=>} uNv$ for arbitrary words u, v. The generation tree of a recursive derivation is also called recursive.

A 'chained recursive derivation' is a derivation $N => uNv$ which is both chained and recursive, corresponding to a generation tree which is a linear sequence of rule applications beginning and ending at N-nodes.

A 'cycle' C of G is defined by a sequence

$$N_1(i_1, i_1) N_2(i_2, j_2) \ldots N_r(i_r, j_r) \ldots N_m(i_m, j_m)$$

of alternating nonterminal instances N_r and number pairs (i_r, j_r) such that

1. for each $r < m$, $C_{i_r j_r} = N_{r+1}$, in the i_rth rule of N_r,
and
2. $C_{i_m j_m} = N_1$, in the i_mth rule of N_m.

Given such a sequence, any cyclic permutation such as

$$N_2(i_2, j_2) \ldots N_r(i_r, j_r) \ldots N_m(i_m, j_m) N_1(i_1, j_1)$$

identifies the same cycle.

Let a cycle C' be $N_1'(i_1', j_1') \ldots N_m'(i_m', j_m')$ and let C and C' have a common nonterminal $N_1 = N_1'$. Then the sequence

$$N_1(i_1, j_1) \ldots N_m(i_m, j_m) N_1'(i_1', j_1') \ldots N_m'(i_m', j_m')$$

also defines a cycle which is said to be 'composed' from C and C'.

Now in $R_{i_r}(N_r)$ let $u_r \equiv C_{i_r 1} C_{i_r 2} \ldots C_{i_r j_r - 1}$, and let

$$v_r \equiv C_{i_r j_r + 1} \ldots C_{i_r p} .$$

Choose a cycle C of G and N_r in it. Then these choices identify a chained recursive derivation

$$N_r \overset{c}{=>} u_r u_{r+1} \ldots u_m u_1 u_2 \ldots u_{r-1} N_r v_{r-1} v_{r-2} \ldots v_1 v_m v_{m-1} \ldots v_{r+1} v_r .$$

Given a cycle C all such chained recursive derivations are called 'the chained recursive derivations of C'. It follows that

Lemma 1
In all the chained recursive derivations $N \overset{c}{=>} u_c N v_c$ of a given cycle C, the strings u_c, v_c are constant, except for permutations of symbols.

Further notation is introduced as required.

2. A useful graph

It is helpful to introduce below a directed graph G_R associated with G. The simple graph theory and terminology used here is adapted from Berge and Ghouila-Houri (1969).

A finite graph consists of a finite number of points or 'nodes' joined together by a finite number of directed lines or 'arcs'.

A 'path' is a sequence of arcs such that the end node of one arc is the start node of the next. A path may pass through a particular node more than once, and use a particular arc more than once.

A 'circuit' is a path in which any node of the path can be considered as both the start node and end node of the path.

An 'elementary circuit' is a circuit in which no node occurs more than once. There are clearly only a finite number of elementary circuits.

If two circuits A, B have a common node p then another circuit C can be 'composed' out of the 'components' A, B by joining them at p. Thus no composed circuit can be elementary. By examining multiple instances of nodes on a circuit it is easy to see that:

Lemma 2
Every (finite) circuit of a graph is either elementary or can be composed out of a finite number of elementary circuits, perhaps repeated.

The graph G_R is now constructed. Its nodes are single instances of the nonterminals and terminals of G. Draw from each N an arc to every C_{ij} in each rule $R_i(N)$. This arc may be labelled (N, i, j). Add a special end node E, and draw an arc t from every terminal t to E.

Example
For the rules

$S -> NS$	$R_1(S)$
$S -> b$	$R_2(S)$
$N -> aNC$	$R_1(N)$
$N -> b$	$R_2(N)$

the graph G_R is shown in **Fig. 3.** G_R exhibits some of the properties of G, and it will be a useful tool in establishing by graph theory the conditions of solution and methods of solution of certain systems of equations and recurrence relations.

Lemma 3
Every cycle of G is composed of one or more of a finite basis of

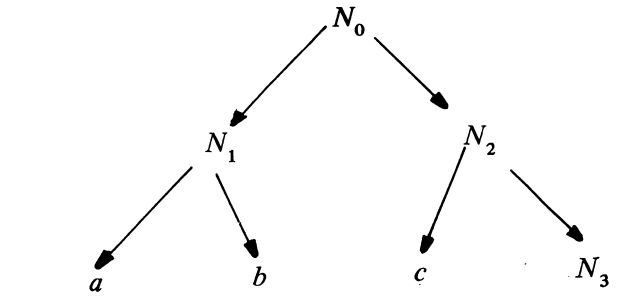

Fig. 1 Generation Tree for the Derivation $N_0 => abcN_3$

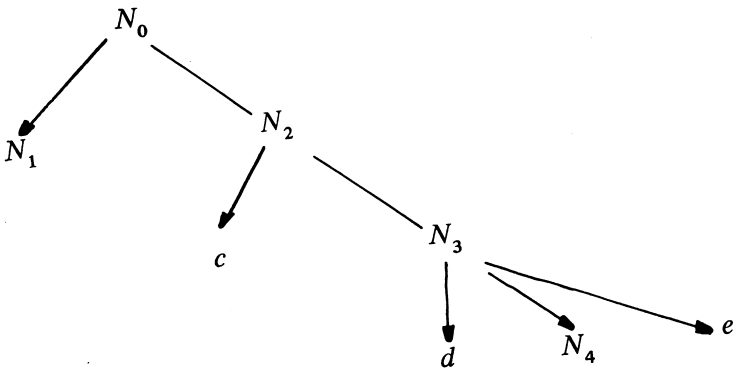

Fig. 2 Generation Tree for $N_0 \stackrel{c}{=>} N_1cdN_4e$

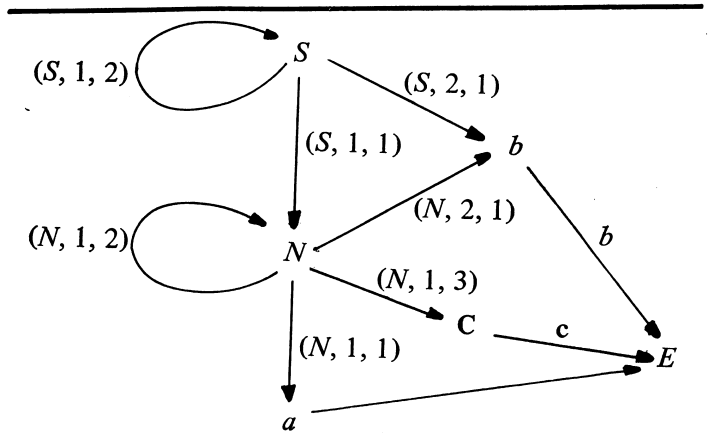

Fig. 3 Example of a graph G_R

cycles of G, repeated as required. These basic cycles correspond to the elementary circuits of G_R.

Proof
It is sufficient to note that there is a one-to-one correspondence between the chained derivations of G and the paths of G_R, and hence between the cycles of G and the circuits of G_R. The result then follows by using Lemma 2.

3. The net number of items yielded by a nonterminal

For any string u define $l(u)$ to be the 'length' or net number of items deposited on the stack by a terminal expansion of u. Then for any string v, $l(uv) = l(u) + l(v)$, by juxtaposition on the stack. Notice that $l(u)$ is a slight generalisation of what one might expect 'length' to mean. If $N -> \mathbf{CC}aaa$, then $l(N) = 1$ although the full effect is to replace the top three items of the stack by one and then to add three more. The section below on 'The gross number of items yielded by a nonterminal' considers this 'full effect' in more detail. Negative lengths are also possible, as for $N -> \mathbf{CCC}$, when $l(N) = -3$.

In general the string u (which may contain nonterminals) will have a set $\{l(u)\}$ of such l-values, possible infinite, and if $v -> w$ then $\{l(v)\} \supseteq \{l(w)\}$, since every terminal expansion of w is a terminal expansion of v. If $\{l(u)\}$ has a (least) upper bound, call it $l^+(u)$. If $\{l(u)\}$ has a (greatest) lower bound, then call it $l^-(u)$. The string u is termed l^--bounded, l^+-bounded, or just l-bounded if respectively $l^-(u)$ exists, $l^+(u)$ exists or both of these exist. A particular case of l-boundedness is 'l-uniqueness', when $l^+(u) = l^-(u)$, and $l(u)$ is unique.

Examples

1. $S -> N_1N_2$, $N_1 -> aN_1$, $N_1 -> a$, $N_2 -> \mathbf{C}N_2$, $N_2 -> \mathbf{C}$. A terminal string of S is $a^m\mathbf{C}^n$, for any integers $m, n >= 1$. Thus $l(S) = m - n$ and is completely unbounded.
2. $S -> aS$, $S -> a$. The terminal strings are a^m, $m >= 1$. $l(S) = m$, so that $l^-(S) = 1$. $l^+(S)$ does not exist.
3. $S -> \mathbf{C}S$, $S -> \mathbf{C}$. Here $l^-(S)$ does not exist, $l^+(S) = -1$.
4. $S -> N_1N_2$, $N_1 -> a$, $N_1 -> \mathbf{C}$, $N_2 -> b$, $N_2 -> \mathbf{C}$, $N_2 -> N_2b\mathbf{C}$. $l(N_1) = \pm 1 = l(N_2)$. Thus $\{l(S)\} = \{-2, 0, +2\}$, $l^-(S) = -2$, $l^+(S) = 2$. The last rule of N_2, though recursive, adds no further lengths to $\{l(N_2)\}$.
5. $S -> N_1N_2\mathbf{C}$, $N_1 -> aS\mathbf{C}$, $N_1 -> a$, $N_2 -> \mathbf{C}S\mathbf{B}$, $N_2 -> b$. Here $l(S)$, $l(N_1)$, and $l(N_2)$ are all unique.

Since the above definitions concerning $l(u)$ can apply to any N and thus to S, the terminology can be applied naturally to grammars as well.

Also for a chained derivation $N \stackrel{c}{=>} uxv$, define the 'lefthand length' $lhl(N \stackrel{c}{=>} uxv) \equiv l(u)$.

4. Recurrence relations for the $l(N)$

Consider a rule $N -> C_1 \ldots C_j \ldots C_p$ and let C_j be N_1. A derivation of N which starts with the above rule is unrestricted as to which derivation of N_1 C_j expands into. Thus all that can be said of $l(C_j)$ is that $l(C_j)$ is in $\{l(N_1)\}$. For any such derivation

$$l(N) = \sum_{j=1}^{P} l(C_j) \ .$$

Hence to find a $l(N)$ value choose known values for each of the $l(C_j)$ and add them. This is a kind of recurrence relation which may be written

$$l(N) <- \sum_{j=1}^{P} l(C_j) \ .$$

Moreover, by taking all the rules of G at once, the recurrence relations which arise determine all the $l(N)$ values, for all N.

By applying a number of rules it is easy to see that if $N => u = w_1 \ldots w_r \ldots w_s$, then

$$l(N) <- \sum_{r=1}^{s} l(w_r) \ .$$

Out of this comes a trivial theorem which helps to determine the unboundedness (or non-uniqueness) of G—it is sufficient to find just one unbounded (or non-unique) nonterminal.

Theorem 1

1. G is l^--bounded if and only if all N are l^--bounded.
2. G is l^+-bounded if and only if all N are l^+-bounded.
3. G is l-bounded if and only if all N are l-bounded.
4. G is l-unique if and only if all N are l-unique.

Proof

1. *Necessity*

Since G is admissible, then for every N not the same as S, there exist u, v, s such that $S => uNv => s$, so that there exists at

least one $l(S)$ value generated by $l(S) <- l(u) + l(N) + l(v)$. Now u, N and v can be expanded independently into strings of terminals, and so their lengths cannot always compensate each other to keep $l(S)$ l^--bounded unless they are all l^--bounded. Thus S and hence G is l^--bounded only if every N is l^--bounded.

Sufficiency

If all N are l^--bounded then S is and so is G.

2, 3 and 4: The proofs are analogous to 1.

5. The 'l-uniqueness' of G

l-uniqueness is a desirable property in the translator-inversion application in particular and for clarity of understanding in general. When G is l-unique the recurrence relations

$$l(N) <- \sum_{j=1}^{P} l(C_j)$$

become consistent equations in the $l(N)$. It is interesting to consider the converse, i.e. whether the $l(N)$ are unique if the equations are consistent. A set of linear equations does not in general have a unique solution (see, say, Griffiths, 1947). However the origin of these equations gives them a special form which does ensure uniqueness as the following theorem shows.

Theorem 2

Let $[L(N_1), L(N_2), \ldots L(N_h) \ldots L(N_n)]$ be a solution of the equations $\{l(N_h) = \sum l(C_{ij})\}$, (for all possible h, i and j) which arise from the production rules of G. Then this solution is unique, so that G is l-unique.

Proof

The proof is by induction on the number r of rule-applications necessary to expand N into a string s_N.

Consider a derivation $N => s_N$ in which just one rule-application is used. This rule must be $N -> C_{i1} \ldots C_{ij} \ldots C_{ip}$, for some i, in which each of the C_{ij} is a terminal. Because the length of the righthand side is constant $l(s_N) = L(N)$. Thus $l(N)$ is unique for all $N => s_N$ such that $r = 1$.

The induction step is as follows. Assume that for all r up to some $\mathbf{r} \geqslant 1$, every derivation $N => s_N$ which has r rule applications has the unique length $L(N)$. Now consider $N => s_N$, if any, with $\mathbf{r} + 1$ rule applications, where the derivation starts with $N -> C_{1i} \ldots C_{ij} \ldots C_{ip}$. Here some of the C_{ij} may be nonterminals. Let $r_1, \ldots r_j, \ldots r_p$ be the numbers of rule applications in the subtrees starting at $C_{i1}, \ldots C_{ij}, \ldots C_{ip}$ respectively. Then

$$\mathbf{r} + 1 = 1 + \sum_{j=1}^{P} r_j$$

so that for each j, $\mathbf{r} >= r_j$. Hence the assumption of the induction step is applicable and the lengths $l(C_{ij})$ are unique. Thus the length of the whole righthand side is unique and therefore must be $L(N)$.

Thus by induction $L(N)$ is unique however many rule applications are involved in a derivation $N => s_N$.

To determine G's l-uniqueness the steps are therefore:

1. Construct a tentative set of lengths $[L(N_1), \ldots L(N_h), \ldots L(N_n)]$ by using the simplest rules of G.
2. Substitute these in all the equations of G. If all the equations are satisfied, then G is l-unique.

6. l-boundedness conditions

The theorem of this section (Theorem 3) proves necessary and sufficient conditions for the lower and upper l-boundedness of G, and also gives a little more when these properties occur together. The proofs concerning upper and lower l-boundedness are analogous, and only l^--boundedness is dealt with in detail. Lemma 4 which precedes the theorem is written in terms of the l^- proof only.

The proof is by induction on the 'recursiveness' of a derivation $N => s_N$, which is a measure of its complexity defined as follows:

A generation tree of the derivation $N => s_N$ of a terminal string s_N, is 'q-recursive' where q is the number of recursive subtrees it contains, including itself if it is recursive.

A general string u is said to be q-recursive if attention is restricted to the subset of derivations of u in which no symbol of u is more than q-recursive, but in which at least one symbol of u is a q-recursive.

Now define $l_q(u)$, $l_q^+(u)$, $l_q^-(u)$ to be analogous to $l(u)$, $l^+(u)$, $l^-(u)$ but where only q-recursive derivations of u are considered.

From the above definitions it follows that if u is q-recursive then the symbols of u can be rearranged arbitrarily without affecting its q-recursiveness or, of course, its length. From Lemma 1 it is therefore reasonable to refer to the length of a cycle C as $l(C) \equiv l(u_c) + l(v_c)$, where there is a chained recursive derivation $N \overset{c}{=>} u_c N v_c$ of C. Similarly $l_q(C) \equiv l_q(u_c v_c)$. Also needed later is the lefthand-length $lhl(C) = l(u_c)$.

The proof of Theorem 3 relates all $l(N)$ values to $l_0^-(N)$, the lower bound of the finite set $\{l_0(N)\}$. Lemma 4 now provides the induction step.

Lemma 4

For any N, $l_q^-(N) >= l_0^-(N)$, $q > 0$, provided that for all r, $0 =< r < q$,

$l_r^-(N) >= l_0^-(N)$, and (C1)

that for all cycles C, $l_0^-(C) >= 0$. (C2)

Proof

Consider a q-recursive derivation $N => s_N$ of N. The aim is to construct from this another derivation $N => s'_N$ which is at most $(q-1)$-recursive and which is no greater in length. Then $l_q(N => s_N) >= l_r(N => s'_N) >= l_0^-(N)$ where $r =< q-1$, from (C1). By choosing $N => s_N$ so that $l_q(N)$ is minimal the relation becomes $l_q^-(N) >= l_0^-(N)$ as required.

$N => s'_N$ is constructed as follows. Let N' be the nonterminal in the base-node of one of the recursive subtrees of $N => s_N$, so that $N => uN'v$ for some strings u, v. Let N'' be an embedded instance of the same nonterminal N', so that $N' => u'N''v'$, say. Now form the tree for $N => s'_N$ by replacing the N' subtree by the N'' subtree. Because one recursive use of N' has been removed, $N => s'_N$ cannot be more than $(q-1)$-recursive, and similarly the string $u'v'$ cannot have any symbol which is more than $(q-1)$-recursive, so that

$$l(u'v') >= l_0^-(u'v'), \quad \text{from (C1)},$$
$$>= 0, \quad \text{from (C2)}.$$

Hence $l(N => s_N) = l(uN'v)$
$= l(uu'N''v'v)$
$= l(uN''v) + l(u'v')$
$>= l(N => s'_N)$, as required .

Theorem 3

1. G is l^--bounded and for every N $l^-(N) = l_0^-(N)$ if and only if $l_0^-(C) >= 0$ for each basic (i.e elementary) cycle C of G.
2. G is l^+-bounded and for every N $l^+(N) = l_0^+(N)$ if and only if $l_0^+(C) =< 0$ for each basic cycle C of G.
3. G is l-bounded and for every N $\{l(N)\} = \{l_0(N)\}$ if and only if $l_0(C) = 0$ for each basic cycle C of G.

Proof

Necessity

1. Suppose G is l^--bounded, and $l^-(N) = l_0^-(N)$, all N, but

there is an elementary cycle C for which $l_0^-(C) >= 0$ is false. Then there exists a derivation $N \stackrel{c}{=}> u_sNv_s$ of C such that $l(u_sv_s) < 0$. Choose some derivation $N => s_N$, and let it have length ${}_0l^N$. Then $N \stackrel{c}{=}> u_sNv_s$ defines a recurrence relation:

$$\begin{aligned} {}_{i+1}l^N &< - l(u_s) + {}_il^N + l(v_s),\ i >= 0 \\ &= l(u_sv_s) + {}_il^N < {}_il^N . \end{aligned}$$

Hence the integer sequence ${}_0l^N, {}_1l^N, \ldots {}_il^N \ldots$ has no lower bound so that N is not l^--bounded, contrary to hypothesis. Hence $l_0^-(C) >= 0$ for all C of G.

2. The proof is analogous to 1.
3. Apply 1 and 2 together.

Sufficiency

1. The proof is inductive using Lemma 4. It remains to prove (C2) and to show that (C1) holds for $q = 1$, $r = 0$. (C1) reduces to showing that $l_0^-(N)$ exists, which is true because $\{l_0(N)\}$ is finite. (C2) follows from the conditions of the theorem by Lemma 3.
2. The proof is analogous to 1.
3. Apply 1 and 2 together to show that G is l-bounded. The conditions of 1 and 2 also give $l^-(u_s) + l^-(v_s) >= 0 >= l^+(u_s) + l^+(v_s)$.
Hence $l^-(u_s) = l^+(u_s) = l(u_s) = 0$, and $0 = l^-(v_s) = l^+(v_s) = l(v_s) = l(u_sv_s)$ by the definition of upper and lower bounds.

Now Lemma 4 can be rephrased to show that if $N => s_N$ is q-recursive, $q > 0$, then one can find a q'-recursive s'_N such that $q' < q$ and $l(s'_N) = l(s_N)$. By applying this as many times as is necessary it follows that for any q-recursive s_N there is a 0-recursive s'_N with the same length. Hence

$$\begin{aligned} \{l(N)\} &\subseteq \{l_0(N)\} \\ &\subseteq \{l(N)\} \end{aligned}$$

by definition of $l_0(N)$. Therefore $\{l(N)\} = \{l_0(N)\}$.

7. Determination of the l-boundedness of G

The following computable steps can therefore be used to determine whether G is l-bounded. (It is only worth doing this if an application of Theorem 2 has shown G is not l-unique).

1. Inspect the rules of G to see if any N is obviously unbounded (Theorem 1).
2. If no unbounded N is apparent draw the graph G_R and find its elementary circuits. These identify a set of basic cycles C of G. (An algorithm for finding the elementary circuits of G_R is given in Weinblatt (1972).)
3. For each C take one of its chained recursive derivations $N \stackrel{c}{=}> u_sNv_s$ and for each symbol x_j in u_sv_s find $l_0^+(x_j)$ and $l_0^-(x_j)$. Hence determine $l_0^+(C)$ and $l_0^-(C)$ and apply Theorem 3. If G is completely bounded then the $l(N)$ are the $l_0(N)$, by Theorem 3.

8. The gross number of items yielded by a nonterminal

The preceding sections have dealt with the effects of depositing on the stack complete terminal strings derived from nonterminals. In this section the effect on the stack is considered at all stages during the deposition of a nonterminal's string. As an example let **C** have the extra concatenate function mentioned in the introduction and consider the rules $S -> \mathbf{C}Sa$, $S -> b$, which yield the strings $\mathbf{C}ba$, $\mathbf{CC}baa, \ldots \mathbf{C}^nba^n$, for all integers n. Then although all of these strings have length $(-n + 1 + n) = 1$. they successively combine more and more of the items already on the stack before depositing more. In contrast, the rules $S -> aS\mathbf{C}$, $S -> b$ yield strings $a^nb\mathbf{C}^n$, all having length = 1, but which successively add more and more items to the stack before matching concatenations take place.

These effects could be of real concern to the implementor of a stack handling grammars of this kind, because words of the language might overempty or overfill the stack. This section deals with conditions for grammars to be 'well behaved' in this way. However, a more severe effect than overemptying is discussed under 'Disturbance measurements' below.

It is useful to define $m^-(u)$ to be the 'gross' minimum length (in the generalised sense of the last section) which any terminal string derived from u can take on the stack at any time during or after its deposition. Similarly define $m^+(u)$ to be the 'gross' maximum length of any terminal string of u. Since $l^-(u)$ and $l^+(u)$ are the minimum and maximum lengths just after the deposition of u, $m^-(u)$ exists only if $l^-(u)$ exists, and $m^+(u)$ exists only if $l^+(u)$ exists. Also $m^-(u) =< l^-(u)$ and $m^+(u) >= l^+(u)$. In the remainder of this section the relevant l-bounds are always assumed to exist.

In order to evaluate $m^-(N)$ for all N in G, consider any rule $N -> C_1 \ldots C_j \ldots C_p => s_N = s(C_1)s(C_2) \ldots s(C_p)$ and consider the process of depositing s_N on the stack. It may be that $s(C_1)$ causes the length of s_N to be a minimum, so that certainly $m^-(N) =< m^-(C_1)$. However it may be $s(C_2)$ which causes the minimal length of s_N. In this case the whole of $s(C_1)$ is deposited before $s(C_2)$ is begun and so

$$m^-(N) =< l^-(C_1) + m^-(C_2) .$$

Similarly $m^-(N) =< l^-(C_1) + l^-(C_2) + m^-(C_3)$,

$$m^-(N) =< \sum_{j=1}^{P-1} l^-(C_j) + m^-(C_p)$$

and

$$m^-(N) =< \sum_{j=1}^{P-1} l^-(C_j) + l^-(C_p) = \sum_{j=1}^{P} l^-(C_j) .$$

However this last inequality can be disregarded since $m^-(C_p) =< l^-(C_p)$. Putting these inequalities together

$$m^-(N) =< \min_{1 \leq j \leq p} \left[\sum_{k=0}^{j-1} l^-(C_k) + m^-(C_j) \right] ,$$

where for convenience $l^-(C_0) \equiv 0$. One of these expressions arises for each production rule of N, so that $m^-(N)$ is the minimum of all these expressions:

$$m^-(N) = \min_{\text{all } R_i(N)} \left[\min_{1 \leq j \leq p} \left(\sum_{k=0}^{j-1} l^-(C_{ik}) + m^-(C_{ij}) \right) \right] .$$

The author has an algebraic algorithm for the solution of this set of equations, one for each N, together with proof of necessary and sufficient conditions for solution. However this approach does not give any understanding of what is happening. Given below is a more illuminating method, based on mapping the problem on to the graph G_R.

Assign to each arc (N, i, j) of G_R the arc length

$$\sum_{k=0}^{j-1} l^-(C_{ik})$$

which is the minimum lefthand length of the trivial chained derivation $N \stackrel{c}{=}> u_sxv_s$, where $u_s = C_{i1}C_{i2} \ldots C_{ij-1}$, $x = C_{ij}$, $v_s = C_{ij+1} \ldots C_{ip}$. Then (as in the proof of Lemma 3) for *any* chained derivation $N \stackrel{c}{=}> u_sxv_s$, the minimum lefthand length $(= l^-(u_s))$ is the sum of the arc lengths which make up the corresponding path in G_R. Also assign the arc length $m^-(t) = l(t)$ to each arc t. Now consider again the cause of N having a gross minimum length $m^-(N)$. This minimum is attained by the deposition of a particular terminal t of a particular terminal expansion of N, i.e. there exist u_s, v_s, such that $N \stackrel{c}{=}> u_stv_s$. Hence by the argument used before

$$m^-(N) = l^-(u_s) + m^-(t) ,$$

which is the length of an arc from N to E on G_R. Hence $m^-(N)$

is the minimum path length from N to E. Berge and Ghouila-Houri (1965) give the well known result that such a minimum path exists for every N if and only if there is no circuit with a negative arc length. This is equivalent to the necessary and sufficient condition that $lhl^-(C) = lhl_0^-(C) >= 0$ for every cycle C of G. This justifies the following theorem:

Theorem 4

1. $m^-(N)$ exists for every N of G if and only if $lhl_0^-(C) >= 0$ and $l_0^-(C) >= 0$ for all basic cycles C of G.
2. $m^+(N)$ exists for every N of G if and only if $lhl_0^+(C) =< 0$ and $l_0^+(C) =< 0$ for all basic cycles of G.

Evaluation of the $m^-(N)$ and $m^+(N)$ is straightforward since the arc lengths are functions of the l^- and l^+ respectively which are all known beforehand. For some possible methods see Berge and Ghouila-Houri, (pp. 180-182) and Iri (1969). Furthermore, the minimum path length problem has a unique solution (although more than one path may attain that length). Hence the original system of equations has a unique solution, because the grammar, the graph and the equations are in (1,1)-correspondence. It follows that if a tentative solution, say $[M^-(N_1), \ldots M^-(N_h) \ldots]$ does satisfy the equations, then the $m^-(N)$ exist and $M^-(N_h) = m^-(N_h)$ for each h. However the analogue of Theorem 2, when $m^-(N_h) = m^+(N_h)$ for each h, is not interesting because this equality is true only for a trivial subset of grammars.

Theorem 5

1. For every N, $m^-(N) =< 1$.
2. For every N, $m^+(N) >= -1$.

Proof

1. For any rule $N -> C_1 \ldots C_p$, $m^-(N) =< m^-(C_1)$. Thus as the lefthand branch of a generation tree of N is followed the m^- value for every subtree encountered cannot decrease. Finally the last nonterminal N_0 (say) is encountered where $N_0 -> t_0 C'_p \ldots C'_2$ (say).
   Then $m^-(N) =< m^-(N_0) =< m^-(t_0) =< \max_{\text{all } t} [m^-(t)] = \max_{\text{all } t} [l(t)] = 1$.
2. The argument is analogous to 1.

9. Disturbance measurements

The P_R permutation symbols are now discussed. Consider a single rule grammar $S -> P_1 ab$, where P_1 is an interchange. Here $l(S) = 2$, $m^-(S) = 1$, but before any symbols at all are deposited on the stack the top two cells are interchanged, thus interfering with material not deposited by this grammar. Thus the necessary condition for no underflow is that no previously deposited material should be disturbed. This is developed as follows.

Let $d(u)$ be the number of previously deposited cells disturbed at any stage during the loading of s, where $u => s$. Then for any derivation $N => s_N$, the maximum disturbance

$$d^+(N => s_N) = \max_{1 \leq j \leq p} [d^+(C_j) - \sum_{k=0}^{j=1} l^-(C_k)] ,$$

by an argument similar to that used in obtaining $m^-(N)$. So the maximum disturbance any derivation of N could make is

$$d^+(N) \equiv \max_{\text{all } R_i(N)} [\max_{1 \leq j \leq p} \{d^+(C_{ij}) - \sum_{k=0}^{j-1} l^-(C_{ik})\}] .$$

An analogous formula applies for $d^-(N)$, which is the minimum disturbance which can take place at any stage during the deposition of any terminal string derived from N. $d^-(N)$ can be negative.

Now let r_M be the maximum number of cells rearranged by any of the P_R operations of G. The disturbances made by individual symbols are zero for normal symbols, 1 or 2 for **C** and r_R for P_R, where $2 =< r_R =< r_M$.

Theorem 6

1. The $d^+(N)$ all exist if and only if the $m^-(N)$ exist. Moreover, for all N, $0 =< d^+(N) + m^-(N) =< r_M$ and $0 =< d^+(N)$.
2. The $d^-(N)$ all exist if and only if the $m^+(N)$ exist. Moreover, for all N, $0 =< d^-(N) + m^+(N) =< r_M$ and $d^-(N) =< r_M$.

Proof

1. Using the graphical method to solve the $d^+(N)$ equations, solutions exist if and only if the circuits of G_R have arc lengths not greater than zero, since maxima are being sought. But each arc length involved in a circuit has the same magnitude but opposite sign compared with the arc lengths in the $m^-(N)$ network, where the necessary and sufficient condition was 'circuit arc length not less than zero'. This proves the equivalence of the existence conditions.
   Call the $d^+(N)$ network D. Now consider the network M which differs from D in that the arcs t have lengths $-m^-(t)$. Then every arc length has the same magnitude but is opposite in sign compared with those in the $m^-(N)$ network.
   The values of $d(t) + l(t) = d^+(t) + m^-(t)$ are as follows, for each type of terminal t:

t	d	l	$d + l$
Normal	0	1	1
C	1	−1	0
	2	−1	1
P_R	r_R	0	r_R

   Thus in every case $0 =< d^+(t) + m^-(t) =< r_M$. Now let t_1 be the terminal at which $d^+(N)$ is attained, where t_1 is reached from N_1. Let t_2 be the terminal at which $m^-(N)$ is attained, where t_2 is reached from N_2. (t_1, t_2 and N_1, N_2 need not be distinct.) Using the maximal property of t_1 in network D,

$$\begin{aligned} d^+(N) &= \text{path length } (NN_1) + d^+(t_1) \\ &>= \text{path length } (NN_2) + d^+(t_2) \\ &= [\text{path length } (NN_2) - m^-(t_2)] + [d^+(t_2) + m^-(t_2)] \\ &>= -m^-(N). \end{aligned}$$

   Also using the maximal property of t_2 in network M,

$$\begin{aligned} -m^-(N) &= \text{path length } (NN_2) - m^-(t_2) \\ &>= \text{path length } (NN_1) - m^-(t_1) \\ &= d^+(N) - [d^+(t_1) + m^-(t_1)] \\ &>= d^+(N) - r_M. \end{aligned}$$

   The proof of the relation $d^+(N) >= 0$ is on the lines of Theorem 5.
2. The proof is analogous to 1 above.

As has been seen, evaluation of the $d^+(N)$ is analogous to the evaluation of the $m^-(N)$, and the testing of a tentative solution $[D(N_1) \ldots D(N_n)]$ by substitution is also valid.

10. Application to Metcalfe-Reeves translators

In Goodwin (1975) it was shown that sufficient conditions for a certain translator-inversion algorithm to work were that for each N the $l(N)$ and $d(N)$ values were unique, that $l(N) = 1$, and that $d(N) = d^+(N) = 0$. Proofs have been given in Theorem 2 and following Theorem 4 that these conditions can be verified by substitution of the desirable values in the l and d equations.

11. Application to compile time data stack checking

Only a sketch of the method is given. A high level language program can be regarded as defining the grammar of a generator

Existence of $m^-(S)$	$m^+(S)$	*Relation*	*Overflow*
YES, NO	YES	$m^+(S) =< L$	Never
YES	YES	$m^+(S) > L >= m^-(S)$	Certain or Possible.
YES	NO	$L >= m^-(S)$	Which is decidable.
NO	YES	$m^+(S) > L$	Certain or Possible.
NO	NO	—	Decidability unknown.

Fig. 4 Stack overflow conditions

Existence of $m^-(S)$	$m^+(S)$	*Relation*	*Underflow*
YES	YES, NO	$d^+(S) = 0$	Never
YES, NO	YES	$d^-(S) > 0$	Certain
YES	YES	$d^+(S) > 0 >= d^-(S)$	Certain or Possible.
NO	YES	$0 >= d^-(S)$	Which is decidable.
YES	NO	$d^+(S) > 0$	Certain or Possible.
NO	NO	—	Decidability Unknown.

Fig. 5 Stack underflow conditions

mechanism which at run time outputs a string of operands or terminals which are loaded on an arithmetic stack. (This is the POP2 kind of stack or (say) a KDF9 nesting store, to be distinguished from an ALGOL-type data storage stack which is not intended.) Arithmetic operators are also issued and are treated as special terminals, e.g. '+' and '−' are special cases of C. The method applies most directly to languages in which function definitions can be separate entities such as in FORTRAN or POP2. Then function names identify nonterminals of the grammar, as do the statements of each function, e.g. $F -> S_1S_2 \ldots S_j \ldots S_p$ might be a function consisting of assignment statements S_j. Or an S_j might be a function call, or of the form IF E_j THEN S_{j1} ELSE S_{j2}. This would give rise to the alternative rules:

1. $F -> S_1S_2 \ldots S_{j-1}E_j$
2. $F -> S_1 \ldots S_{j-1}S_{j1}S_{j+1} \ldots S_p$
3. $F -> S_1 \ldots S_{j-1}S_{j2}S_{j+1} \ldots S_p$.

Each assignment statement would correspond to one rule of the grammar and would there be expressed in Reverse Polish form, ending with a store operator which is another special case of C. Again function calls could be included. The treatment of the l, m and d quantities above is sufficient to allow functions which take from the stack an arbitrary fixed number of parameters and place on it any fixed number of results. So-called variadic functions in which the number of results or parameters varies at run time could not be allowed.

Loops as defined by backward GOTO statements or DO-type statements are allowed so long as their stack length is zero. This is always true in FORTRAN since the elementary stack altering operation is the assignment statement whose length is zero. Forward GOTO statements, if part of a condition, lead to the function in which they occur having more than one rule in the grammar.

Let the finite allowable stack length be L. Then analysis of this derived grammar at compile time could answer the overflow questions according to the table in **Figs. 4** and **5**.

The 'Certain or possible' cases in Fig. 4 need explanation. The maximum gross length $m^+(S)$ may be attained during the deposition of all strings s_s of the language, in which case the corresponding program is bound to fail; on the other hand strings may exist whose individual gross length is always far short of $m^+(S)$, i.e. depending on its data the execution of the program may well not demand the use of the whole physical stack. These two different types of grammars can be distinguished by an algorithm when the number of relevant net and gross lengths which strings s_N can take is finite. Conditions are given when this is true, but for brevity here the proof is deferred to a later article. When these conditions do not hold, it is not known whether an algorithm exists, although the author conjectures that it does. A similar discussion applies to Fig. 5.

These overflow and underflow results could be used simply to reject or accept the program at compile time. Alternatively they might be used to set the value of L, or as an automatic method of determining when to insert coding to check for stack overflow or underflow. Of course, these basic ideas are well known, and originality is only claimed for the systematic treatment above.

Acknowledgement
The author is indebted to Professor C. M. Reeves for reading a draft of this article and making valuable constructive criticisms.

References

BERGE, C. and GHOUILA-HOURI, A. (1965). *Programming, Games and Transportation Networks*, London: Methuen.
GOODWIN, D. T. (1975). An Algorithm for Inverting Certain Translators of Context-Free Languages, *The Computer Journal*, Vol. 18, pp. 349-354.
GRIFFITHS, L. W. (1947). *Introduction to the Theory of Equations*, p. 209, New York: Wiley.
IRI, M. (1969). *Network Flow, Transporation and Scheduling*, pp. 178-188, New York: Academic Press.
METCALFE, H. H. (1964). A Parameterised Compiler based on Mechanical Linguistics. *Annual Review in Automatic Programming*, Vol. 4, pp. 125-165 (R. Goodman, Editor) Pergamon Press
REEVES, C. M. (1967). Description of a Syntax-directed Compiler, *The Computer Journal*, Vol. 10, pp. 244-255.
WEINBLATT, H. (1972). A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph. *JACM*, Vol. 19, pp. 43-56.