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 \( P_1, P_2, \ldots, P_n \) may be used. Any instance of \( P_n \) in the string is not loaded, but instead causes a particular (non-identity) permutation of a fixed number \( r \) 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 \( P_n \) 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 \( cbXcaXC \) is processed from left to right as follows:

<table>
<thead>
<tr>
<th>Stack contents (top cell at right)</th>
<th>String to be loaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>cbXcaXC</td>
</tr>
<tr>
<td>c</td>
<td>bXcaXC</td>
</tr>
<tr>
<td>b</td>
<td>CaXC</td>
</tr>
<tr>
<td>be</td>
<td>aXC</td>
</tr>
<tr>
<td>be a</td>
<td>XC</td>
</tr>
<tr>
<td>a be</td>
<td>C</td>
</tr>
<tr>
<td>abc</td>
<td>—</td>
</tr>
</tbody>
</table>

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 \( a, b, c \ldots \) 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 \( N_1, N_2, \ldots, N_n \), are used to denote all the non-terminals of \( G \).

3. A finite number of 'production rules' each of which is of the form

\[ N \rightarrow C_1C_2\ldots C_j\ldots C_p, \]

where each of the \( C_j \) may be either a single terminal or a single nonterminal. There may be a number of rules with \( N_p \), say, on the left hand side. These are called 'the rules of \( N_p \)'. 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

\[ N \rightarrow C_{i1}C_{i2}\ldots C_{ij}\ldots C_{ip}. \]

(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 x. u^n denotes the string uu...u where u is repeated n times.

If a string w of nonterminal N can be 'expanded' by an application of a rule R(N). Let w_0 = u_0N_0v_0. Having chosen i, this instance of N_0 is replaced in w_0 by

\[ C_{i1}C_{i2}...C_{ij}...C_{ip} \]

This operation is written

\[ w_0 = u_0N_0v_0 \rightarrow u_0C_{i1}C_{i2}...C_{ij}...C_{ip}v_0 = w_i \text{ (say)} \]

Similarly if w_i contains an instance of N_1 (say), not necessarily distinct from N_0, a rule of N_1 can be used to expand w_i into w_{i+1}. This operation w_{i+1} \rightarrow w_0 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_i in one or more steps, for any k > 0. w_0 \rightarrow 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}...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+1} 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 \rightarrow w_1 \rightarrow ... \rightarrow w_k \rightarrow ... \rightarrow w_q, in which for each k, N_{k+1} is chosen only out of the C_{i1}...C_{ip} in w_{k+1} which arise from the expansion of N_k. Such a derivation w_0 \rightarrow w_q is here called a 'chained derivation' and is denoted by w_0 = \rightarrow w_q. It follows that there exist u_{0}, u_{1}, u_{2},... such that

\[ w_0 = u_0N_0v_0 = \rightarrow u_0u_1N_1v_1v_0 = \rightarrow u_0u_1u_2N_2v_2v_1v_0 = \rightarrow \ldots \rightarrow u_0u_1u_2\ldots u_qN_qv_qv_0 = w_q \]

A derivation N = \rightarrow w_q 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,..., w_q. Branches leave a node N_r to arrive at the symbols into which N_r is expanded by a rule application.

**Example:**

Given rules N_0 \rightarrow N_1N_2, N_1 \rightarrow ab, N_2 \rightarrow cN_3, N_3 \rightarrow dN_4, then N_0 \rightarrow abcdN_4 has the generation tree shown in Fig. 1. The chained derivation N_0 = > N_1cdN_4 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 = > 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

\[ N_1(i_1,j_1)N_2(i_2,j_2)\ldots N_k(i_k,j_k)\ldots N_m(i_m,j_m) \]

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

1. for each r < m, C_{ir}r = N_{r+1}, in the i_rth rule of N_r, and
2. C_{im}m = N_1, in the i_mth rule of N_m.

Given such a sequence, any cyclic permutation such as

\[ N_2(i_2,j_2)\ldots N_k(i_k,j_k)\ldots N_m(i_m,j_m)N_1(i_1,j_1) \]

identifies the same cycle.

Let a cycle C' be \[ N_1(i_1,j_1)\ldots N_k(i_k,j_k)\] and let C and C' have a common nonterminal N_1 = N_1'. Then the sequence

\[ N_1(i_1,j_1)\ldots N_m(i_m,j_m)N_1(i_1,j_1)\ldots N_k(i_k,j_k) \]

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

Now in R_i(N_r) let \[ v_r = C_{i_r}C_{i_{r+1}}...C_{i_{p_r}} \], and let

\[ v_r = C_{i_{r+1}}...C_{i_p} \]

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

\[ N_1 = > u_0u_{1}u_2\ldots u_{r-1}N_rv_{r-1}v_{r-2}\ldots v_1v_0v_{r-1}\ldots v_{r+1}v_r \]

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 = > u_0N_rv_r of a given cycle C, the strings u_r, v_r 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 Ghoulia-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_r in each rule R_i(N). This arc may be labelled (N, i_r). 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 \[ R_3(N) \]

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
have a set \( \{ l(u) \} \) of such \( l \)-values, possible infinite, and if \( v \rightarrow w \) 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. \( S \rightarrow N_1 N_2, N_1 \rightarrow a N_1, N_1 \rightarrow a, N_2 \rightarrow C N_2, N_2 \rightarrow C \). A terminal string of \( S \) is \( a^n C c^n \), for any integers \( m, n \geq 1 \). Thus \( l(S) = m - n \) and is completely unbounded.

2. \( S \rightarrow a S, S \rightarrow a \). The terminal strings are \( a^m, m \geq 1 \). \( l(S) = m \), so that \( l^-(S) = 1 \). \( l^+(S) \) does not exist.

3. \( S \rightarrow C S, S \rightarrow C \). Here \( l^-(S) \) does not exist, \( l^+(S) = -1 \).

4. \( S \rightarrow N_1 N_2, N_1 \rightarrow a, N_1 \rightarrow C, N_2 \rightarrow b, N_2 \rightarrow C \). \( l(N_1) = \pm 1 = l(N_2) \). 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. \( S \rightarrow N_1 N_2 C, N_1 \rightarrow a C, N_1 \rightarrow a, N_2 \rightarrow C S \). \( N_2 \rightarrow b \). 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.

For a chained derivation \( N \equiv N \equiv u w v \equiv l(u) \equiv \text{the 'left hand length'} \).

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

Consider a rule \( N \rightarrow C_1 \ldots C_p \) and let \( C_j \) be \( N_1 \). A derivation of \( N \) which starts with the above rule is unrestricted as to which derivation of \( N \), \( C_i \) 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 \( N \equiv u = w_1 \ldots w_w \) then

\[
l(N) = \sum_{r=1}^{w} l(w_r).
\]

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 \( S \rightarrow u v \rightarrow s \), so that there exists at

The Computer Journal

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 \( v, l(uw) = l(u) + l(v) \), by juxtaposition on the stack. Notice that \( l(u) \) is a slight generalisation of what one might expect 'length' to mean. If \( N \rightarrow C C a a a a a \), 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 \( N \rightarrow C C C C C C \), when \( l(N) = -3 \).

In general the string \( u \) (which may contain nonterminals) will
least one \( I(S) \) value generated by \( I(S) = -I(u) + I(N) + I(v) \). Now \( u, N \) and \( v \) can be expanded independently into strings of terminals, and so their lengths cannot always compensate each other to keep \( I(S) \) \( I^- \)-bounded unless they are all \( I^- \)-bounded. Thus \( S \) and hence \( G \) is \( I^- \)-bounded only if every \( N \) is \( I^- \)-bounded.

**Sufficiency**

If all \( N \) are \( I^- \)-bounded then \( S \) is and so is \( G \). 2, 3 and 4: The proofs are analogous to 1.

5. The '\( I^-\)uniqueness' of \( G \)

\( I^-\)uniqueness is a desirable property in the translator-inversion property in particular and for clarity of understanding in general. When \( G \) is \( I^-\)unique the recurrence relations

\[
I(N) < - \sum_{j=1}^{k} I(C_j)
\]

become consistent equations in the \( I(N) \). It is interesting to consider the converse, i.e. whether the \( I(N) \) 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 \( \{L(N_1), L(N_2), \ldots, L(N_n)\} \) be a solution of the equations \( \{l(N_i) = \sum_{j} l(C_{ij})\} \), (for all possible \( h, i \) and \( j \)) which arise from the production rules of \( G \). Then this solution is unique, so that \( G \) is \( I^-\)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 \( N \Rightarrow s_N \) in which just one rule-application is used. This rule must be \( N \Rightarrow C_{11} \ldots C_{ip} \), for some \( i \), in which each of the \( C_{ij} \) is a terminal. Because the length of the right-hand side is constant \( I(s_N) = L(N) \). Thus \( I(N) \) is unique for all \( N \Rightarrow s_N \), such that \( r = 1 \).

The induction step is as follows. Assume that for all \( r \) up to some \( r \), every derivation \( N \Rightarrow s_N \) has \( r \) rule applications has the unique length \( L(N) \). Now consider \( N \Rightarrow s_N \), if any, with \( r \) + 1 rule applications, where the derivation starts with \( N \Rightarrow C_{11} \ldots C_{ip} \). Here some of the \( C_{ij} \) may be nonterminals. Let \( r_1, \ldots, r_p \) be the numbers of rule applications in the subtrees starting at \( C_{i1}, \ldots, C_{ip} \) respectively. Then

\[
r + 1 = 1 + \sum_{j=1}^{p} r_j
\]

so that for each \( j \), \( r > r_j \). Hence the assumption of the induction step is applicable and the lengths \( I(C_{ij}) \) are unique. Thus the length of the whole right-hand 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 \( N \Rightarrow s_N \).

To determine \( G \)'s \( I^-\)uniqueness the steps are therefore:

1. Construct a tentative set of length \( \{L(N_1), L(N_2), \ldots, L(N_n)\} \) using the simplest rules of \( G \).
2. Substitute these in all the equations of \( G \). If all the equations are satisfied, then \( G \) is \( I^-\)unique.

6. \( I^-\)boundedness conditions

The theorem of this section (Theorem 3) proves necessary and sufficient conditions for the lower and upper \( I^-\)boundedness of \( G \), and also gives a little more when these properties occur together. The proofs concerning upper and lower \( I^-\)-boundedness are analogous, and only \( I^-\)-boundedness is dealt with in detail. Lemma 4 which precedes the theorem is written in terms of the \( I^- \) proof only.
there is an elementary cycle C for which \( l(C) > 0 \) is false. Then there exists a derivation \( N \Rightarrow u_N v_N \) of C such that \( l(u_N v_N) < 0 \). Choose some derivation \( N \Rightarrow s_N \), and let it have length \( r(N) \). Then \( N \Rightarrow u_N v_N \) defines a recurrence relation:

\[
i+1 = l(i) - l(u_i) + l(v_i) + l(\varepsilon), \quad i > 0 = l(u_{\varepsilon}) + l(v_{\varepsilon}) < l(i).
\]

Hence the integer sequence \( l(N), l(N), \ldots, l(N) \ldots \) has no lower bound so that \( N \) is not \( l^*-\)bounded, contrary to hypothesis. Hence \( l(N) > 0 \) for all \( C \) of \( G \).

2. The proof is analogous to 1.

3. Apply 1 and 2 together.

**Sufficiency**

The proof is inductive using Lemma 4. It remains to prove (C2) and to show that (C1) holds for \( q = 1, \tau = 0 \). (C1) reduces to showing that \( l(N) \) exists, which is true because \( \{l(N)\} \) 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 \( l^-(u_i) + l^+(u_i) > 0 \). Hence \( l^-(u_i) = l^+(u_i) = l(u_i) = l(v_i) \) by the definition of upper and lower bounds.

Now Lemma 4 can be rephrased to show that if \( N \Rightarrow s_N \) is \( q \)-recursive, \( q > 0 \), then one can find a \( q \)-recursive sequence \( s_N \) 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 sequence \( s_N \) there is a 0-recursive sequence \( s_N \) with the same length. Hence \( \{l(N)\} \subseteq \{l(N)\} \subseteq \{l(N)\} \) by definition of \( l(N) \). Therefore \( \{l(N)\} = \{l(N)\} \).

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). Theorem 1.

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 \( G_N \) and find its elementary circuits. These identify a set of basic cycles \( C \) of \( G \). (An algorithm for finding the elementary circuits of \( G \) is given in Weinblatt (1972).)

3. For each \( C \) take one of its chained recursive derivations \( N \Rightarrow u_N v_N \) and for each symbol \( x_i \) in \( u_N v_N \) find \( l(C) \) and \( l(x_i) \). Hence determine \( l(C) \) and \( l(C) \) and apply Theorem 3. If \( G \) is completely bounded then the \( l(C) \) is the \( l(C) \), 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 \( S \rightarrow CSa, S \rightarrow b \), which yield the strings \( Cb, Cbab, \ldots, C^*ba^* \), for all integers \( n \). Then although all of these strings have length \( n + 1 = 1 \). They successively combine more and more of the items already on the stack before depositing more. In contrast, the rules \( S \rightarrow aSc, S \rightarrow b \) yield strings \( a^*bC^*a^* \), 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 overemply or overfill the stack. This section deals with conditions for grammars to be 'well behaved' in this way. However, a more severe effect than overemplying 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 \( m(u) = m(u) \) and \( m(u) \neq m^+(u) \). 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 \( N \Rightarrow C_1 \ldots C_j \ldots C_p \Rightarrow s_N \), \( s_N \Rightarrow s(C_1)s(C_2) \ldots s(C_p) \) and consider the process of depositing \( s_N \) on the stack. It may be that \( s(C_i) \) causes the length of \( s_N \) to be a minimum, that certain \( m(N) = m^+(C_i) \). However it may be \( s(C_j) \) which causes the minimal length of \( s_N \). In this case the whole of \( s(C_j) \) is deposited before \( s(C_j) \) is begun and so

\[
m(N) = l^+(C_i) + m^+(C_j) + m^+(C_p) + m^+(C_j) \quad \text{Similarly} \quad m(N) = l^+(C_i) + m^+(C_j) + m^+(C_p) + m^+(C_j) + l^+(C_p).
\]

and

\[
m(N) = l^+(C_i) + l^+(C_j) = \sum_{j=1}^{p-1} l^-(C_j) + l^+(C_j) = \sum_{j=1}^{p-1} l^+(C_j).
\]

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

\[
m(N) = \min_{1 \leq i \leq p} \left[ \sum_{k=0}^{i-1} l^-(C_k) + l^+(C_k) \right] + \min_{1 \leq i \leq p} \left[ \sum_{k=0}^{i-1} l^-(C_k) + l^+(C_k) \right] + \min_{1 \leq i \leq p} \left[ \sum_{k=0}^{i-1} l^-(C_k) + l^+(C_k) \right] \quad \text{where for convenience} \quad l^+(C_0) = 0.
\]

One of these expressions arises for each production rule of \( N \), so that \( m(N) \) is the minimum of all these expressions:

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

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

\[
j \geq 1 \sum_{k=0}^{j-1} l^-(C_k) + l^+(C_k) \quad \text{which is the minimum lefthand length of the trivial chained derivation} \quad N \Rightarrow u_N v_N \text{ where} u_i = C_i \ldots C_i \ldots C_i, v_i = C_i \ldots C_i \ldots C_i \text{. Then as (in the proof of Lemma 3)} \text{ for any chained derivation} \quad N \Rightarrow u_N v_N \text{ the minimum lefthand length} \quad l(i) = l(i) \text{ to each arc} \text{. Now consider again the cause of} \text{ having a gross minimum length} \quad m(N) \text{. This minimum is attained by the deposition of a particular terminal} \text{ of a particular terminal expansion of} \text{ of} \text{ i.e.} \text{ there exist} \text{ such that} \quad N \Rightarrow u_N v_N \text{. Hence by the argument used before} \quad m(N) = l(i) + m(i) \text{ , which is the length of an arc from} \text{ to} \text{ on} \text{. Hence} \quad m(N) \text{ }
\]
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 $l^-(C) = l^+(C) > 0$ 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 $l^+(C) > 0$ and $l^+(C) > 0$ for all basic cycles $C$ of $G$.

2. $m^+(N)$ exists for every $N$ of $G$ if and only if $l^-(C) < 0$ and $l^-(C) < 0$ for all basic cycles $C$ 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 $[M^-(N_1), \ldots, M^-(N_k), \ldots]$ does satisfy the equations, then the $m^-(N)$ exist and $M^-(N_k) = m^-(N_k)$ for each $k$. However the analogue of Theorem 2, when $m^-(N_k) = m^-(N_k)$ for each $k$, is not interesting because this equality is true only for a trivial subset of grammars.

**Theorem 5**

1. For every $N$, $m^-(N) = < 1.0$

2. For every $N$, $m^+(N) = > -1.0$

**Proof**

1. For any rule $R \rightarrow C_1 \ldots C_p$, $m^-(N) = m^-(C_i)$. 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 $N_0 \rightarrow C_0 C_p \ldots C_2$ (say).

Then $m^-(N) = < m^-(N_0) = < m^-(t_0) = < \max_{t \in t} [m^-(t)] = \max_{t \in t} [l(t)] = 1.0$

2. The argument is analogous to 1.

9. Disturbance measurements

The $P_R$ permutation symbols are now discussed. Consider a single rule grammar $S \rightarrow P_{1}ab$, 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 $N = s_R$, the maximum disturbance

$$d^+(N = s_R) = \max_{\emptyset \in j \in \emptyset} \left( \sum_{k=0}^{j} l^-(C_i) \right),$$

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

$$d^+(N) = \max_{R \in R(N)} \left( \sum_{k=0}^{j} l^-(C_i) \right).$$

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 $R_M$ for $P$. 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 $G$ 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 $t$:

<table>
<thead>
<tr>
<th>$t$</th>
<th>$d$</th>
<th>$l$</th>
<th>$d + l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$P_R$</td>
<td>$r_R$</td>
<td>0</td>
<td>$r_R$</td>
</tr>
</tbody>
</table>

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 $N_2$. ($t_1, t_2, t_1, N_1, N_2$ need not be distinct.) Using the maximal property of $t_1$ in network $D$,

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

$$> = -m^-(N).$$

Also using the maximal property of $t_2$ in network $M$,

$$-m^-(N) = \text{path length } (NN_1) - m^-(t_2) > = \text{path length } (NN_1) + m^-(t_1) = d^+(N) - [d^+(t_1) + m^-(t_1)]$$

$$> = d^+(N) - r_M.$$

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.
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 $s_i$ 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 $s_i$ 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


