5.0 Tutorial Examples

The following tutorials assume that the user has read Chapters 1,2 and 4 of this manual and has a basic familiarity with graphical interfaces which employ mice and menus to communicate with users, with X windows, and with UNIX commands and editors. These tutorials will guide you, step by step, through the process of creating and running CODE programs.

5.1 TUTORIAL: EXAMPLE 1

In this example we will be creating a CODE program that sums the elements of a 100 element vector. We first initialize two vectors of 50 elements each and then add the two vectors separately, in parallel. The resulting partial sums are then added to produce the final sum.

Contents of the Tutorial

I. Preliminaries
II. Entering Example 1
  Step 1. Draw all the nodes and arcs in a graph.
  Step 2. Use attribute forms to enter all information about the nodes, arcs, graphs and programs.
III. Executing Example 1 on the Sequent.
  Step 1. Save and compile the program.
  Step 2. Create the files which record the node function definitions.
  Step 3. Create the executable and run it.

I. Preliminaries

Get the X-Windows system running. Create a directory to hold the files associated with the program and *cd* to that directory. Before running CODE, ensure that it is present on your UNIX path (or be prepared to type a complete path name). Then enter

```
code2 sumvector.grf
```

at the UNIX prompt. This will run CODE and create a program called "sumvector." A single window will be opened and all activity will take place within it. Figure 4.1 shows the CODE window.

II. Entering Example 1

Our goal is to draw the following graph:

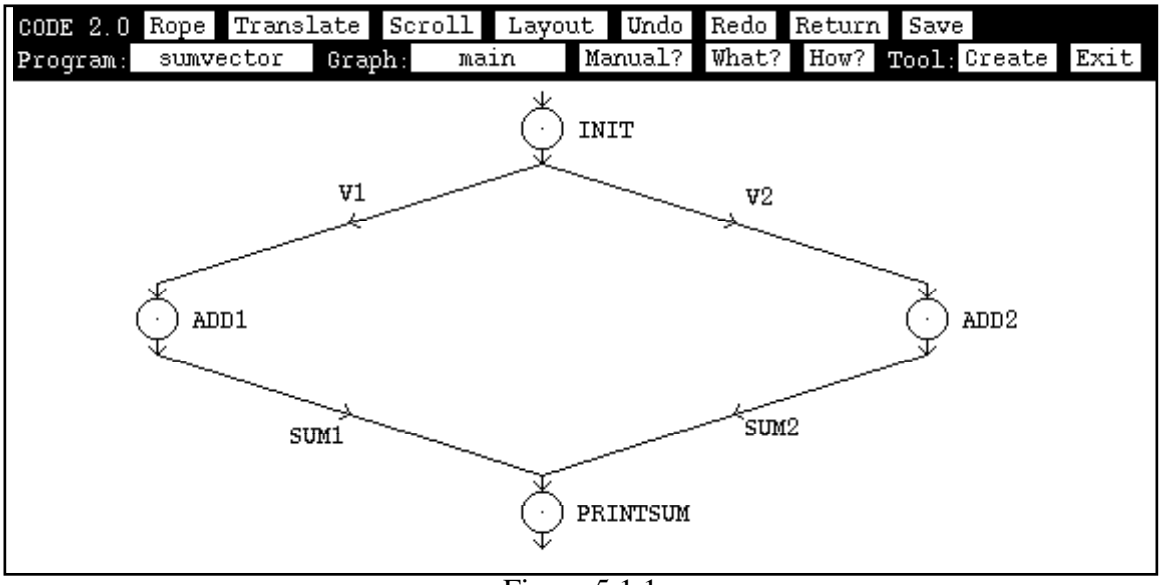

Figure 5.1.1

You may enter and annotate the nodes and arcs of your program in any order, but we will adopt the following sequence of operations:

Step 1. Draw all the nodes and arcs in a graph.
Step 2. Use attribute forms to enter all information about the nodes, arcs, graphs and programs.

Before we start, it is important to point out that you should save your work periodically. Then, in the (unlikely of course!) event that the CODE system crashes, your work will not be lost.

To save your work as you go, click on the Save button at the upper right of the banner line. By doing this regularly you can avoid losing your work in case of a system crash.

Correcting mistakes:

a. Finish, drawing the node or arc.
b. Select the erase cursor from the tool window.
c. Place the erase cursor on the object to be deleted and click the left mouse button.
d. If you want to undo the deletion, click on the 'Undo' button in the upper part of the CODE window.
e. Clicking on the 'Redo' button undoes the undo!
f. If you make any syntax errors in an attribute form, CODE will beep to let you know that you have made a mistake.

STEP 1. Draw all the Nodes and Arcs in a Graph

A. Draw all the nodes.

1.Draw the UC node, INIT by

a. Position the create cursor in the CODE window, where the UC INIT is to be located.
b. Click with the left mouse button, to bring up the menu of node types.
c. Pick the Unit of Computation (UC) node by clicking on it.

2. Draw the UCs ADD1, ADD2, and PRINTSUM.

.

B. Draw all the arcs.

1. Draw the arc V1 by,

a. With the create cursor, click on the lower arrow of UC, INIT.
b. Also, click on the upper arrow of UC, ADD1.

2. Draw the arcs V2, SUM1 and SUM2.

At this point the graph looks like Fig. 5.1.2.

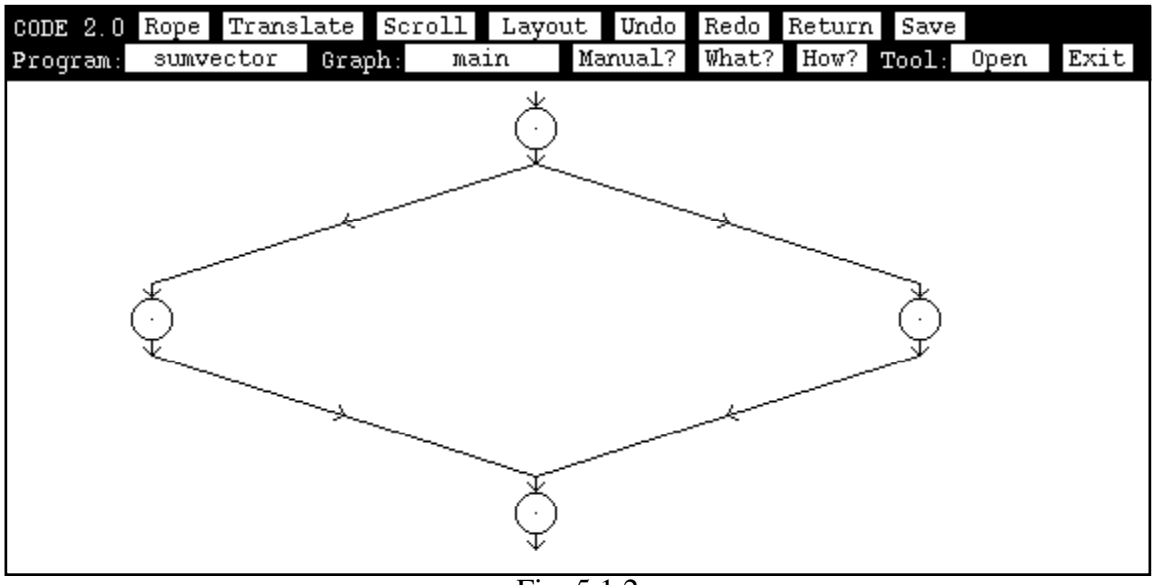

Fig. 5.1.2

STEP 2. Use Attribute Forms to enter all Information about Nodes, Arcs Graphs and Programs

A. Enter information about the nodes

1. Enter information about INIT

a. Select the open cursor from the tool window.

b. Click the open cursor on the UC INIT to open its attribute form. It looks like Fig.5.1.3

c. Click on the node name field and a text box will appear. Enter the name 'INIT' in this box. You can make the text box go away by clicking on the white space in the UC form

to return to it.

d. The value in the UID field is automatically generated by CODE. It is a debugging aid which you can ignore for now. (It will appear in error messages that CODE generates during the translation process.)

e. The termination node and start node fields have a default value of 'No'. Since INIT is a start node, update the Start node field by clicking on it and picking 'Yes'.

f. The Node Function Signatures and Node Function Definitions fields are to be left blank.

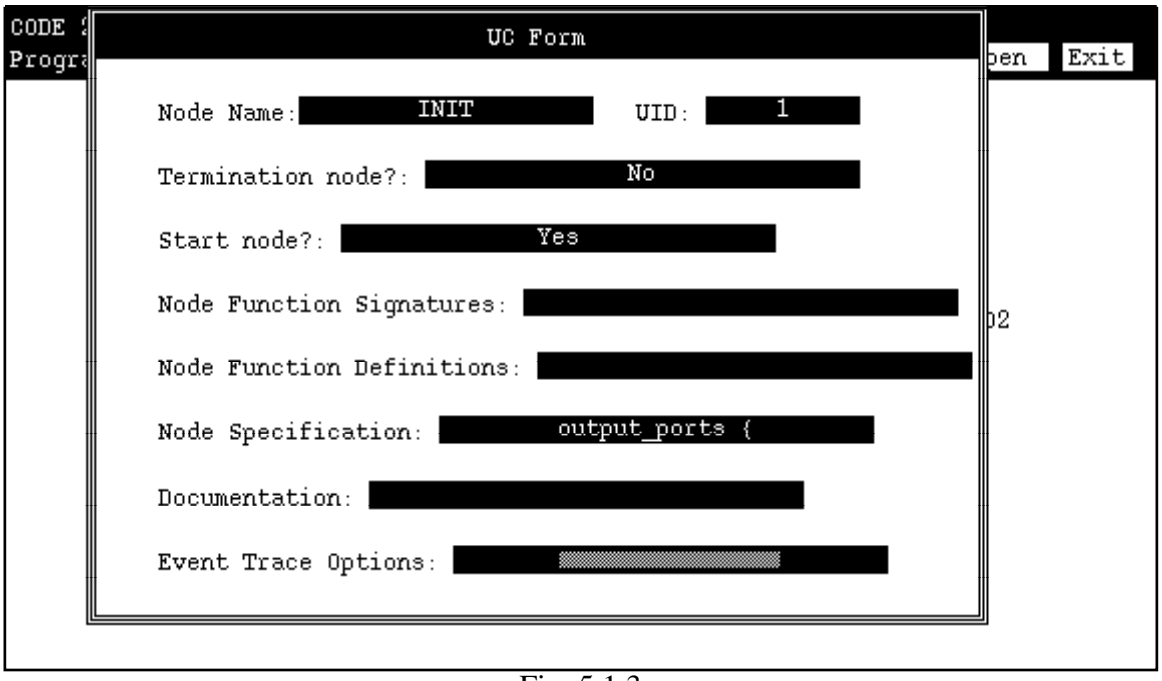

Fig. 5.1.3

g. Click on the Node Specification field. A text box labelled 'Text of UC Specs' will appear. Fig. 5.1.4 shows the code to be entered in this text box. This code can also be entered via a window running the editor defined by your EDITOR environment variable. To bring up this window press ESC-e (the escape key followed by e). After entering the code, you can exit from the window, the way you would normally exit from your editor.

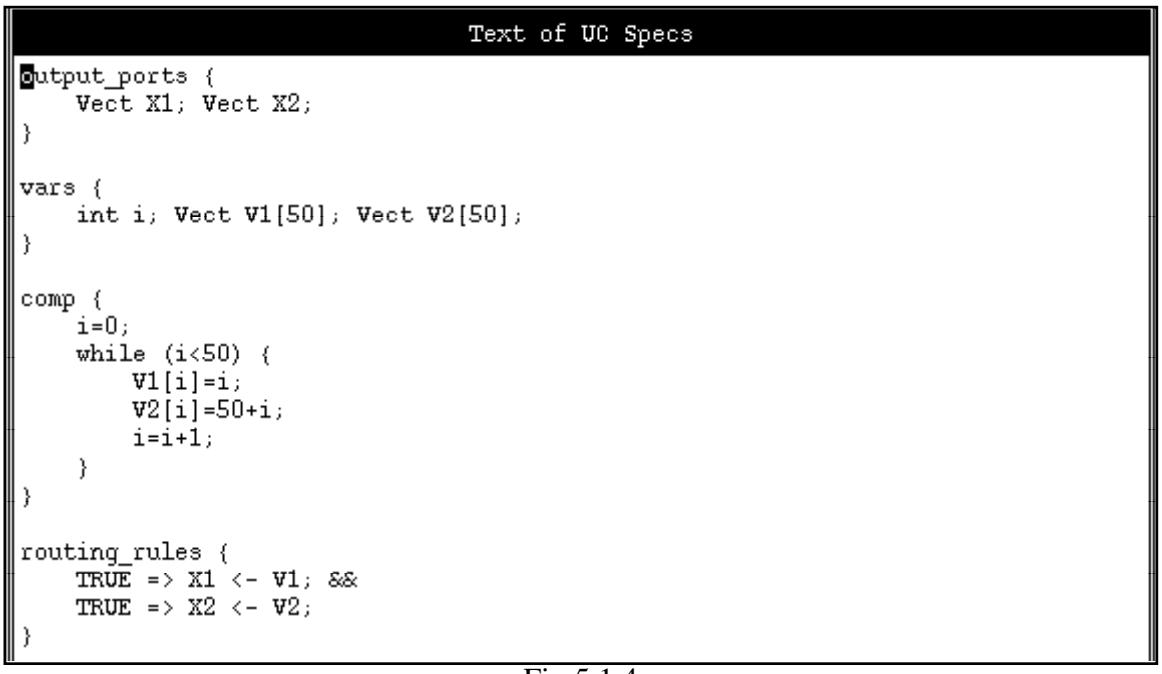

Fig.5.1.4

h. The documentation field is to be left blank.

i. To exit the UC INIT attribute form, click on the white space outside the form in the CODE window.

The name 'INIT' will be displayed next to the node, but you can move it with the Move cursor.

2. Enter information about ADD1 as in 1.

The one difference is that ADD1 is not a Start node. The code to be entered in the Node Specification field of ADD1 is as shown below,

```
input_ports {
 Vect Y1;
}

output_ports {
 int P1;
}

vars {
 Vect V1[50]; int i; int sum1;
}

firing_rules {
 Y1 -> V1 =>
}
```

```
comp {
 i=0;
 sum1=0;
 while (i<50) {
 sum1=sum1+V1[i];
 i=i+1;
 }
}

routing_rules {
 TRUE => P1 <- sum1;
}
```

3. Enter information about ADD2 as in 1.

The one difference is that ADD1 is not a Start node. The code to be entered in the Node Specification field of ADD2 is as shown below,

```
input_ports {
 Vect Y2;
}

output_ports {
 int P2;
}

vars {
 Vect V2[50]; int i; int sum2;
}

firing_rules {
 Y2 -> V2 =>
}

comp {
 i=0;
 sum2=0;
 while (i<50) {
 sum2=sum2+V2[i];
 i=i+1;
 }
}

routing_rules {
 TRUE => P2 <- sum2;
}
```

4. Enter information about PRINTSUM as in 1.

The one difference is that PRINTSUM must be designated as a termination node. The code to be entered in the Node Specification field of PRINTSUM is as shown below,

```
input_ports {
 int Q1; int Q2;
}
```

```
vars {
 int sum1; int sum2; int sum;
}

firing_rules {
 Q1 -> sum1, Q2 -> sum2 =>
}

comp {
 sum=sum1+sum2;
 print(sum1,sum2,sum);
}
```

B. Enter information about the arcs

1. Enter information about V1

   a. Click the open cursor on the arc V1 to open its attribute form. It looks like Fig. 5.1.5.

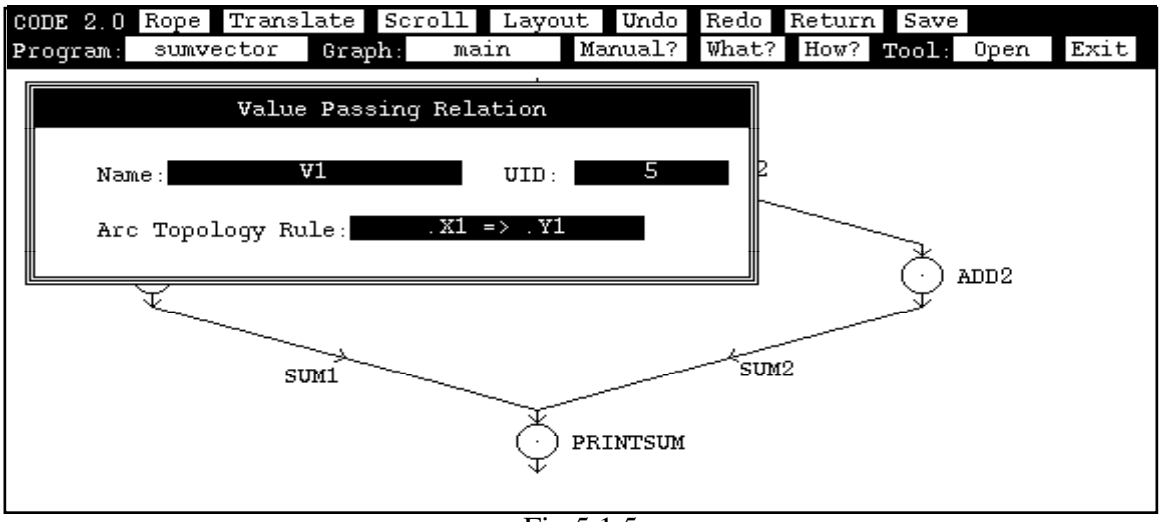

Fig.5.1.5

   b. Click on the name field and a text box will appear. Enter the name 'V1' in this box. You can make the text box go away by clicking on the white space in the Value Passing Relation form to return to it

   c. The value in the UID field is automatically generated by CODE.

   d. Click on the Arc Topology Rule field and another text box will appear. The code to be entered in this text box is shown below,

```
.X1 => .Y1
```

   e. Exit from both the text box and the form by clicking on the white space outside the two in the CODE window.

   The name 'V1' will be displayed next to the arc, but you can move it with the Move cursor.

2. Enter information about V2, SUM1 and SUM2 as in 1.

The Arc Topology Rule field of V2 is,

```
.X2 => .Y2
```

The Arc Topology Rule for SUM1 is,

```
.P1 => .Q1
```

The Arc Topology Rule for SUM2 is,

```
.P2 => .Q2
```

C. Enter information about the program

a. Click the open cursor on the Program button in the CODE window, to open its attribute form. It looks like Fig. 5.1.6.

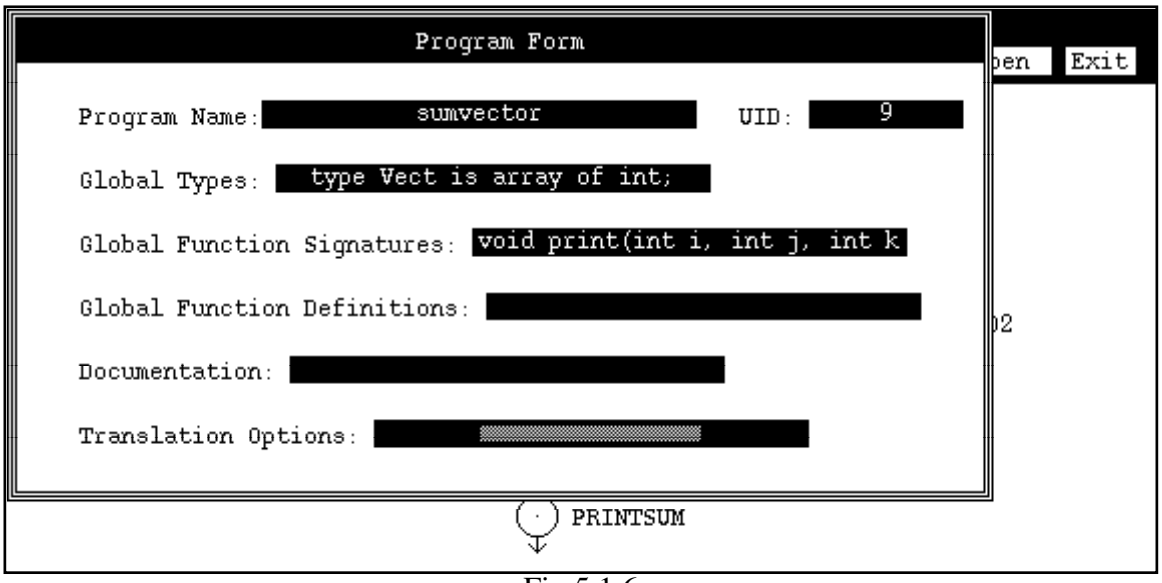

Fig.5.1.6

b. The values in the program name and UID fields are automatically generated by CODE.

c. Click on the Global Types field and a text box will appear. The code to be entered in this text box is shown below,

```
type Vect is array of int;
```

d. Click on the Global Function Signatures field and another text box will appear. The code to be entered in this text box is shown below,

```
void print (int i, int j, int k);
```

e. The Global Function Definitions and Documentation fields are to be left blank.

f. Click on the Translation Options field, to bring up it's attribute form. Now, click on the 'Object Files to Link' field. A textbox will appear. Enter the name of the file in which node function definitions will be specified, in this textbox.

```
filename.o
```

g. Exit from the program form by clicking on the white space outside the form in the CODE window.

III. Executing Example1 on the Sequent

We will now compile and run the program in the following order:

Step 1. Save and compile the program.
Step 2. Create the files which record the node function definitions.
Step 3. Create the executable and run it.

STEP 1. Save and Compile the Program

a. Click on the Save button.
b. Click the create cursor on the Translate button and select the 'Sequent' option.
c. If there are any errors, these are displayed in the window from where CODE was run.
d. Eliminate the errors, if any. They will likely be typographical.
e. After the program has been successfully compiled and saved, exit the program by clicking on the Exit button. Select the 'save' option and quit.

At this stage, the following files and directories will be present under the directory example1.

sumvector.grf	File
sumvector.sequent	Directory

STEP 2. Create the Files which record the Node Function Definitions

$cd ./sumvector.sequent The following files will be present in this directory.

Makefile
c2_globtype.h
c2_main.h
c2_main.c
main.c

$vi filename.c Record all node function definitions in this file. Give it the same name as the object file to be linked. The text to be entered in this file is as shown below. This file "routines.c" can also be picked up from the directory,
`examples/tutorial1/sumvector.sequent.`
These will be present in the directory in which CODE2 has been installed.

```
print (i, j, k)
int i, j, k;
{
printf("The first partial sum is %d\n",i);
printf("The second partial sum is %d\n",j);
```

```
printf("The total sum is %d\n",k);
}
```

STEP 3. Create the executable and run it

These operations will depend on how your workstation is connected to the Sequent. What you must do is get the generated source files onto a sequent and enter "make" to compile them. You must also have compiled any external source files that you wish to link with the CODE program. See the Chapter on 'Translating and Running Programs' for the details. At the University of Texas Department of Computer Sciences, the following command should be given to login onto the Sequent.

$rlogin qt

After having logged in on the Sequent, give the following commands,

$cc -c filename.c	Compile the file specifying node function definitions.
$make	This will create the executable 'sumvector'.
$./sumvector	Run the executable. The output displayed should be that shown below,

```
The first partial sum is 1225
The second partial sum is 3725
The total sum is 4950
```