43

CORRECTNESS OF THE ALGOL PROCEDURE ASKFORHAND

by

Ralph L. London*

Computer Sciences Technical Report #50

November, 1968

*Computer Sciences Department and
Mathematics Research Center

Table of Contents

NOTE: Inside the back cover appears a second, but unbound, set of Appendices that can be removed for easier reference.

ABSTRACT

Reasons are given to justify the writing of this particular proof of correctness. They include (i) illustration of some new techniques of proof, (ii) experimentation with a method of presenting a proof and (iii) presentation of an example of a successful proof to encourage more such proofs of other programs. The specific example is an Algol procedure ASKFORHAND which is supposed to read a bridge hand from a teletype. That task is defined and a proof is given that the procedure properly performs this task.

CORRECTNESS OF THE ALGOL PROCEDURE ASKFORHAND

by

Ralph L. London

Introduction

This paper is one result in an on-going research effort aimed at developing methods for proving the correctness of realistic computer programs. Such programs are ones that are encountered in actual practice, ones for which there is some interest in knowing that they are correct and ones that are complex enough so that their correctness is not immediately apparent.

The entire proofs of two of the several programs proved in this effort were presented in previous papers. One of these programs makes the opening bid for bridge hands [4]; the second faithfully performs the four arithmetic operations of interval arithmetic [2].

A proof of the program ASKFORHAND is here presented for three main reasons. First, since ASKFORHAND is significantly different from the other programs, new and different techniques of proof are needed. ASKFORHAND has a double loop (the other two programs have essentially no loops), and thus the utility of mathematical induction in proofs of correctness is demonstrated in this research effort for the first time.

ASKFORHAND is also different since it is basically two sections of code carefully interleaved to gain both space and efficiency. As a result, there is a large number of go to statements, and the flow of control transfers from one part of the code to another with high frequency. To follow the many transfers of control and to show that neither section interferes with the other, two tables of information have been prepared from the code (Appendices 2 and 3). These are a table showing the places where each variable might be altered and a table showing the flow of control between program labels. Compared to working directly from the code, these tables permit easier demonstrating that variables are unchanged between two points or changed only in known ways, and that flow must get from one point to another under given assumptions.

The first table has been used in limited ways before, but the second table is entirely new. Also new with the proof of ASKFORHAND is the extensive use of both tables.

The second reason for writing the current proof is that the proof is intended as an experiment in presenting a complex proof of validity. Can a complete proof be understood if it is a series of lemmas where each lemma is proved by giving the necessary reasons in a standard prose fashion? Should other methods of presentation be devised?

Since ASKFORHAND is a complicated program, the proof, although comprehensible, is long and involved. The object of a proof, however, is much more than explaining how the program works. The proof must demonstrate why it works and must answer the many questions and doubts that may arise. Issues of presentation are therefore relevant.

As with any lengthy proof, certain details have been "left to the reader." Hopefully this will not detract from the completeness and credibility of the entire proof. These omissions are also part of the experiment: what details can be safely omitted and what details are essential to the reader's ability to accept the proof?

Third, the proof is an example of a successful proof of a realistic program. It should encourage others to prove their own programs and those of others, although it is doubtful that a reader will be able immediately to prove correct an arbitrary realistic program from this example alone. Some techniques are illustrated, but more work is needed to abstract the essential features of the techniques. Later papers are planned for this purpose. At this stage, however, examples of successful proofs, particularly proofs of realistic programs, are one way to answer the often asked, but seldom answered question, "How does one prove that a program does what it is supposed to do?"

Another paper [3] discusses the meaning, rationale and importance of program proving, as well as techniques and strategies of program proving. The paper also summarizes the salient features of the opening bid proof, the interval arithmetic proof, the current proof and two proofs by other workers [1, 5]. Its main conclusion is that it is indeed feasible and realistic to prove the correctness of actual computer programs. More programs can and should be proved. The already existing proofs are models indicating how one might proceed in this creative process.

We now turn to ASKFORHAND and its definition.

Definition of ASKFORHAND

ASKFORHAND is a (Burroughs B5500) Algol procedure taken from a bridge bidding program written by A. I. Wasserman [6] for purposes unrelated to program proving. Only after it was written and running was it considered as a convenient candidate for proving correctness.

The code for ASKFORHAND appears in Appendix 1. It is an accurate copy of the running version except that input/output statements have been replaced by comments, and some input/output variables have been deleted entirely. No changes have been made in the program logic, though, and no changes have been made in the code which would have made the proof easier. The goal has been to take the program as given and unalterable and then to prove that it is correct. Of course, the code was changed when minor but outright errors were detected in the attempt at proof. After an error was discovered the proof was begun anew.

ASKFORHAND will first be informally defined in order that the reader may understand generally what the Algol procedure is to do. Then a formal definition will be given; the proof is based on the formal definition.

ASKFORHAND is to read a bridge hand from a teletype as four separate lines of input. Each line represents the cards of one suit, and the order of the lines is the standard bridge ordering: spades, hearts, diamonds, clubs, starting with spades. The cards of a suit are represented as they are in bridge books, for example, A Q J 9 X or even ACE QUEEN JACK NINE X. The hand is "read" by storing it in the first thirteen locations of the array MYHAND, using the integers from 1 to 52 as the internal data.

To input the hand consisting of king, jack, small spade, king, queen, seven, six, two small hearts, no diamonds, and king, queen, jack, deuce of clubs, one might type (where "←" denotes the end-of-line character for the teletype):

```
K J X←                        KING J X←

K Q 7 6 X X←                     K QUN SIX SEVEN X X←
                   or
                  even
-←                             VOID←

K Q J 2←                     KINGOFCLUBS, 2 J. QUEEEENOFHARTS←
```

Cards must be separated by some punctuation or space. Only the first character of a card, or if necessary the first two characters (e.g., SIX and SEVEN), are checked to obtain an identification and the rest of the characters are ignored.

This much of ASKFORHAND is worth proving, but the additional capabilities of error recovery are what make ASKFORHAND an interesting and challenging program to prove. If only 12 cards are given, there is special provision for easily adding the 13th card without retyping the entire hand. Similarly if 14 cards are given, it is easy to delete the extra one. When 12 or 14 cards have been given, a card is specified for addition or deletion, for example, as ACE OF SPADES or SPADE ACE or ACE SPADES. Again only the first one or two characters of a word are needed. Fewer than 12 or more than 14 cards requires that the hand be retyped. An illegal symbol within a suit allows just that suit to be retyped. Finally several errors can be corrected in a single call of ASKFORHAND before the procedure will stop attempting to read the hand and exit.

The task and processing that ASKFORHAND is to perform is now formally defined. As previously stated the four ordered lines of input correspond to the four ordered suits. From the four suits a total of precisely 13 cards is expected although other totals are handled by the error recovery facilities. The teletype representation of a single suit is defined by a Backus Naur Form definition and additional qualifications that are in English:

```
<RANK ID> ::=A|K|Q|J|1|2|3|4|5|6|7|8|9|X|N|E|SE|SI|FI|FO|TH|TW|T
<SUIT HOLDING> ::=<RANK ID>|<SUIT HOLDING>b<RANK ID>
<VOID> ::=V|-
<SUIT INPUT> ::=<VOID>←|←|<SUIT HOLDING>←
```

"b" denotes the character blank. "←" denotes the end-of-line character or group mark for the teletype. The <RANK ID>s correspond respectively to Ace, King, Queen, Jack, 10, 2, 3, 4, 5, 6, 7, 8, 9, small (standard bridge notation), Nine, Eight, SEven, SIx, FIve, FOur, THree, TWo, Ten. Naturally 9 and Nine are the same card and similarly for the other pairs.

The additional qualifications are (i) the maximum number of total characters in a <SUIT INPUT> is 80, i.e. one line of teletype input, (ii) the maximum number of <RANK ID>s in a <SUIT HOLDING> is 13, (iii) a card (not just a <RANK ID>) may appear at most once in a <SUIT HOLDING>, and (iv) the number of Xs in a <SUIT HOLDING> is less than the lowest ranked card minus one. The first restriction is a property of the teletype and is part of the definition of the procedure GETNEXTLINE (see next section). Violation of the other three restrictions will cause the error recovery mechanism to be invoked.

The input under these restrictions constitutes *simplified input*. The first sample input is simplified. *Non-simplified* or *extended input* (the second sample) allows the following less restrictive modifications. Let blank (b), period (.) and comma (,) be the non-group mark delimiters. Adjacent <RANK ID>s must always be separated by precisely one blank in simplified input. But here any number of non-group mark delimiters is allowed. Zero or more such delimiters may appear before the first <RANK ID> and after the last <RANK ID>. Leading zeros on a <RANK ID>

are ignored. A <RANK ID> may be followed by any string of characters not containing a non-group mark delimiter, for example ACE or ACX or AQJ or ACEOFTRUMP all represent an ace. There is one exception: if "T" is a <RANK ID>, then the first character of any immediately following string of characters must not be "W" or "H".

While expanded input can be similarly expressed in BNF notation, it is confusing to do so. The English description is adequate and more easily comprehended.

The internal representations of the cards are the integers 1 to 52 according to the formula 13 * SUITVALUE + RANKVALUE where

SUIT	S	H	D	C
SUITVALUE	3	2	1	0

and

RANK	A	2	3	4	5	6	7	8	9	10	J	Q	K	X
RANKVALUE	1	2	3	4	5	6	7	8	9	10	11	12	13	see below

.

Thus the jack of hearts is represented by 13 * 2 + 11 = 37 and the ace of hearts by 13 * 2 + 1 = 27. The rank value of X is determined by the rule that the first X encountered in a suit has rank value 2, the second X has rank value 3, etc., up to a maximum rank value of 9.

The error recovery capabilities will be described formally later. The following main theorem will be proved:

Theorem: All cards of the hand are read, converted to their internal representation, and stored as MYHAND[J] for J = 1,2,...,13. MYHAND[J] > 0 will represent the Jth card; MYHAND[J] = 0 will indicate that the Jth card has not been read. MYHAND is later ordered. In addition to setting the array MYHAND, ASKFORHAND is to set the Boolean parameter NOPE to <u>false</u> if the hand is successfully read (including successful error recovery) and is to set NOPE to <u>true</u> if the hand is not read for any reason.

As an illustration of the theorem, the previous sample hand will be converted to the (ordered) MYHAND array below and with NOPE set <u>false</u>.

J	1	2	3	4	5	6	7	8	9	10	11	12	13
MYHAND[J]	52	50	41	39	38	33	32	29	28	13	12	11	2

<u>Other procedures used by ASKFORHAND</u>

ASKFORHAND uses six externally defined procedures; the effect of each is given in English, but the code and proofs of these procedures are omitted.

(i) GETNEXTLINE--Transfers the 80 or fewer characters of the input text including the mandatory final "←" to TEXT[0],...,TEXT[J] where J is the index of the first "←". It is assumed that such a "←" always exists and hence at most 80 characters are read, the last of which is "←".

(ii) GETCHAR(NUM, FLAG)--In general the first non-blank character at or following TEXT[NUM+1] is sought, and FLAG is set <u>true</u> if the character found is "←". More precisely, let NUM_0 be the input value of NUM. The procedure assumes $-1 \leq NUM_0 \leq 79$. Let C_I be the first non-blank character at or following TEXT[I]. The effect of GETCHAR is explained by the following table giving all the possibilities. $NUM \leq 79$ on output.

Input conditions		Results	
NUM_0	$TEXT[NUM_0]$	NUM on output	FLAG(Boolean valued)
≥ 0	←	NUM_0	<u>true</u>
≥ 0	not ←	index of C_{NUM_0+1}	$C_{NUM_0+1} =$ ←
-1	—	index of C_0	$C_0 =$ ←

(iii) FINDDELIMITER(NUM)--Assumes $-1 \leq NUM \leq 78$. A delimiter is either a blank, a comma, a period or a "←". Starting at TEXT[NUM + 1], $NUM \leq 79$ on output is the index of the first delimiter.

(iv) ASORT(A, M)--Sorts the array A[1:M] into descending order.

(v) PRINTJUSTONE--Prints a hand.

(vi) CHECKFORNEGATIVE--Boolean valued, <u>true</u> if the reply to a program-asked question is "No", <u>false</u> if "Yes".

Proof of correctness of ASKFORHAND

The proof that ASKFORHAND accomplishes what is stated in the theorem is given at the level of the Algol code and makes no mention of any hardware and software. The proof consists of a series of some 20 lemmas ordered so that when a lemma is needed it has already been proved.

Each lemma accomplishes a small, convenient step in the proof. Lemmas 1-7 together show that ASKFORHAND is correct under the assumptions (i) that the input is legal--simplified or non-simplified--and (ii) that no error recovery is necessary. Lemma 1 bounds the character scanning. Lemma 2 shows that a single card of a suit is identified, lemma 3 shows that all cards of a suit are identified, and lemma 4 shows that non-simplified input is handled. Lemma 5 covers the initializing, lemma 6 shows that all four suits are read, and lemma 7 handles the final checks and details.

The ordering of these lemmas corresponds to the so-called "inside-out" strategy of programming--starting with the innermost code and working out. Induction is explicitly used as a proof technique in lemma 3 (for the inner loop of the double loop). Although not actually needed in lemma 6 (for the outer loop), induction could have been used there too.

Lemmas 8-20 cover the case where error recovery is needed to read the hand successfully. The formal statements of the error recovery capabilities are included as part of these lemmas (lemmas 8, 10 and 13 and also the definition preceeding lemma 17).

Highlights of the error recovery process follow. Lemma 9 covers an illegal symbol in a suit, lemma 10 more than 14 cards, and lemma 13 fewer than 12 cards. Lemma 14 handles preparation to read a card for deletion if 14 cards were read while lemma 15 does likewise for addition if only 12 cards were read. Lemma 17 shows that a card is read correctly either for addition or deletion. The hand's being added to or deleted from is covered by lemmas 19 and 20, respectively. The ordering of these lemmas corresponds somewhat to the execution order of the code.

In addition to Appendix 1 which contains the code of ASKFORHAND (with sequence numbers so that the proof may refer to a line of code), Appendices 2 and 3 contain useful information in the form of tables to aid in the proof. There is a table listing for each variable all the locations where that variable is changed in value and a table listing for each label all the labels to which control could pass (without passing an intermediate label). The tables give faster access to this information than does the code.

The proof that each table is correct consists of the phrase "Examination of the code." These tables could be properly considered as additional lemmas. They are invoked many times in the proofs of the lemmas often without explicit mention. A statement that a certain variable is unchanged between two points or changed in specified ways is an appeal to Appendix 2. A statement about the flow of control is an appeal to the information contained in Appendix 3.

The symbols "//" will signify "This completes the proof of the lemma." The proof now begins with lemma 1, under the assumption that the input is legal and no error recovery is needed.

The overall plan of ASKFORHAND is that the TEXT array will be scanned starting at TEXT[0]. TEXT[I] will be the "current" character under scan where I is a program variable.

Lemma 1: If the input is legal and error-free, then $-1 \leq I \leq 79$ throughout ASKFORHAND. This relation on I insures that no scanning of the TEXT array is outside the limits of the array.

Proof: By the table of variable value changes (Appendix 2), I may be set to -1 at 756 or 829 or I may be changed by the procedures GETCHAR and FINDDELIMITER since the actual parameter I is altered. From the definitions of these two procedures (i) no change to I decreases its value and (ii) $I \leq 79$ on exit from these procedures.

One small detail remains. Since FINDDELIMITER assumes $I \leq 78$ on input, there might be an error if FINDDELIMITER were called with $I = 79$. It can be shown that $I \neq 79$ when FINDDELIMITER is called, but for brevity this proof is omitted. It is in this part of the proof of the lemma that the assumption on the input is used. Alternatively the entire problem of $I \neq 79$ can be avoided by a simple change in the definition of FINDDELIMITER.//

Lemma 2: From LASTRANK through 778 a card is recognized. Specifically, if TEXT[I] is the first character of a <RANK ID>, and if SUIT has value 3, 2, 1 or 0, then $N > 0$ at 778 is the internal representation of the card.

Proof: L is set to TEXT[I]. Each <RANK ID> is identified, possibly in conjunction with the second character, from 758 through 778 and the corresponding positive setting to N is the correct internal representation of the card in all cases. N will be set exactly once except that a <RANK ID> of TW or TH will be set at 760 and then reset correctly at 776 to a rank value of 3 or at 777 to a rank value of 2. The collating sequence does permit the test $L \geq 2$ and $L \leq 9$ at 761.

Note that the second character (TEXT[I+1]) is always defined, at worst "←". That is, by Appendix 2 I is unchanged from LASTRANK through 778. Since TEXT[I] is, by assumption, the first character of a <RANK ID>, at least the "←" must follow.

The handling of a <RANK ID> of an X needs special justification since its rank value varies. The rank value is correctly determined by the variable SMALL, since by Appendix 2 SMALL is initially 2 at 756 and is incremented by 1 (after setting N) at 763 for each X encountered. If SMALL $\geq$ 11 at 763, then the current X was assigned a rank value greater than 9, an error. Otherwise EXES[KARDS] is set <u>true</u>. In this paper this fact is needed only in the error recovery procedure (see lemma 19) to note if the 13th card was read as an X.//

It will be convenient to refer to the MYHAND array and assert that the first $T - 1$ cards have been read and that the Tth through the 13th cards have not been read. For this purpose let SCANNED(T) denote the two statements $MYHAND[S] > 0$ for $S = 1, \ldots, T-1$ and $MYHAND[S] = 0$ for $S = T, \ldots, 13$. In other words the argument of SCANNED denotes the first unread or unscanned card.

The next lemma deals with the inner loop from REPEATSUIT to ENDOFSUIT. The lemma states in essence that all the <RANK ID>s of a single suit are recognized and are added to the hand.

Lemma 3: Assume that the <SUIT INPUT> is simplified and that both ONETOGO and EXTRA are false. Further assume at REPEATSUIT the following:

(i) initial value of KARDS ≤ 14,

(ii) SUIT = 3, 2, 1 or 0,

(iii) SOFAR has some initial value (unspecified),

(iv) SCANNED(KARDS) holds and

(v) there are P <RANK ID>s of the suit with $0 \leq P \leq 13$.

Then control does reach ENDOFSUIT at which point KARDS has value $(KARDS + P) \leq 14$, SCANNED(KARDS) holds and SOFAR is unchanged.

Proof: Line 756 (at REPEATSUIT) transfers the suit input to the TEXT array, sets I to -1 to start the scan at TEXT[0] and sets SMALL to 2 (see lemma 2). The proof continues by cases on the form of <SUIT INPUT>.

Case 1. Input of the form $\leftarrow$. GETCHAR sets FLAG to <u>true</u> and control goes to ENDOFSUIT. P = 0 and the conclusion is true, that is no change to KARDS, SOFAR and MYHAND.

Case 2. Input of the form <VOID> $\leftarrow$. GETCHAR locates the "V" or "-" so FLAG is <u>false</u>. No test succeeds from 758 to 779 but the test at 780 succeeds sending control to ENDOFSUIT. P = 0 and the conclusion is true.

Case 3. Input of the form <SUIT HOLDING> $\leftarrow$. Use induction on P. Cases 1 and 2 prove the lemma for P = 0. GETCHAR locates the first character of the first <RANK ID> as TEXT[I] and sets FLAG <u>false</u>. By lemma 2, at 778 $N > 0$ is the internal representation of the first <RANK ID>. The tests at 779 and 780 fail but at 781, the first <RANK ID> is added to the hand changing only MYHAND[KARDS] and therefore the statements of SCANNED. By assumption ONETOGO and EXTRA are both <u>false</u> and $N \neq 0$ sends control to X. KARDS $\leq$ 14 and by 798 KARDS $\leq$ 15.

Since KARDS is initially 1 and stepped by 1 at 798 <u>after</u> reading a card, KARDS after 798 represents one more than the number of cards read.

If KARDS = 14 then legal input implies FINDDELIMITER must locate a "$\leftarrow$" which causes GETCHAR to set FLAG <u>true</u> and control goes to ENDOFSUIT with the conclusion true.

KARDS = 15 is impossible. For if KARDS = 15 at 798, then it must have been 14 at REPEATSUIT or at 798 at a previous time. This is so because from REPEATSUIT to ENDOFSUIT KARDS is changed only at 798 (Appendix 2). Hence KARDS was either 14 at the start at REPEATSUIT or else it became 14 at 798. In the latter case, by the argument for KARDS = 14, control would have gone to ENDOFSUIT and not to 798 to make KARDS = 15. In the former case there can be no more <RANK ID>s so GETCHAR must have found a "←" sending control again to ENDOFSUIT and not to 798.

If KARDS $\leq$ 13 FINDDELIMITER locates the blank character between <RANK ID>s and then GETCHAR locates the first character of the next <RANK ID>, if any. The induction assumption is true for strictly smaller P and the conclusion follows. If there is no next <RANK ID>, FINDDELIMITER locates the "←" and the conclusion follows as above when KARDS = 14.

It should be noted that I is changed only by GETCHAR and FINDDELIMITER so that no <RANK ID> is skipped. Similarly the MYHAND array is changed precisely P times and so no <RANK ID> is "lost."

Note also that termination has been proved, that is control does reach ENDOFSUIT. By the induction assumption the lemma, including control's reaching ENDOFSUIT, is true for smaller P.//

Lemma 4: If in lemma 3 the assumption of simplified <SUIT INPUT> is removed, lemma 3 still holds.

Proof: Each case of non-simplified input will be considered, and it will be shown that such characters are skipped. The net effect is as if such characters were absent. Characters are scanned, that is I is changed, only by GETCHAR and by FINDDELIMITER (Appendix 2). GETCHAR at 757 will skip any string of consecutive blanks, and GETCHAR together with the test at 779 will skip any "." or any ",". There can be at most 79 such skipped characters. Only L and N are set but they will each be reset after the first character of the next <RANK ID> is found by GETCHAR. If there is no next, then L and N are no longer relevant. Thus extra non-group mark delimiters at any place in the input cause no change in the conclusion of lemma 3.

Leading zeros will be skipped by GETCHAR together with the test at 779. Any string of extra characters following a <RANK ID> are skipped by FINDDELIMITER at 799.

Note that control is at GETCHAR at 757 before each <RANK ID> including the first, and control is at GETCHAR after the last <RANK ID>. Control is at FINDDELIMITER after each <RANK ID> with TEXT[I] the first character of the <RANK ID>.//

Lemma 5: Starting at 743 and after 745 the following hold:

SCANNED(1), EXES[K] = <u>false</u> for K = 1,2,...,15;

ONETOGO, EXTRA and NOPE are all <u>false</u>.

Proof: Obvious.//

Lemma 6: Starting at GOAGAIN, the cards of each of the four suits are read and control reaches 801 with KARDS = 14, SOFAR = 13 and SCANNED(14).

Proof: Let S, H, D and C here denote the <u>number</u> of spades, hearts, diamonds and clubs, respectively. S + H + D + C = 13 by assumption of legal error-free input. Apply lemma 3 four times to obtain the table below. Lemma 5 and 746 justify the assumptions of lemma 3 the first time. The previous use of lemma 3 justifies the assumptions for the remaining applications.

Time	KARDS	SUIT	SOFAR	MYHAND
prior to requesting				
SPADES	$1 \leq 14$	3	0	SCANNED(1)
HEARTS	$S+1 \leq 14$	2	S	SCANNED(S+1)
DIAMONDS	$S+H+1 \leq 14$	1	S+H	SCANNED(S+H+1)
CLUBS	$S+H+D+1 \leq 14$	0	S+H+D	SCANNED(S+H+D+1)
at 801	S+H+D+C+1 = 14	-1	S+H+D+C = 13	SCANNED(14)

In the code covered by lemma 6, SUIT is changed only at 800. The initial value of SUIT, the decrementing by 1 at 800, the failure of the test at 800 and line 820 together show that lemma 3 is used precisely four times.//

Lemma 7: Control passes from 801 to the exit with the hand successfully read. Thus NOPE is false and SCANNED(14), that is MYHAND[S] > 0 for $S = 1,\ldots,13$.

Proof: Using lemma 6, KARDS is 13 after 801 and control reaches DUP. A duplicate card which might be caused by too many Xs in comparison to the lowest ranked card will be checked in 811-817. For example, 4 X X X will cause a duplicate 4. Assuming no duplicate cards, control reaches 818. This is so because $M > K$ by 812 and hence MYHAND[K] $\neq$ MYHAND[M] each time at 813. K and M are changed only at 811 and 812.

From 818 control reaches FINALLY. The hand is sorted (by ASORT) and printed (by PRINTJUSTONE) in sorted order, and the user is asked if it is correct. Assuming a reply of yes, that is CHECKFORNEGATIVE is false, ASKFORHAND terminates. NOPE is unchanged from 745. The conclusion of lemma 7 holds.//

This completes the proof of the main theorem under the assumption that the input is legal and no error recovery is needed. There is still the case when error recovery is required. It will be shown that correct results always occur in this case, too.

Lemma 8: If control gets to GOOF and if RETRY ≥ 2, control is at the exit and NOPE is set true. At GOOF if RETRY ≤ 1, messages ask for the entire hand again and control is at GOAGAIN. Resetting of variables gives the initial state of computation except for RETRY which is incremented by 1 and except for EXES[15].

Proof: Obvious. RETRY keeps a count of the number of errors before ASKFORHAND stops attempting to read the hand and so RETRY should change. That EXES[15] need not be reset is explained in the proof of lemma 9.//

Lemma 9: Assume ONETOGO and EXTRA are both _false_. Starting at LASTRANK, if an illegal or unidentified symbol (one that is not a delimiter nor a legal <RANK ID>) is encountered as part of a <SUIT HOLDING>, the user is asked to retype the suit and control gets to NEXTCARD ready to read the suit again. Another such error in that suit sends control to GOOF (see lemma 8).

Proof: By definition of illegal symbol, starting at LASTRANK, none of the tests through 780 succeed. Thus N, and by 781 MYHAND[KARDS], are zero (from 758). Control is at 782 where ONETOGO and EXTRA are both _false_, and since MYHAND[KARDS] = 0, control is at 787. If TWOSUIT is _false_ from 745 (from an analysis of the flow of control using Appendix 3 and the settings to TWOSUIT (Appendix 2), it can only be _true_ if a symbol was unidentified in a previous attempt to read this suit), a message is given asking for the suit again and control goes to RESUIT and hence to REPEATSUIT and then to NEXTCARD. The following variables must be and are reset to ignore any cards of the offending suit already recognized: those to make the conditions expressed by SCANNED(SOFAR + 1) true and the proper EXES _false_ are done at 821-822, KARDS is reset at 823, TWOSUIT at 824 and SMALL and I at 756. All others are already correct or irrelevant so at NEXTCARD ASKFORHAND is ready to read the suit again.

In particular, MYHAND[15] and EXES[15] will be initially set at 744 but surprisingly they need not be reinitialized for rereading. Briefly this is because the 15th positions of MYHAND and EXES are used only (i) to read a 15th card (causing the hand to be retyped, see lemma 10) or (ii) used to identify the 14th card to be deleted (see lemmas 15 and 17). EXES[15] may be set but it will then never be used so it need not be reset. MYHAND[15] will always be reset by identifying a card before it is used to delete that card. (This and the definition of FINDDELIMITER were the only instances where it was tempting to change the code to ease the proof; those temptations were barely resisted.)

If TWOSUIT is <u>true</u> when control reaches 787, a message is given identifying the illegal symbol and control goes to GOOF.//

Lemma 10: Assume ONETOGO and EXTRA are <u>false</u>. If a 15th card is read then KARDS = 15 and the hand must be retyped or else complete failure.

Proof: Control is at 781 where, since the card was identified, $N > 0$ and hence control goes from 786 to X. Since KARDS is initially 1 and stepped by 1 at 798 (Appendix 2) <u>after</u> reading a card, reading a 15th card implies KARDS = 15. Hence control at X passes to GOOF and the rest of the conclusion follows by lemma 8.//

The next lemma considers how control reaches 801 where the intent is that all four suits have been completely read. Lemmas 9 and 10 cover two of the cases when the four suits are not completely read and instead retyping is required. At 801 the hand may have as many as 14 cards (see lemma 12) or the hand may contain too few cards.

Lemma 11: Assume ONETOGO and EXTRA are false. Starting at the beginning at 743, control reaches ENDOFSUIT and then 801 after all four suits have been read except for an illegal symbol (lemma 9), 15 cards (lemma 10) or too many Xs (end of proof of lemma 2).

Proof: Using Appendix 3 the conclusion follows by an analysis of the flow of control from 743 to ENDOFSUIT and then to 801. The analysis is aided by an argument similar to the proof of lemma 6 but modified to allow for too few cards or 14 cards. As in lemma 6 $SUIT < 0$ implies all four suits have been read.//

Lemma 12: After KARDS is decremented at 801, KARDS represents the number of cards identified. $KARDS \leq 14$.

Proof: KARDS must be decremented since there was no card corresponding to the last increment of KARDS at 798. By the test at X, by 798 and by lemma 10 $KARDS \leq 14$.//

The next three lemmas assume control starts at 801.

Lemma 13: If 11 or fewer cards are identified (801), the hand must be retyped or else complete failure.

Proof: The test at 801 causes a message (11 or less cards), and control goes to GOOF.//

Lemma 14: If exactly 12 cards are identified (803), the user will be asked to type the last card. The card is transferred to the TEXT array, ONETOGO = true, KARDS = 13 and control passes to INTERPRET.

Proof: Obvious at 803-806.//

Lemma 15: If exactly 14 cards are identified (807), the user is asked to remove a card. The card is transferred to the TEXT array, EXTRA = true, KARDS = 15 and control passes to INTERPRET.

Proof: Obvious at 807-810.//

Lemma 16: If exactly 13 cards are identified, control reaches DUP. A duplicate card causes retyping of the hand or else complete failure.

Proof: By lemma 12, KARDS $\leq$ 14. Hence to reach DUP, KARDS must be 13. The case of no duplicate card has already been covered in lemma 7. If there is a duplicate card, MYHAND[K] = MYHAND[M] for some K and M and control goes to GOOF. 811-812 insures that a duplicate card will be detected. (A more rigorous argument by induction is easy to give.)//

It remains only to show that the 13th or 14th card is identified and respectively added to or deleted from the hand. All of these operations start at INTERPRET where lemmas 14 and 15 left matters.

It is first necessary to define the input for specifying a card. The <RANK ID>s will be as before. But means for specifying the suit must be defined since the suit can no longer be determined from the numbered line. An additional complexity is that "S" is the first letter of "Spades" and also of "Six" and "Seven". The rule is that "S" will be taken as "Spades" if the next character is "P", "b" or "$\leftarrow$".

A Backus Naur Form definition for specifying cards with suits is:

<RANK ID> ::= as before

<SUIT ID> ::= S|SP|H|D|C

<CARD> ::= <SUIT ID>b<RANK ID>$\leftarrow$|<RANK ID>b<SUIT ID>$\leftarrow$|

<RANK ID>bOFb<SUIT ID>$\leftarrow$

The rank values and suit values are as before. The above is simplified input; a form of non-simplified input also exists for specifying cards with suits, but none of its properties will be proved. There is no error correcting in specifying a card with suit.

Lemma 17: Starting at INTERPRET, a card will be identified from <CARD> input. If ONETOGO is true (lemma 14), this card will be MYHAND[13]. If EXTRA is true (lemma 15), this card will be MYHAND[15]. Control reaches 844.

Proof: Note that for control to reach INTERPRET, either ONETOGO or EXTRA is true (see Appendix 3). Conversely ONETOGO or EXTRA is true only if control passes through INTERPRET (see Appendix 2). The proof of this lemma is by cases on the form of <CARD>. In all cases GETCHAR locates the first character of the input.

Case 1. <CARD> is of the form <SUIT ID>b<RANK ID>. The suit is identified at 830-834 and hence SUIT $\neq$ 4. FIRST = true from 829, control goes to ALMOST, FINDDELIMITER finds the "b", GETCHAR finds the first character of <RANK ID>, FLAG is set false and control goes to LASTRANK. An "X" will be taken as a 2 by 829. The <RANK ID> is identified implying $N > 0$. Since (precisely) one of ONETOGO or EXTRA is true, control reaches 783 with the card identified. Since SUIT $\neq 4$, $N \leq 52$ and control goes to CALC and then to 844.

Case 2. <CARD> is of the form <RANK ID>b<SUIT ID>. The suit is not identified at 830-834 so SUIT = 4 and control goes to LASTRANK. Again "X" will be taken as a 2, and the card is identified, but since SUIT = 4, $53 \leq N \leq 65$. Control reaches 783, then ALMOST where the "b" is found and where GETCHAR finds the first character of <SUIT ID>. Control goes to GETSUIT where the suit is obtained (SUIT = 3,2,1 or 0), and then to CALC and then to 844.

Case 3. <CARD> is of the form with "OF" in it. This case is the same as case 2 except for an additional return to ALMOST from 837 to read and to skip the "OF".

That the card is MYHAND[13] or MYHAND[15] follows from the respective setting to KARDS in lemmas 14 and 15.//

Lemma 18: $-1 \leq I \leq 79$ throughout ASKFORHAND.

Proof: The proof of lemma 1 holds except for showing that $I \neq 79$ when FINDDELIMITER is called. This is omitted here too.//

It is now necessary to show that the hand is suitably altered in each case after the card is identified in lemma 17.

Lemma 19: Except in the cases delimited in the proof, if ONETOGO is _true_, control gets from 844 to DUP after modifying the 13th card, if necessary, before accepting it.

The lemma is false without the exception since sometimes an "X" is not handled correctly. But let us proceed. EXTRA is _false_. If the 13th card was read in case 1 of lemma 17, MYHAND[13] $\leq$ 52 and control reaches 847. If, however, the 13th card was read in cases 2 or 3, $53 \leq N \leq 65$ but now SUIT is the correct value of 3, 2, 1 or 0. Hence 846 gives MYHAND[13] the correct internal value. If the 13th card was not an "X", control goes to DUP and the lemma is proved.

At 847 if the 13th card was an "X" it is necessary, to avoid spurious duplication, to adjust the internal value to the smallest rank of its suit that is absent (hopefully 2 through 9). Lines 848-851 do this correctly if not all of the 2 through 9 of the suit are present in MYHAND[J] for $J = 1,\dots,12$. For suppose 2 through $R - 1$ are present but not R ($R \leq 9$). The test at 849 will succeed (changing MYHAND[13] by 1) for some J, $1 \leq J \leq 12$, each of the first $R - 2$ times that the J loop is started. However, on the $(R - 1)$st start the test will fail for all J and hence control reaches 852 and then DUP.

Thus MYHAND[13], if read as an "X", is reset to the smallest missing rank R of the suit. If $R > 9$, however, that will be illegally the 10, J, Q, K or even the Ace of the next higher suit. In the case of spades the illegal value of 53 could be generated.//

Lemma 20: If EXTRA is true and if deleting the 14th card is possible, then control goes from 844 to DUP. If deleting is not possible, control goes to GOOF.

Proof: As in the proof of lemma 19, but with EXTRA = true, control reaches ELIM. If MYHAND[15] ≠ MYHAND[J] for J = 1,...,14, control goes to GOOF. This could happen if the 15th card is not in the hand, or seems not to be in the hand because the 15th card is an "X" with the lowest rank of the suit already present being 3 through 9.

If, however, MYHAND[15] = MYHAND[J] for some J = 1,...,14, then MYHAND[K] becomes MYHAND[K + 1] for K = J,...,14. Since K starts at J, the original Jth card is deleted. In other words, MYHAND[K + 1] is shifted to MYHAND[K], restoring the hand to 13 cards. Then control goes to DUP.//

Thus either by lemma 19 or by lemma 20 (and ignoring the possibility of going to GOOF), MYHAND has 13 cards and control is at DUP. The case of no duplicate cards is covered in lemma 7 while the case of a duplicate card is covered by lemma 16.

This completes the proof of the main theorem.

Other properties of the error correcting capabilities of ASKFORHAND are presented without proof. Control is sent to GOOF if (i) a symbol of an (expected) <SUIT ID> or <RANK ID> cannot be identified as either, for example the input is of the form <SUIT ID>b<SUIT ID> or <RANK ID>b<RANK ID>, (ii) an input ends before sufficient information has been read (but see below), or (iii) the user replies that the hand is incorrect when he is asked. Input of the form

<RANK ID>b...b<RANK ID>b<RANK ID>b<SUIT ID>

causes no error, giving the underlined result. Other illegal combinations may similarly be examined. Finally if the reply to a request for a <CARD> is simply "←", the request is repeated. A further reply of "←" repeats the cycle.

Concluding remarks

The errors noted in lemmas 19 and 20 were all uncovered only in the attempt to prove the correctness of the code. Although easily remedied, they are deemed so pathological that they remain. None is fatal because control goes to GOOF, or the user, when later asked, can say the hand is incorrect. Several other minor errors, again uncovered only by the proof process, were later fixed.

Incidentally, this should not be taken as criticism of the programmer; if anything it serves as praise. The program meets its design criteria of performing efficiently a utility type task for a much larger program in a "quick and dirty manner." It certainly was not designed to be subjected to the detailed study and examination of the proof process.

Finally, the purpose of presenting the proof is not limited to demonstrating that ASKFORHAND is correct, although that has been done. As noted in the introduction, (i) the proof uses new techniques, (ii) the proof, a complicated one, is presented as a series of lemmas and (iii) the proof is an example of how one might proceed to prove correct a realistic program.

Acknowledgments

This research is supported by NSF Grant GP-7069 and the Mathematics Research Center, United States Army under Contract Number DA-31-124-ARO-D-462.

I wish to thank Anthony I. Wasserman for providing ASKFORHAND in the first place, for patiently explaining its operation to me when questions arose and for not forbidding my displaying the code publicly. Both he and Donald I. Good have provided useful suggestions in presenting the proof. I, of course, am solely responsible for the accuracy and form of the proof.

References

1. Evans, A. Jr., Syntax analysis by a production language. Ph.D. thesis, Carnegie-Mellon University, 1965.

2. Good, D. I. and London, R. L., Interval arithmetic for the Burroughs B5500: Four Algol procedures and proofs of their correctness. Computer Sciences Technical Report No. 26, University of Wisconsin, 1968.

3. London, R. L., Computer programs can be proved correct. In *Proceedings of the Fourth Systems Symposium: Formal Systems and Non-Numerical Problem Solving by Computers*, Case Western Reserve University, to appear.

4. London, R. L. and Wasserman A. I., The anatomy of an Algol procedure, Computer Sciences Technical Report No. 5, University of Wisconsin, 1967.

5. McCarthy, J. and Painter, J. A., Correctness of a compiler for arithmetic expressions. In *Proceedings of a Symposium in Applied Mathematics*, Vol. 19--Mathematical Aspects of Computer Science, Schwarcz, J. T. (ed.), American Mathematical Society, Providence, R. I., 1967, pp. 33-41.

6. Wasserman, A. I., Bridge bidding by computer. Unpublished research, University of Wisconsin, undated.

Appendix 1. Algol Listing of ASKFORHAND

```
BEGIN INTEGER I, J, K, L, M, N; ALPHA ARRAY TEXT[0:79];                   7000
BOOLEAN ARRAY EXES[1:15];                                                 7010
PROCEDURE GETNEXTLINE; BEGIN END;                                         7020
PROCEDURE GETCHAR(NUM, FLAG); INTEGER NUM; BOOLEAN FLAG; BEGIN END;       7030
PROCEDURE FINDDELIMITER(NUM); INTEGER NUM; BEGIN END;                     7040
PROCEDURE ASORT (A,M); INTEGER M; INTEGER ARRAY A[1]; BEGIN END;          7050
PROCEDURE PRINTJUSTONE; BEGIN END;                                        7060
BOOLEAN PROCEDURE CHECKFORNEGATIVE; BEGIN END;                            7070
                                                                          7080
PROCEDURE ASKFORHAND (NOPE); BOOLEAN NOPE;                                7160
BEGIN INTEGER ARRAY MYHAND[1:15];                                         7170
   INTEGER KARDS, SUIT, SMALL, SOFAR, RETRY;                              7180
   LABEL GOAGAIN,HEREGOES,REPEATSUIT,NEXTCARD,LASTRANK,ENDOFSUIT,X,       7190
   RESUIT,RESET,INTERPRET,GETSUIT,ALMOST,CALC,GOOF,FINALLY,FAILURE;       7200
   LABEL DUP,FIXUP,CHANGE,ELIM;                                           7210
   BOOLEAN FLAG,TWOSUIT,ONETOGO,FIRST,EXTRA;                              7220
   FOR J←1 STEP 1 UNTIL 15 DO                                             7430
   BEGIN MYHAND[J]←0; EXES[J]←FALSE; END;                                 7440
   TWOSUIT←ONETOGO←NOPE←EXTRA←FALSE; RETRY←0;                             7450
GOAGAIN: KARDS←1; SUIT←3; SOFAR←0;                                        7460
HEREGOES:COMMENT REQUEST CORRECT SUIT, IE IF SUIT = 3 THEN REQUEST        7470
   SPADES ELSE IF SUIT = 2 THEN REQUEST HEARTS ELSE IF SUIT = 1           7480
   REQUEST DIAMONDS ELSE REQUEST CLUBS;                                   7490
```

```
REPEATSUIT: GETNEXTLINE; SMALL←2; I←-1;
NEXTCARD: GETCHAR(I,FLAG); IF FLAG THEN GO TO ENDOFSUIT;
LASTRANK: L←TEXT[I]; N←0; IF L="K" THEN N←13×(SUIT+1);
   IF L="Q" THEN N←13×SUIT+12; IF L="J" THEN N←13×SUIT+11;
   IF L="T" OR L="1" THEN N←13×SUIT+10; IF L="N" THEN N←13×SUIT+9;
   IF L≥2 AND L≤9 THEN N←13×SUIT+L; IF L="A" THEN N←13×SUIT+1;
   IF L="X" THEN
   BEGIN N←13×SUIT+SMALL; SMALL←SMALL+1; IF SMALL≥11 THEN GO TO GOOF;
      EXES[KARDS]←TRUE;
   END;
   IF L="E" THEN N←13×SUIT+8;
   IF L="S" THEN
   BEGIN IF TEXT[I+1]="E" THEN N←13×SUIT+7;
      IF TEXT[I+1]="I" THEN N←13×SUIT+6;
   END;
   IF L="F" THEN
   BEGIN IF TEXT[I+1]="I" THEN N←13×SUIT+5;
      IF TEXT[I+1]="O" THEN N←13×SUIT+4;
   END;
   IF L="T" THEN
   BEGIN IF TEXT[I+1]="H" THEN N←13×SUIT+3;
      IF TEXT[I+1]="W" THEN N←13×SUIT+2;
   END;
```

```
    IF L="O" OR L=",," OR L="." THEN GO TO NEXTCARD;                              7790
    IF L="V" OR L="-" THEN GO TO ENDOFSUIT;                                       7800
    MYHAND[KARDS]←N;                                                              7810
    IF ONETOGO OR EXTRA THEN                                                      7820
    BEGIN FIRST←FALSE; IF MYHAND[KARDS]>52 THEN GO TO ALMOST                      7830
    ELSE IF MYHAND[KARDS]>0 THEN GO TO CALC ELSE GO TO GOOF;                      7840
    END;                                                                          7850
    IF MYHAND [KARDS] ≠0 THEN GO TO X;                                            7860
    IF TWOSUIT THEN                                                               7870
    BEGIN COMMENT WRITE UNRECOGNIZED SYMBOL MESSAGE; GO TO GOOF; END;             7880
    COMMENT WRITE UNRECOGNIZED SYMBOL MESSAGE AND REQUEST                         7890
    THE SUIT AGAIN; GO TO RESUIT;                                                 7900
   X:IF KARDS>14 THEN                                                             7960
    BEGIN COMMENT WRITE TOO MANY CARDS MESSAGE; GO TO GOOF; END;                  7970
    KARDS← KARDS+1;                                                               7980
    FINDDELIMITER(T); GO TO NEXTCARD;                                             7990
  ENDOFSUIT:SUIT←SUIT-1; SOFAR←KARDS-1; IF SUIT< 0 THEN                           8000
    BEGIN KARDS← KARDS-1; IF KARDS≤11 THEN                                        8010
       BEGIN COMMENT WRITE 11 OR LESS CARDS; GO TO GOOF; END;                     8020
       IF KARDS=12 THEN                                                           8030
       BEGIN COMMENT WRITE ONLY 12 CARDS AND ASK FOR LAST CARD;                   8035
          ONETOGO←TRUE;                                                           8040
          GETNEXTLINE; KARDS←13; GO TO INTERPRET;                                 8050
```

```
     END;
     IF KARDS=14  THEN
     BEGIN COMMENT WRITE 14 CARDS AND ASK WHICH IS TO BE DELETED;
        EXTRA←TRUE;KARDS←15;
        GETNEXTLINE; GO TO INTERPRET;
     END;
     DUP: FOR K← 1 STEP 1 UNTIL 13 DO
     FOR M← K+1 STEP 1 UNTIL 13 DO
     IF MYHAND[K]=MYHAND[M] THEN
     BEGIN COMMENT ANNOUNCE DUPLICATE CARD; GO TO GOOF;
     END;
     GO TO FINALLY;
  END;
  GO TO HEREGOES;
RESUIT: FOR J← SOFAR+1 STEP 1 UNTIL 14 DO
  BEGIN MYHAND[J]←0; EXES[J]←FALSE; END;
  KARDS←SOFAR+1;
  TWOSUIT←TRUE; GO TO REPEATSUIT;
RESET: IF RETRY ≥ 2 THEN BEGIN NOPE←TRUE; GO TO FAILURE; END;
  FOR J←1 STEP 1 UNTIL 15 DO MYHAND[J]←0;RETRY←RETRY+1;
  ONETOGO←EXTRA←TWOSUIT←FALSE;
  FOR J← 1 STEP 1 UNTIL 14 DO EXES[J]←FALSE; GO TO GOAGAIN;
INTERPRET:I←-1;GETCHAR(I,FLAG);IF FLAG THEN GO TO ENDOFSUIT;
```

```
   SUIT←4;FIRST←TRUE;SMALL←2;
GETSUIT: IF TEXT[I]="S" AND (TEXT[I+1]="P" OR TEXT[I+1]=" "
   OR TEXT[I+1]="←")THEN SUIT←3;
   IF TEXT [I] = "H" THEN SUIT←2;
   IF TEXT[I] = "D" THEN SUIT←1;
   IF TEXT[I] = "C" THEN SUIT←0;
   IF SUIT=4 THEN GO TO LASTRANK; IF NOT FIRST THEN GO TO CALC;
ALMOST: FINDDELIMITER(I);GETCHAR(I,FLAG);IF FLAG THEN GO TO GOOF;
   FIRST←FALSE; IF TEXT[I]="O" THEN GO TO ALMOST;
   GO TO IF SUIT=4 THEN GETSUIT ELSE LASTRANK;
CALC:IF SUIT = 4 THEN
GOOF: BEGIN IF RETRY≤1 THEN
      BEGIN COMMENT WRITE START AGAIN MESSAGE;END;
      GO TO RESET;
   END;
      IF EXTRA THEN GO TO IF MYHAND[KARDS]>52 THEN FIXUP ELSE ELIM;
      IF MYHAND[KARDS]>52 THEN
   MYHAND[13]←MYHAND[13]-13×(4-SUIT);
   IF EXES[13] THEN
 CHANGE: BEGIN FOR J←1 STEP 1 UNTIL 12 DO
      IF MYHAND[J]=MYHAND[13] THEN
      BEGIN MYHAND[13]←MYHAND[13]+1; GO TO CHANGE; END;
   END;
   GO TO DUP;                                                  8520
FIXUP: MYHAND[15]←MYHAND[15] - 13×(4-SUIT);                    8530
 ELIM:FOR J←1 STEP 1 UNTIL 14 DO IF MYHAND[J]=MYHAND[15] THEN  8540
   BEGIN FOR K← J STEP 1 UNTIL 14 DO                           8550
      BEGIN MYHAND[K]←MYHAND[K+1]; EXES[K]←EXES[K+1]; END;     8560
      GO TO DUP;                                               8570
   END;    GO TO GOOF;                                         8580
FINALLY: ASORT(MYHAND,13);                                     8590
   COMMENT WRITE HAND WILL BE TYPED MESSAGE;                   8600
   PRINTJUSTONE;                                               8610
   COMMENT ASK IF HAND IS CORRECT;                             8620
   IF CHECKFORNEGATIVE THEN GO TO GOOF;                        8630
FAILURE:END;                                                   8640
```

Appendix 2. Variable Value Changes

Variable	Changed at line numbers
EXES	744, 764, 822, 828, 856
EXTRA	745, 808, 827
FIRST	783, 829, 837
FLAG	GETCHAR(757, 829, 836)
I	756, GETCHAR(757, 829, 836), FINDDELIMITER(799, 836), 829
J	743, 821, 826, 828, 848, 854,
K	811, 855
KARDS	746, 798, 801, 805, 808, 823
L	758
M	FINDDELIMITER(799, 836), 812
MYHAND[ ]	744, 781, 822, 826, 846, 850, 853, 856
N	758^2, 759^2, 760^2, 761^2, 763, 766, 768, 769, 772, 773, 776, 777
NOPE	745, 825
ONETOGO	745, 804, 827
RETRY	745, 826
SMALL	756, 763, 829
SOFAR	746, 800
SUIT	746, 800, 829, 831, 832, 833, 834
TEXT[ ]	GETNEXTLINE(756, 805, 809), CHECKFORNEGATIVE(863)
TWOSUIT	745, 824, 827

^{2} means twice in the same line.

Appendix 3. Flow Between Labels (without passing an intermediate label)

From label[1]	To label(s)[1]
ALMOST(836)	LASTRANK(758), ALMOST(836), GOOF(840), GETSUIT(830)
CALC(839)	CHANGE(848), GOOF(840), FIXUP(853), ELIM(854), DUP(811)
CHANGE(848)	DUP(811), CHANGE(848)
DUP(811)	GOOF(840), FINALLY(859)
ELIM(854)	GOOF(840), DUP(811)
ENDOFSUIT(800)	INTERPRET(829)[2], HEREGOES(747), GOOF(840), DUP(811)
FAILURE(864)	"EXIT"
FINALLY(859)	FAILURE(864)
FIXUP(853)	ELIM(854)
GETSUIT(830)	LASTRANK(758), CALC(839)
GOAGAIN(746)	HEREGOES(747)
GOOF(840)	RESET(825)
HEREGOES(747)	REPEATSUIT(756)
INTERPRET(829)	ENDOFSUIT(800), GETSUIT(830)
LASTRANK(758)	NEXTCARD(757), RESUIT(821), X(796), CALC(839), ALMOST(836), GOOF(840)[3], ENDOFSUIT(800)
NEXTCARD(757)	LASTRANK(758), ENDOFSUIT(800)
REPEATSUIT(756)	NEXTCARD(757)
RESET(825)	GOAGAIN(746), FAILURE(864)
RESUIT(821)	REPEATSUIT(756)
X(796)	GOOF(840), NEXTCARD(757)

[1] numbers in parentheses are sequence numbers where the label is located.

[2] two separate paths exist.

[3] three separate paths exist.