RESEARCH
SERIES

NBSIR 88-3741

A SCHEME FOR TRANSLATING CONTROL FLOW IN THE C PROGRAMMING LANGUAGE TO GRAFCET WITH EXAMPLES

March 18, 1988

By:
Bruce H. Thomas

U.S. DEPARTMENT OF COMMERCE ■ National Bureau of Standards ■ Gaithersburg, Maryland

A Scheme for Translating Control Flow in the C Programming Language to Grafcet with Examples

Bruce Hunter Thomas

Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

A Scheme for Translating Control Flow in the C Programming Language to Grafcet with Examples

Bruce Hunter Thomas

National Bureau of Standards

Gaithersburg, MD

Keywords: Grafcet, C programming language, control flow, programming languages

Abstract:

The purpose of this paper is to show a translation scheme from control flow in the C programming language to the Grafcet language. Grafcet is a graphical language for expressing control flow. Grafcet is used to design parallel systems such as in a manufacturing environment. The control constructs covered in this paper are: conditional statement, while, do, for, switch, break, continue, goto, label, and null. The Grafcet used in this paper is the language, as augmented by Savoir. The C programming language is the one described by Kernighan and Ritchie. This translation is to be used as a reference for programmers to translate existing C source code into Grafcet.

Introduction:

Grafcet is a powerful graphical language for expressing control flow. Savoir has implemented a version of Grafcet which uses C source code[1]. Grafcet is an excellent tool for designing, documenting, and demonstrating the control flow of a system. A system which is already written in C can be translated into a set of Grafcet programs. This paper shows a translation scheme from the control flow in a C program to a set of Grafcet programs. The C source code constructs are those defined by Kernighan and Ritchie [2].

A basic introduction to the portion of the Grafcet language needed for the translations is given in the paper. A number of the extensions to Grafcet by Savoir, used for translating, are described. The translation of these C source code control constructs are described: conditional statement, **while**, **do**, **for**, **switch**, **break**, **continue**, **goto**, label, and null.

The problem of representing data flow or a definitive method for translating functions into a Grafcet form from a C program is not directly addressed, but a discussion of the problems of representing C functions in Grafcet is given. A possible scheme for functions translated to macro steps is outlined, but a definitive solution is not given.

Background:

For the purposes of this paper we will define our problem as translating a set of C source code control constructs into a Grafcet system, where a Grafcet system is a hierarchy of Grafcet programs. A Grafcet program is either a main program or a macro program. There is

only one main program for each Grafcet system and it is at the top of the hierarchy. A macro program is an expanded Grafcet program from a macro step. Below is a definition of the Grafcet primitives, the goal of using these primitives, and the features Savoir augmented to the Grafcet language.

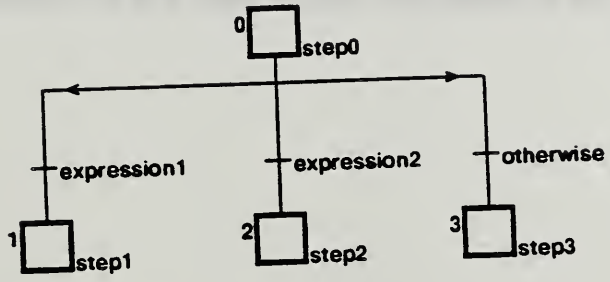
The Grafcet language has three major primitives: regular steps, macro steps, and transitions. Regular steps allow the user to represent an arbitrary control action in the form of embedded C source code. The Grafcet places a dot in the regular step's square to indicate that C source code is embedded in it, see the example below. Macro steps allow the user to name embedded Grafcet macro programs as control actions. Transitions act as gates on the flow of control through the Grafcet program. Associated with each transition is a C expression which determines if control can pass through this point. If the expression evaluates to **False**, the gate stops the control flow. If the expression is **True**, the gate allows the control flow to continue. These primitives can be connected to form a control flow path by attaching a link between two primitives. A link can only be attached between a step primitive and a transition primitive; that is to say a transition primitive cannot be linked to another transition primitive. These links are represented by a line from one primitive to another; sometimes an arrowhead is used to indicate the direction of the control flow. Examples of what these primitives look like are shown below. In these examples **0** and **M1** are labels supplied by the Grafcet programming environment, and **label** is a user supplied name for a step.

(regular step) (macro step) (transition)

It has been shown that all the other C control constructs can be formed from just conditional and **goto** statements [3],[4]. Therefore by translating the C conditional and **goto** statements first, the rest of the C control constructs will follow.

There are three features with which Savoir has augmented the Grafcet language that are used in translating C source code into Grafcet. The Grafcet transitions use C expressions for the condition that determines whether or not control can pass through the transition. In an asynchronous branch a special transition condition of **otherwise** was added. An asynchronous branch is a single step followed by two or more transitions. Control can flow through one or more of the transitions whose conditions are true. **Otherwise** is defined as a condition that is true if the conditions of the transitions to its left in a set of asynchronous branches are false. An example of an asynchronous branch with an **otherwise** transition is shown below.

Example of an asynchronous branch with an **otherwise** transition:



(asynchronous branch)

The C programming language was chosen for this translation because Savoir has augmented Grafcet to support C source code in their "regular step." This translation can be extended to other languages with similar constructs such as Pascal, Fortran, or Algol 60.

Definitions:

In the examples of Grafcet programs, T is defined as **True** or:

```
T != 0
```

Description of Translation:

The translation of these C source code control constructs is described in this section: null, label, **goto**, conditional statement, **while**, **do**, **for**, **switch**, **break**, and **continue**. To help clarify the descriptions of translations a mixture of flowcharts, C statements, or examples are used. The translated Grafcet statements are fragments which may be linked together to form a complete Grafcet program.

Null statement: ;

A null statement translates to a regular step with no C source code associated with it. (For the examples shown here, the null statement regular steps will be labeled null.)

(null in Grafcet)

Labeled statement: *identifier* :

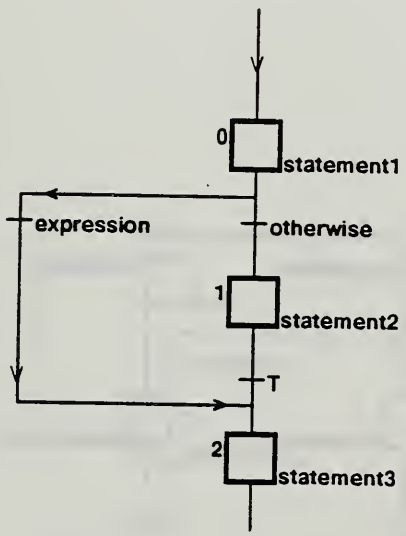
The label statement in C is used for a destination location for the **goto** statement. There is no explicit translation into Grafcet, but the **goto** statement is translated.

Goto statement: **goto** *identifier* ;

As described above, control follows the links in a Grafcet program. Each link acts similar to a **goto** from point to point in a Grafcet program. A link must be contained in one Grafcet macro program. The **goto** statement is not the most popular construct in the C language; therefore it is anticipated the C **goto** construct will not be used often [5].

Example **goto** statement :

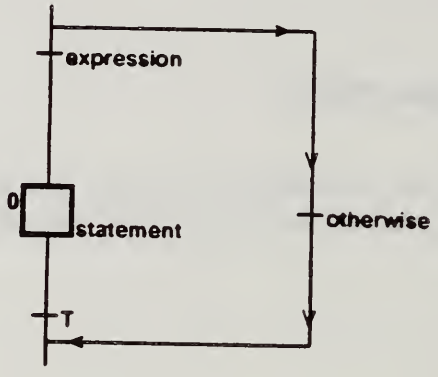
```
        statement1;
        if expression goto NEXT;
        statement2;
NEXT:   statement3;
```



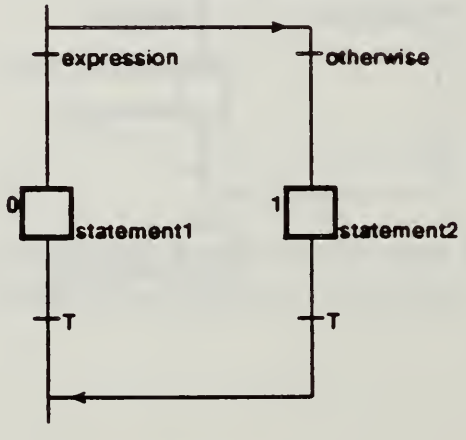
(**goto** in Grafcet)

Conditional statement: **if** (*expression*) *statement*
if (*expression*) *statement1* **else** *statement2*

The conditional statement is made with two asynchronous branches. One branch is a transition with the condition *expression*, and the other transition has the condition **otherwise**.



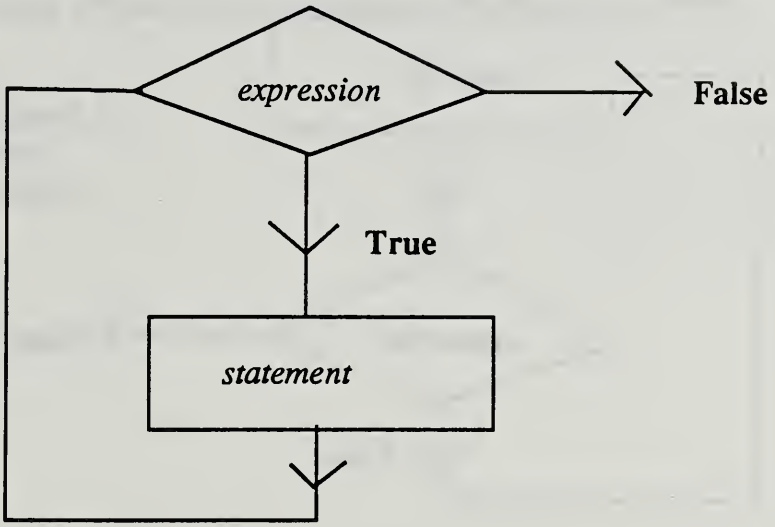
(**if then** in Grafcet)



(**if then else** in Grafcet)

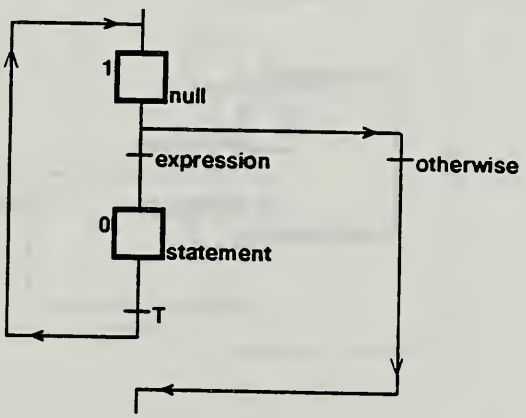
While statement: **while** (*expression*) *statement*

The logic for the **while** statement is as follows [6]:



(**while** flowchart)

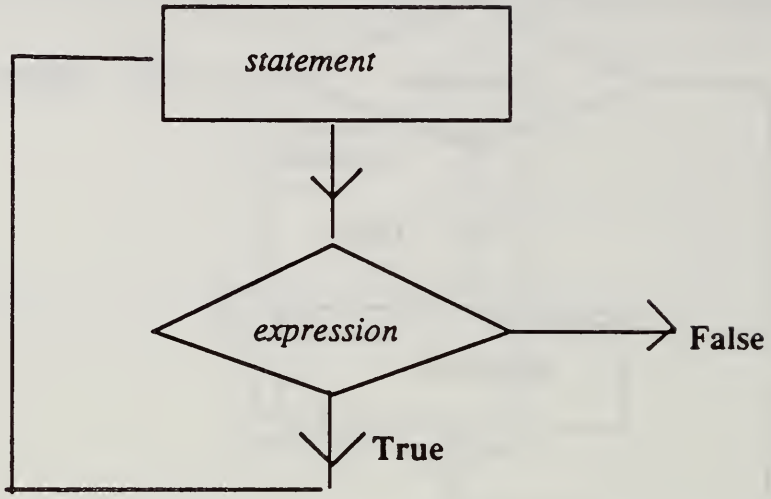
The Grafcet equivalent is:



(**while** in Grafcet)

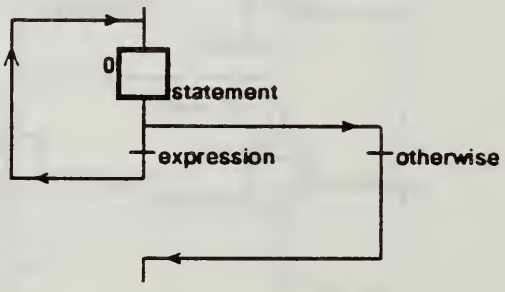
Do statement : **do** *statement* **while** (*expression*) ;

The logic for the **do** statement is as follows:



(**do** flowchart)

The Grafcet equivalent is:

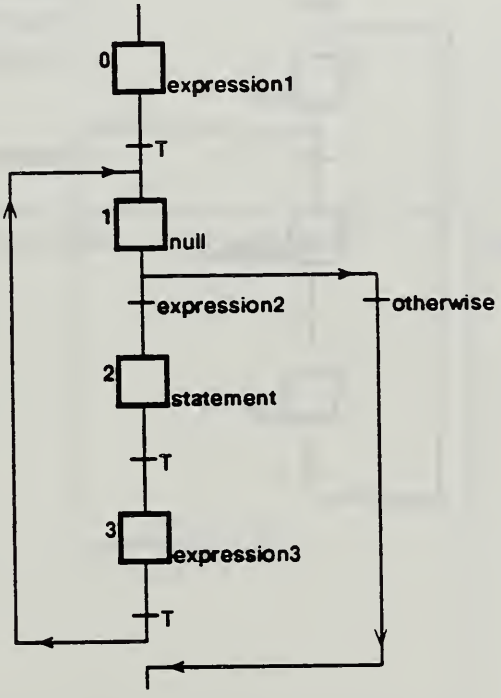


(**do** in Grafcet)

For statement : **for** (*expression1* ; *expression2* ; *expression3*) *statement*

The **for** statement can be expressed as a **while** statement; where *expression1* and *expression3* are constructed by placing C source code of the expressions in regular steps. The C source code using a **while** loop to represent a **for** statement is shown:

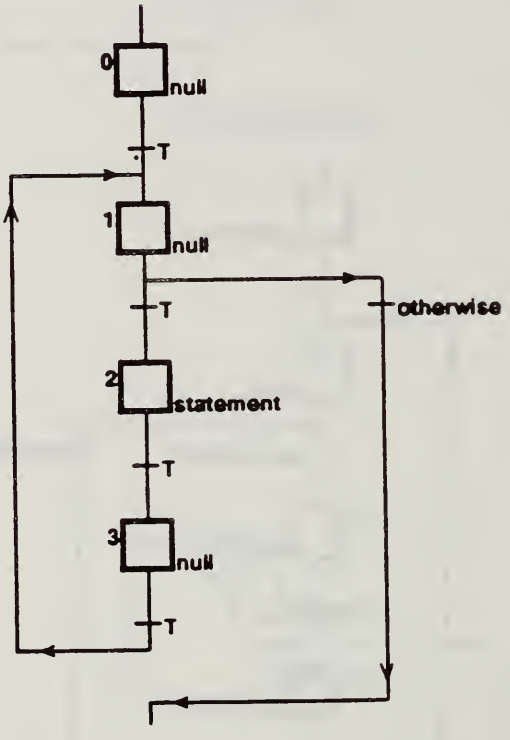
```
expression1;
while ( expression2 ) {
        statement;
        expression3;
        }
```



(**for** in Grafcet)

Each of the three expressions in the **for** statement are optional. If the expressions *expression1* or *expression3* are not used in a **for** statement, the unused expression is replaced by a null statement in the Grafcet program. If the expression *expression2* is not used in a **for** statement, the expression in transition replaced by T in the Grafcet program. The Grafcet **for** loop shown below demonstrates all of the replacements of the three optional expressions, even though the Grafcet code functional makes little sense.

for (;;) *statement*



(**for** (;;) in Grafcet)

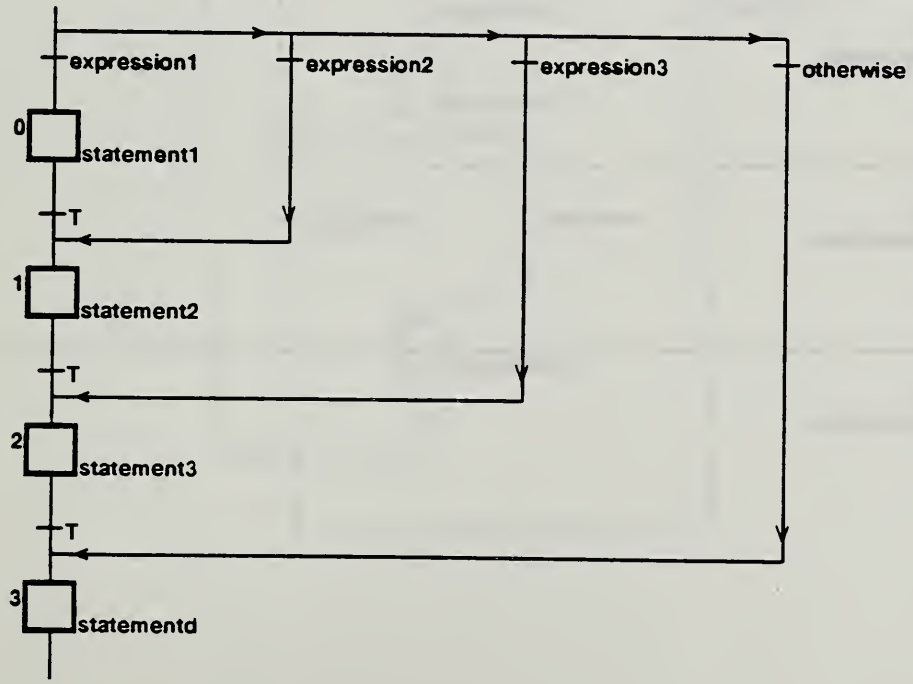
Switch statement : **switch** (*expression*) *statement*
case *constant-expression* **:**
default :

The **switch** statement can easily be expressed with only conditional and **goto** statements [3]. The **case** label is replaced by an equivalence comparison between the constant-expression and the expression. The **default** case is handled by the **otherwise** condition in the Grafcet. The translation of the **switch** statement in Grafcet always produces a **default** statement to allow the control flow to continue. If there is no **default** in the C source code, a null statement must be inserted after the **otherwise** condition. The **switch** statement is commonly used with **break** statements, the **break** statement is defined in the next section. The first example is a **switch** statement without the **break** statements, and the second example is the **switch** statement with **break** statements.

Examples :

The **switch** statement without the **break** statements:

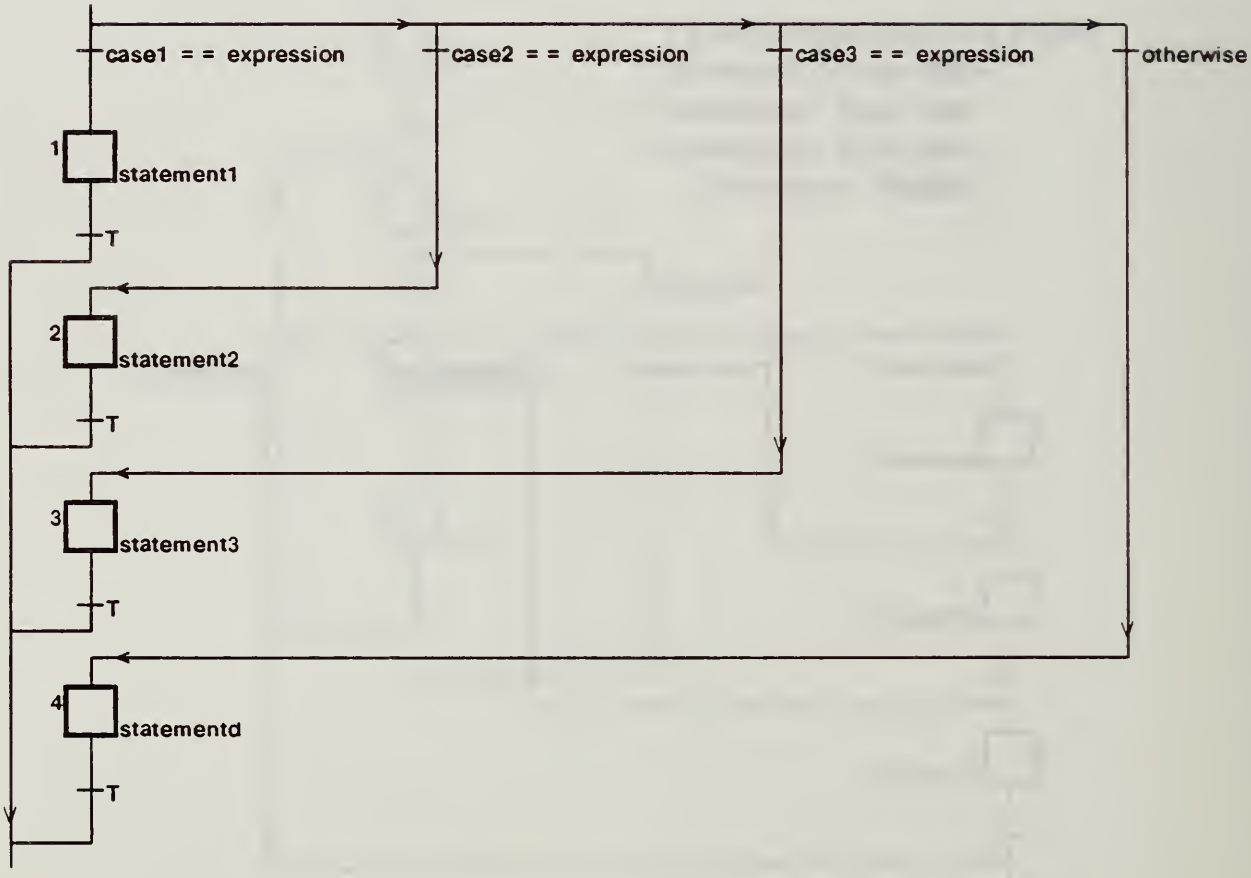
switch (*constant- expression*) {
case *case1* : *statement1*;
case *case2* : *statement2*;
case *case3* : *statement3*;
default : *statementd*;
}



(**switch** in Grafcet)

The **switch** statement with the **break** statements:

```
switch ( expression ) {
        case case1 : statement1;
                     break;
        case case2 : statement2;
                     break;
        case case3 : statement3;
                     break;
        default : statementd;
        }
```



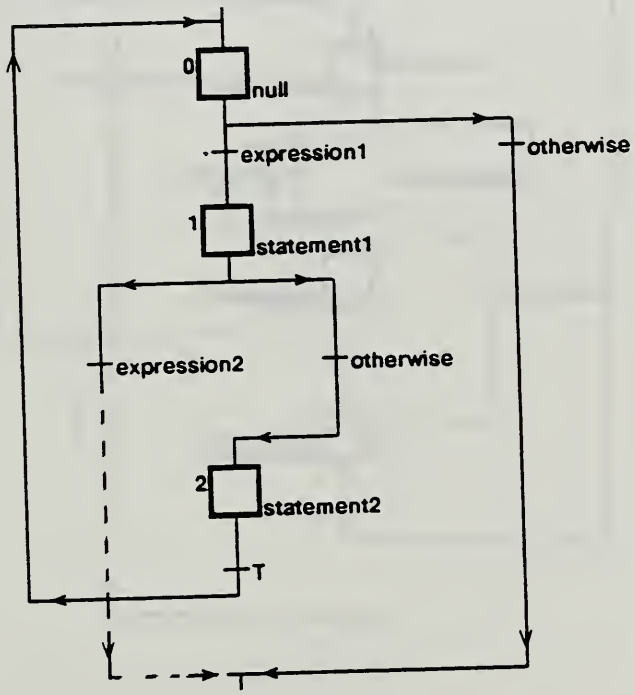
(case in Grafcet)

The last two constructs, **break** and **continue**, can be expressed in the form of **goto** and label statements; therefore links in the Grafcet programs can replace them. Examples are given to help show how to perform these conversions.

Break statement : **break;**

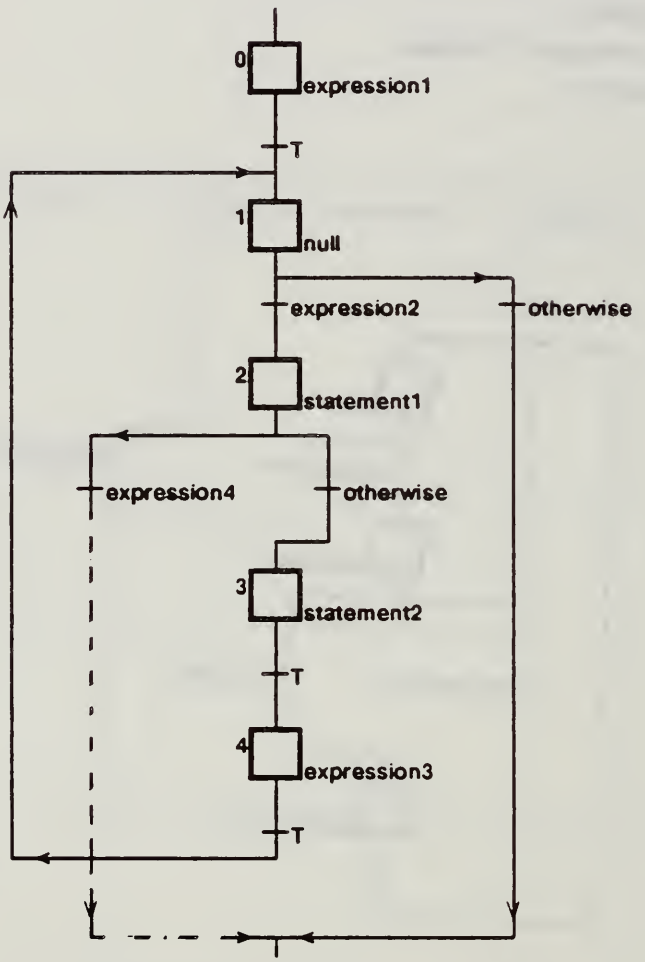
The **break** statement is equivalent to a **goto** statement which passes control to next statement after the smallest enclosing **while, do, for,** or **switch** statement. The **break** statement is shown earlier in the discussion of the **switch** statement. Examples of the **break** statement in the **while, do,** and **for** statements are shown below. The links in the examples which are added for the **break** statements are shown as dashed lines.

Examples : **while** (*expression1*) {
statement1;
if (*expression2*) **break;**
statement2;
}



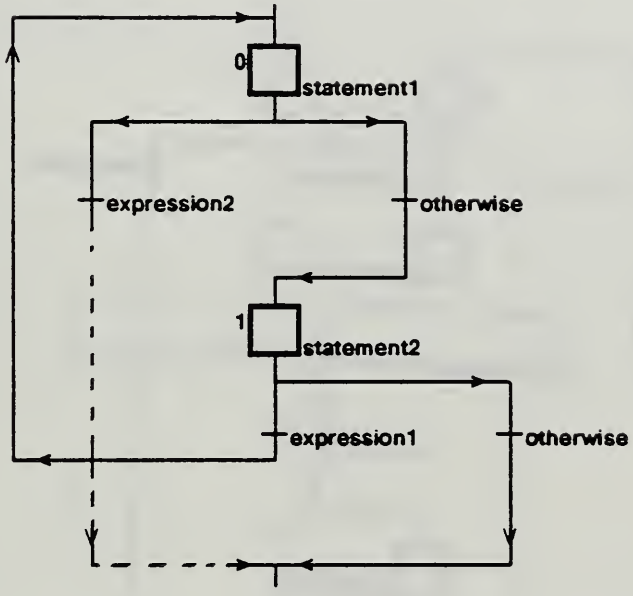
(**break** in a **while** statement in Grafcet)

```
for ( expression1 ; expression2 ; expression3 ) {
   statement1;
   if ( expression4 ) break;
   statement2;
   }
```



(**break** in a **for** statement in Grafcet)

```
do {
   statement1;
   if ( expression2 ) break;
   statement2;
   } while ( expression1 ) ;
```

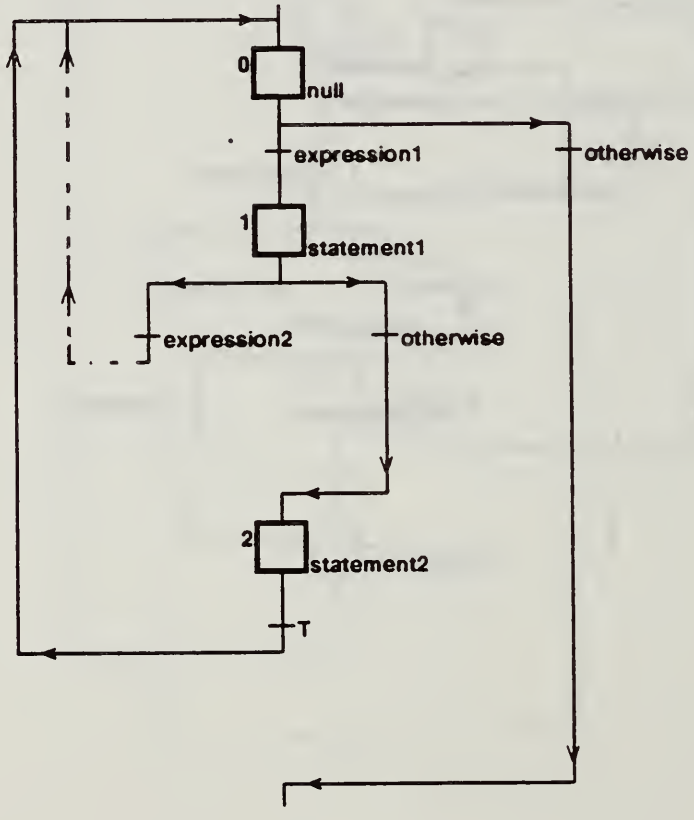


(**break** in a **do** statement in Grafcet)

Continue statement : **continue ;**

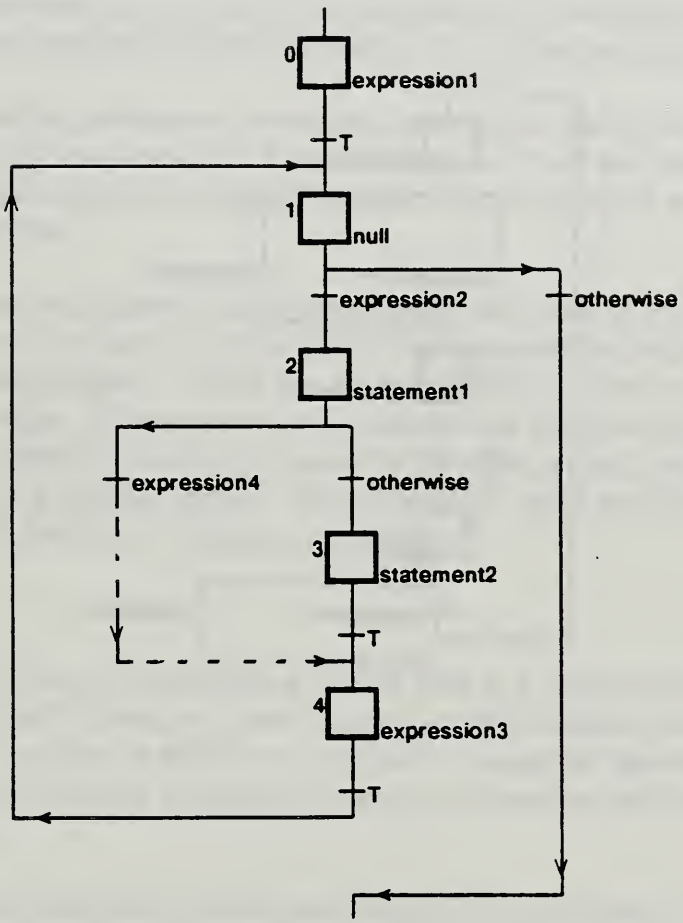
The **continue** statement causes control to pass to the loop-continuation portion of the smallest enclosing **while, do,** or **for** statement. Examples of the **continue** statement in the **while, do,** and **for** statements are shown below. The links in the examples which are added for the **continue** statements are shown as dashed lines.

Examples : **while** (*expression1*) {
statement1;
if (*expression2*) **continue;**
statement2;
}



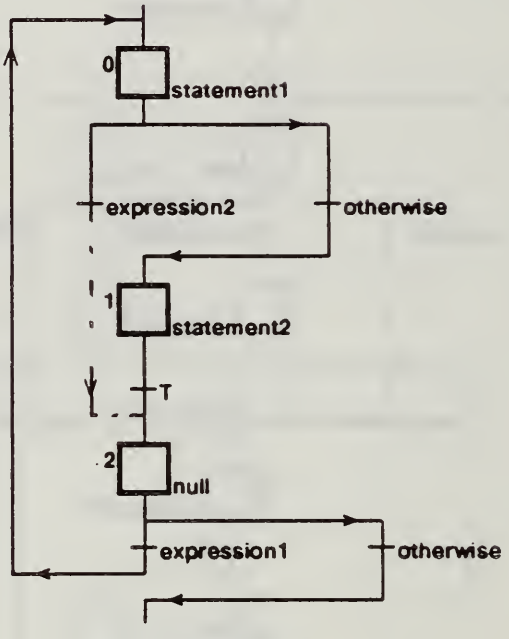
(**continue** in a **while** statement in Grafcet)

```
for ( expression1 ; expression2 ; expression3 ) {
    statement1;
    if ( expression4 ) continue;
    statement2;
    }
```



(**continue** in a **for** statement in Grafcet)

```
do {
    statement1;
    if ( expression2 ) continue;
    statement2;
    } while ( expression1 ) ;
```



(**continue** in a **do** statement in Grafcet)

Representing Function Calls as Macro Steps :

This is a discussion of some possible solutions to the problem of translating function calls to Grafcet macro programs, but this is not to be taken as a definitive translation scheme between C source code and Grafcet for functions. This section contains a description of the problems representing C function calls in Grafcet, a possible parameter passing scheme in Grafcet, and a possible solution to this limitation that a macro program can be used only once.

There are some problems inherent in the Grafcet language which makes representing function calls difficult. One problem is that macros do not support parameter passing. A second matter is that macros do not have return values. A third matter is that Grafcet programs become cumbersome when they become very large because of their graphical nature. A fourth matter is that return statements have to be included in the Grafcet macro programs that are associated with the enter and exit portion of the function. A final problem is that a Grafcet macro program can only be used once in a Grafcet system.

A possible solution to the problems of parameter passing is to assign the parameters to variables which are only used locally to a macro program. There are scoping rules in Grafcet that allow for making local copies of global variables. A return value would have to be passed as a global variable.

Consider the limitation of a Grafcet macro being called only once in a Grafcet system. If a function is called more than once in a C program, that function then has to be translated into multiple Grafcet macro programs. There has to be a copy of the Grafcet macro program each time the function is called. There also has to a unique name for each copy of the macro program generated. Another solution is to keep the function in the C programming language and add the function to a callable library. This in itself can cause problems with timing and mixing control flow in both the C code and the Grafcet.

Conclusion:

The paper provides a scheme for translating control flow in C source code into Grafcet. The control constructs are easily translated from C source code to Grafcet, but control flow through the use of C functions is difficult to translate. It might be appropriate to augment Grafcet in some way to allow for parameter passing, return values, and macros which can be called multiple times.

Grafcet is being used for designing and documenting Cell Controllers in the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards. A major language used for writing these controllers is C and this translation scheme was used to translate the high level logic of the Cell Controller [6]. Grafcet has provided an excellent means of visualizing the control structure of these large systems.

References:

[1] Savoir [1986] "Savoir Grafcet" Savoir, Oakland, CA.

[2] Kernighan, B. W., and Ritchie D. M. [1978] *The C Programming Language*, Prentice-Hall, Inc.

[3] Knuth, D. E. [1974] "Structured Programming with *go to* Statements", *Computer Surveys*, 6, 4, 261-277.

[4] Wirth N. [1974] "On the Composition of Well-Structured Programs", *Computer Surveys*, 6, 4, 247-259.

[5] Dijkstra, E. W. [1968] "Go To Statement Considered Harmful", *Comm. ACM*, 11, 3, 147-148.

[6] Pratt T. W. [1975] *Programming Languages: Design and Implementation,* Prentice-Hall, Inc.

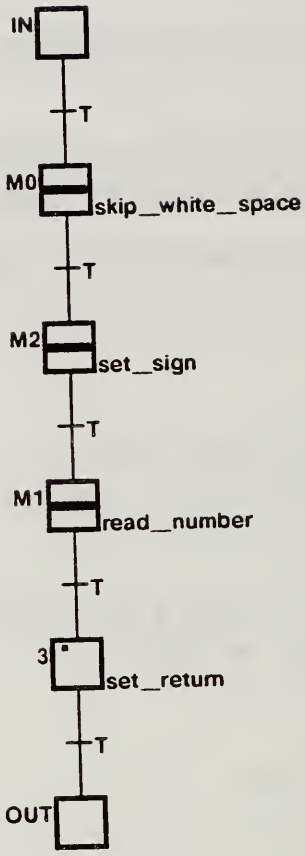
[7] Thomas B. H., and McLean C. [1988] "Using Grafcet to Design Generic Controllers" *First International Conference on Computer Intergrated Manufacturing* (to be presented).

Appendix:

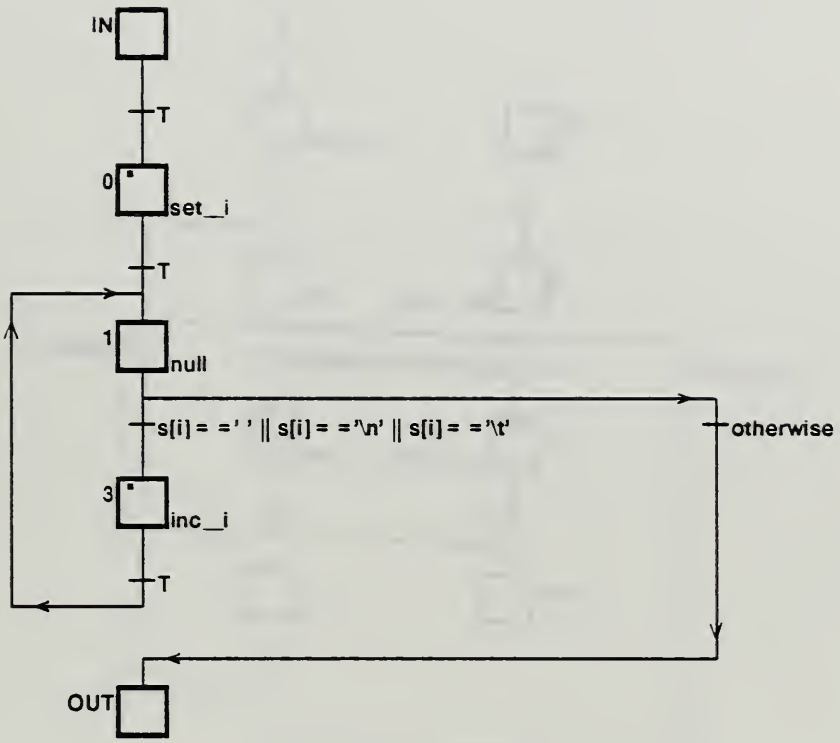
This section shows some examples of translations of C source code to Grafcet. The function "atoi", converts of an ASCII string to an interger number. In the example, the first section is the C source code and the second section is the Grafcet programs with associated C source code embedded in the regular steps.

Example :

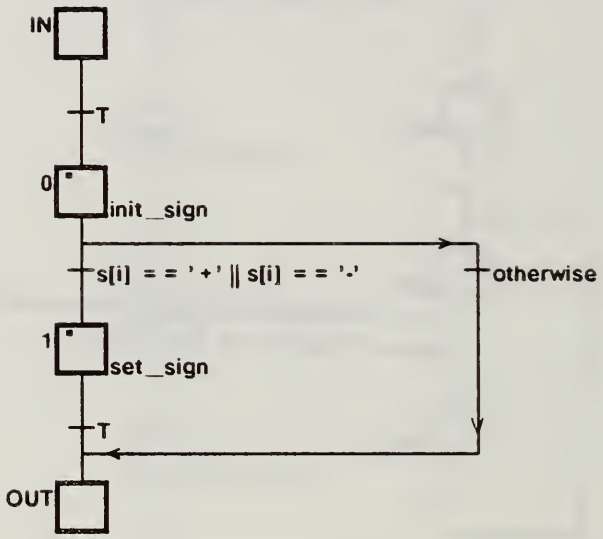
```
atoi ( s , return_int )    /* convert s to integer */
char s[];
int  *return_int;
{
     int i , n , sign;

     /* skip white space */
     i = 0;
     while (s[i]==' ' || s[i]=='\n' || s[i]=='\t')
            i++;

     /* set sign */
     sign = 1;
     if (s[i] == '+' || s[i] == '-')
         sign = (s[i++]=='+') ? 1 : -1;

     /* read_number */
     for ( n=0 ; s[i] >= '0' && s[i] <= '9'; i++)
          n = 10 * n + s[i] - '0';

     /* set return */
     *return_int = sign * n;
}
```



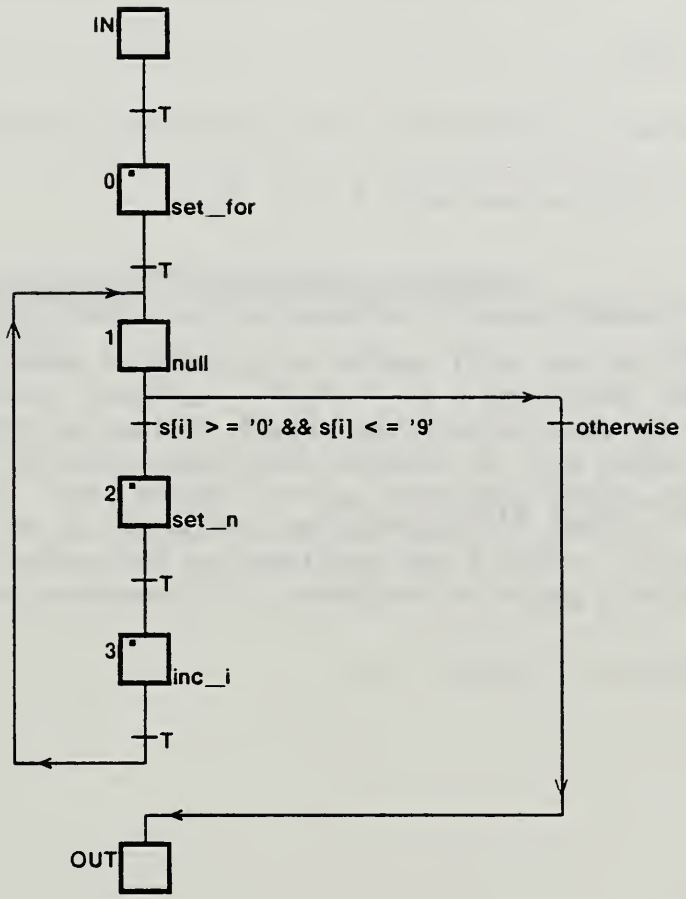
(atoi Grafcet program)



(skip_white_space Grafcet program)



(set_sign Grafcet program)



(read_number Grafcet program)

These blocks of code are associated with labeled regular steps in the Grafcet programs associated with the function atoi.

set_i:

```
i = 0;
```

inc_i:

```
i++;
```

init_sign:

```
sign = 1;
```

set_sign:

```
sign = (s[i++]=='+') ? 1 : -1;
```

set_for:

```
n = 0;
```

set_n:

```
n = 10 * n + s[i] - '0';
```

set_return:

```
return_int = sign * n;
```

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 88-3741	2. Performing Organ. Report No.	3. Publication Date MARCH 1988

4. TITLE AND SUBTITLE

A Scheme for Translating Control Flow in the C Programming Language to Grafcet With Example

5. AUTHOR(S)

Bruce H. Thomas

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The purpose of this paper is to show a translation scheme from control flow in the C programming language to the Grafcet language. Grafcet is a graphical language for expressing control flow. Grafcet is used to design parallel systems such as in a manufacturing environment. The control constructs covered in this paper are: conditional statement, while, do, for, switch, break, continue, goto, label, and null. The Grafcet used in this paper is the language, as augmented by Savoir. The C programming language is the one described by Kernighan and Ritchie. This translation is to be used as a reference for programmers to translate existing C source code into Grafcet.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Grafcet, C programming language, control flow, programming languages

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES 29

15. Price $11.95