USER GUIDE for a DATABASE of Categories Version 1.0 March 1996

M. Fleming R. Gunther
R. Rosebrugh [*]
Department of Mathematics and Computer Science
Mount Allison University
Sackville, NB E0A 3C0, Canada

1 Introduction

Use of the program requires a basic knowledge of category theory. It allows storage and manipulation of finitely-presented categories. The program was written in ANSI C and is menu-based for ease of use. Its format for storing categories, and some of the data structures are based on those developed by S. Carmody and R. F. C. Walters of Sydney University [1]. The program allows creation, editing and storage of finitely presented categories. In addition, there are several tools for testing properties of objects and arrows, and the computation of right and left Kan extensions of finite-set valued functors along finitely presented functors is available.

This document, available as
`ftp://sun1.mta.ca/pub/papers/rosebrugh/catuser.{tex,dvi}`
is devoted to explaining all of the options available to the user through the menus. For

[*]This work was done while the first two authors were supported by NSERC Canada Undergraduate Summer Research Awards

further information consult the *Guide to Data structures and Algorithms* which is available as
`ftp://sun1.mta.ca/pub/papers/rosebrugh/catdsalg.{tex,dvi}`
The program is stored in executable form (Sun Sparc1+) as
`ftp://sun1.mta.ca/pub/sources/rosebrugh/unix/category.exe`
(the C source code is in the same directory.) A DOS executable is stored as
`ftp://sun1.mta.ca/pub/sources/rosebrugh/DOS/category.exe`

2 The Main Menu

When the program starts the Main Menu is displayed. The menu looks as follows:

```
                         Categories Database

                    (1)  Category Menu
                    (2)  Functor Menu
                    (3)  Category Tools
                    (4)  Right Kan Extension
                    (5)  Left Kan Extension
                    (6)  Change maximum order of endomorphisms

                    (0)  Quit

Your choice...
```

To choose an option type the number of your choice and $<CR>$ ('Enter'). If an invalid character is typed the program will re-display the menu and wait for the user to make another choice. Most of the menu options will call up another menu, while others will actually get the user to enter some form of input. The first five choices will be discussed later in this document, but we will first look at options (0) and (6).

`(0)Quit`
This option allows you to leave the program. However, before you exit, for each category or functor currently in memory that has not been saved you will be prompted as to whether or not you wish to save it. Once you have responded to all of the prompts the program will terminate.

`(6)Change maximum number of endomorphisms`
Although many of the manipulation tools below require a finite category, it is possible to store any finitely presented category. *Correct results from use of the tools require that endomorphisms be of finite order with a prespecified bound.* This menu option allows you to control the maximum number of times an endomorphism will be traversed by some of the tools in the program. The default value is 2. If you choose this menu option, the current value will be displayed, followed by the prompt:
`New value>`
Simply enter the new value and hit <*CR*>.

3 The Category Menu:

Selecting option one, `Category Menu`, from the Main Menu will display the following menu:

```
              CATEGORY MENU

        (1)   Create category
        (2)   Load category
        (3)   Edit category
        (4)   Display category
        (5)   List current categories
        (6)   Save category
        (7)   Remove category

        (0)   Back to main menu
```

Again, type the number of the selection you want and <*CR*>. Option three, `Edit category`, calls a new menu and will be discussed later. The other selections are covered below.

`(1)Create Category:`
Once this option is chosen the following prompt will be displayed:
`Category name:`
The program is waiting for the user to type in the name of the new category to be created. There are no restrictions as to what the name can be, however it is recommended that it be fairly short and somewhat descriptive of the category. Once a name has been picked and typed in, the screen will be cleared again and at the bottom of the screen you will see:

```
Enter @ to display all objects
Enter object name (type `enter´ when finished) :
```

This is where you enter the objects. An object name must be a single character and all characters are valid except for @. If more than a single character is entered for an object name then the program will name the object with the first character entered. After each object name is typed in, type <CR> to proceed. If at any time you wish to view the objects already entered, type in '@' and <CR>. When all objects in the category are named, type <CR> at the prompt. **Warning:** *no two objects may have the same name.* If a duplicate object name is entered the program will not accept it. An error message will be displayed, and the new object will not be included in the category.

Once the last object has been entered the following will be displayed:

```
Enter @ to display all arrows
Enter arrow name (type `enter´ when finished) :
```

Entering arrows is very similar to entering objects. '@' followed by <CR> will display all of the arrows entered so far and once the last arrow has been entered, type <CR> at the prompt. After the name of an arrow has been entered, the program will ask the user to input the domain of the arrow and then the codomain of the arrow. The program will not accept any arrow which has a duplicate name, or an invalid domain or codomain. The domain and codomain must be objects of the category; otherwise, the arrow will not be added. Furthermore, an arrow cannot be named `1`, the symbol reserved by the program for the identity arrow.

Next the user enters the equations among composites of the generating arrows. After the last arrow has been entered the following will be displayed:

```
Enter @ for left side to display all arrows and relations
Enter left side of equation:
```

To enter an equation, type in the left side of the equation and <CR>. Then type in the right side of the equation and <CR>. If you wish to have some path of arrows in the category equal the identity, type in this path for the left side and enter 1 for the right side of the equation. To display the current arrows and equations type in '@' for the left side and then <CR> for the right side. Once the equations have been entered, typing <CR> for both the left and right sides of the equation will bring back the Category Menu and the newly created category will be stored in memory. The program will only accept valid equations. An equation is invalid if it contains arrows that are not in the category, or an illegal composition, or if the the domain and codomain of the left side do not equal the domain and codomain of the right side.

`(2)Load Category`
Choosing this option will clear the screen and display the following prompt at the bottom of the screen:
```
(-1 to cancel)
Enter name of category you wish to load>
```
If you wish to cancel this operation, type -1 followed by $<CR>$. Otherwise, enter the name of the file containing the category you wish to load and the program will load the category in memory. If the program cannot locate the file, an error message will be displayed and you will be asked if you wish to try again. Type in 'y' or 'n'; 'n' will bring back the Category Menu without any category having been loaded.

`(4)Display Category`
This option will clear the screen and display a list of the categories currently in memory. The categories are listed by their actual name, not their file name. The following prompt will then be displayed:
```
(-1 to cancel)
Category to display >
```
To choose a category type in the number of that category and $<CR>$. Enter -1 to cancel, and you will be prompted to type $<CR>$ to return to the Main Menu. If an invalid number is entered the list will be displayed again and you will be asked to make another selection. The category you choose will be displayed on the screen. To exit to the Category Menu type $<CR>$.

`(5)List current categories`
Choosing this menu option will display a list of the categories that are currently in memory. Once again it displays the actual category names, not the file names. Typing $<CR>$ will bring you back to the Category Menu.

`(6)Save Category`
This selection clears the screen and displays a list of the categories currently in memory. The following prompt is displayed at the bottom of the screen:
```
(-1 to cancel)
Category to save >
```
To make a selection type in the number of the category you wish to save and type $<CR>$. An invalid number will result in the list and prompt being re-displayed. Once a category has been chosen the screen will be cleared again and a new prompt will be displayed at the bottom of the screen:
```
Enter file name for storing category:
```

Now enter any name up to eight letters long. Saving the category under this name would be sufficient, but we recommend adding the extension .cat to make it clear that this is a file that contains a category. If the name that you have chosen to save your category under is already the name of a file on disk, the program will inform you and ask if you wish to erase the old file and replace it with your new file. If you don't wish to erase the old file type in 'n' and type $<CR>$. You will then be prompted to enter a new file name. After you have saved your category to a file, the Category Menu will be redisplayed.

`(7)Remove Category`
This selection will clear the screen and display a list of the categories currently in memory. The following prompt will then be displayed:

```
(-1 to cancel)
Category to remove >
```

Now type in the number of the category you want to remove from the list of categories currently in memory. This category will then be erased from memory. Removing unneeded categories from memory is extremely important as the computer has a limited amount of memory and each category in memory requires a significant amount. After you have chosen a category to remove from memory the Category Menu will be redisplayed on the screen.

`(0)Back to Main Menu`
This option will return program control to the Main Menu.

4 The Category Edit Menu:

Choosing option three, `Edit category`, from the Category Menu will clear the screen and display a list of the categories currently in memory. Enter the number of the category you wish to edit and type $<CR>$. A new menu will be displayed, the Category Edit Menu:

```
Category Edit Menu
(1)  Change Name
(2)  Add Objects
(3)  Add Arrows
(4)  Add Relations
(5)  Remove Objects
(6)  Remove Arrows
(7)  Remove Relations
```

```
                         (8)  Display Category

                         (0)  Exit to Category Menu

Your choice...
```

To choose an option, type the number of the selection you want and $<CR>$. All of these menu options deal with the specific category you previously selected.

`(1)Change Name`
This selection causes the screen to be cleared and the following prompt to be displayed at the bottom of the screen:
`Category Name:`
Type the new name you wish to give this category and $<CR>$. After typing $<CR>$ the Category Edit Menu will be displayed again.

`(2)Add Objects`
Choosing this option will clear the screen and display the following at the bottom of the screen:

```
Enter @ to display all objects
Enter object name (type `enter´ when finished) :
```

To add an object to the category type in a character for the name of the object and $<CR>$. Remember, two objects cannot have the same name. To see a list of the current objects in the category type in '@' and type $<CR>$. When you are finished adding objects type $<CR>$ to return to the Category Edit Menu.

`(3) Add Arrows`
This choice will display a list of objects in the current category and then the following message:

```
Enter @ to display all arrows
Enter arrow name (type `enter´ when finished) :
```

Again, type in the name of a new arrow and $<CR>$, and you will then be prompted for the arrow's domain and codomain. Remember that the domain and codomain must be objects in the category, and that two arrows may not have the same name. A list of arrows currently in the category may be seen by typing '@' and $<CR>$.

`(4) Add Relations`
Choosing this option will display a list of arrows and equations currently stored in the category, followed by the message:

`Enter @ for left side to display all arrows and relations`
`Enter left side of equation:`
Now the procedure is the same as for entering equations when creating a category. Type in the left side of an equation and <CR>, and then do the same for the right side. '1' may be entered for the right side to represent the identity. Entering '@', followed by <CR>, will once again display all arrows and equations, while typing <CR> by itself for both the left and right sides will return you to the Category Edit Menu. Again, the program will not accept equations which contain arrows not in the category, invalid compositions, or left and right sides with unequal domains or codomains.

`(5) Remove object`
This selection will display a numbered list of objects in the category, and then the following prompt:
`Object to be removed (-1 to cancel):`
If you decide not to delete an object, enter -1; otherwise, enter the number of the object you wish to delete and then <CR>. If there are arrows in the category which have this object as their domain or codomain, the display will show:
`Object has arrows dependent on it.  Unable to remove.`
Type <CR> to clear this message and you will return to the Category Edit Menu. The object will not be removed. If you still wish to delete this object, then remove all dependent arrows by selecting `Remove Arrows` from the menu, and then choose `Remove Object` again to delete the object.

`(6) Remove Arrows`
This option is very similar to the `Remove Objects` option. A numbered list of arrows will be displayed followed by the prompt:
`Arrow to be removed (-1 to cancel):`
Enter the number of the arrow you wish to delete (or -1 to cancel), and then <CR>. If this arrow is contained in any of the category's equations, the message
`Arrow has equations dependent on it.  Unable to remove.`
will appear. Type <CR> to clear the message and return to the Category Edit Menu. Then, if you still want to delete the arrow, remove all necessary equations from the category (by selecting `Remove equations`) and then return to `Remove Arrows`.

`(7) Remove Relations`
Choosing (7) will display a numbered list of equations in the category. Enter the number of the equation you wish to delete (or -1 to cancel) and then <CR>.

`(8) Display Category`
This selection will display a list of categories currently in memory, followed by the prompt:
`Category to display>`
Enter the number of the category you wish to see, and <*CR*>. The category will be shown on the screen. Press <*CR*> to return to the Category Edit Menu.

`(0) Exit to Category Menu`
This will return control to the Category Menu.

5 The Functor Menu:

Choosing option 2 from the Main Menu, `Functor Menu`, will clear the screen and display the Functor Menu:

```
                         FUNCTOR MENU

                    (1)  Create functor
                    (2)  Create functor to SET
                    (3)  Load functor
                    (4)  Load functor to SET
                    (5)  Remove functor
                    (6)  Save functor
                    (7)  Display functor
                    (8)  List current functors

                    (0)  Exit to main menu

Your choice>
```

Type the number of the option you would like, and <*CR*>.

`(1) Create Functor`
Once this option has been selected, a list of categories in memory will be displayed, followed by the prompt:
`(-1 to cancel)`
`Functor from>`
Select the appropriate number from the list to choose the domain category for your newly

created functor (or type -1 to cancel). This will bring up the prompt
```
Functor to>
```
at which time you should enter the number of the codomain category of the functor. After this has been done, you will be prompted to enter the functor name. The screen will be cleared and you will see a display of the two categories, followed by a prompt similar to the following:
```
(Object) X-->_
```
In this example, X is an object in the first category, and you are being asked to enter the object in the second category that is the image of X under the functor. The program will not accept any input which is not an object in the codomain category. Notice that the categories are still displayed directly above the prompt so that you do not need to remember the names of the objects. This is repeated for each object in the domain category. Once you have defined the functor on objects, you will be prompted to do the same for arrows:
```
{-1 to quit} (Arrow) f-->_
```
Enter the *path* in the second category that is the image of f under the functor. This input can be a single arrow or a composable string of arrows in the second category. However, the path must have as its domain the object that is the image of the domain of f under the functor. Similarly, the codomain of the path must be the image of the codomain of f. For instance, if f is an arrow from A to B in the first category, the path you enter must have domain $F(A)$ and codomain $F(B)$, where F is the functor. Once you have successfully defined the functor for arrows, the program will check to see that all equations in the domain category hold for values of the functor in the second category. If all equations hold, then the creation of the functor is complete and it will be stored in memory. If equations do not hold, an error message will be displayed, such as
```
Relation gf = h does not hold in category B.
```
In this case, the functor is not well-defined and therefore has not been successfully created. You must type <*CR*> to clear the error message and then select `Create functor` to attempt the operation again.

`(2) Create functor to SET`
This selection is quite similar to the `Create functor` option. Here, you will be able to create a functor to the category of finite sets, $\mathbf{set}_0$. Since this functor always has the same codomain, you only need to respond to the
```
Functor from>
```
prompt, and then to enter the name of the set-valued functor. Once this is accomplished, you will see the source category displayed on the screen, followed by a prompt such as:
```
(Object) A-->
```

At this prompt, enter a number representing cardinality of the finite set which is the image of A under the functor. In fact we consider only the objects in the finite-cardinal skeleton of **set**$_0$ *Note: An object can be mapped to the empty set by entering 0.* For example, entering 3 will tell the program that A is taken to the set $[3] = \{1, 2, 3\}$. When this has been done for each object, you will be prompted to define the set functor on arrows. This prompt will appear as, for example:

```
{-1 to quit} (Arrow) f : [2] --> [3].
```

In this example, the domain of arrow f is an object which has as its image the set [2], and the codomain of f has as its image the set [3]. If, for instance, your set functor is named X, you will be asked to define $X(f)$ as follows. For each element in the domain (in this case [2]), you will be prompted for the element in the codomain ([3]) to which that element is sent by the functor. Again, using the example of f, since the domain has two elements, you will see the prompt

```
1 --> _
```

followed by

```
2 --> _
```

For each of these, you must enter an element of the codomain; in this case, an integer from 1 to 3. So, if $X(f)(1) = 3$ and $X(f)(2) = 2$, you would enter 3 and 2 respectively at these prompts. If you input an invalid number (4 for example), the computer will repeat the prompt. This entire process will be repeated for each arrow in the source category. Once the functor is defined on arrows, the program will check to see that all equations in the domain category hold under the functor as defined. If they do, the functor creation is complete and it will be stored in memory. If one or more equations do not hold, you will see an error message. Type <*CR*> to clear the message, and select `Create functor to SET` to attempt the creation of the functor again.

`(3) Load functor`

This option allows you to load a functor saved on disk. You will see the prompt:

```
Enter file name for functor you wish to load> _
```

Now, type the filename where the functor is stored. Be careful that you have selected the correct type of functor. **Warning:** *If you attempt to load a SET functor when the program is expecting a functor to a finitely-presented category, an error message will be displayed and the program will terminate.* If the functor is successfully loaded, you will see the following:

```
*****SUCCESS*****
type <return> to go back to Main Menu...
```

The functor has now been loaded and stored in memory. Type <*CR*> to continue.

`(4) Load functor to SET`
This is basically identical to choice (3), except that it allows you to load a stored functor from one of your loaded categories to the category of sets.

`(5) Remove functor`
This selection will display a numbered list of functors currently stored in memory, followed by the prompt
`Functor to remove> _`
Select the number of the functor you wish to delete from memory and <*CR*>. The functor will be removed and the Functor Menu will return to the screen.

`(6) Save functor`
This choice will display a list of functors in memory, and then the prompt:
`Functor to save> _`
Enter the appropriate number, and <*CR*>. You will be asked to type in the filename under which you would like to save the functor. If this file already exists, the message
`<filename> already exists.  Overwrite (Y/N)? _`
will appear. If you wish to overwrite the old file, enter 'y', the functor will be saved, and the program will return to the Functor Menu. If you do not wish to overwrite the existing file, enter 'n', and you will be prompted for a new filename.

`(7) Display Functor`
Again, you will see a numbered list of available functors. Enter a number and <*CR*>, and the corresponding functor will be displayed. This display will include the functor name, and the image of each object and each arrow under the functor. Type <*CR*> to return to the Functor Menu.

`(8) List current functors`
Selecting this option will display a list of functors currently in memory. Type <*CR*> to clear the display and return to the Functor Menu.

`(0) Exit to main menu`
Making this selection will return control to the Main Menu.

6 The Category Tools Menu

Choosing option 3 from the Main Menu, `Category Tools`, will clear the screen and display a list of all the categories currently in memory. Type in the number of the category you wish to work with and <CR>. A new menu will be displayed, the Category Tools Menu:

```
                        CATEGORY TOOLS

                    (1) Make Confluent
                    (2) Initial Object?
                    (3) Equality of Composites
                    (4) Make Dual
                    (5) Sum?
                    (6) Display Category

                    (0) Exit to Main Menu

   Your choice...
```

Type in the number of the option you would like and then type <CR>.

`(1) Make Confluent`
Choosing this option will cause the program to make the set of equations in the current category confluent by adding new equations if necessary. For a discussion of confluent sets of equations see Chapter 7 of *Categories and Computer Science* by R. F. C. Walters [2]. As equations are added, you will be notified with a message such as

```
The equation ghkm = fhm has been added to the set of equations.
```

When the process is completed, or if no equations need to be added to make the category confluent, the following message will appear:

```
The set of equations is now confluent...
Type `enter´ to continue
```

Typing <CR> will then return you to the Category Tools Menu.

`(2) Initial Object?`
Warning: *Before choosing this option, be sure that the current category is confluent; choose option (1) if unsure.*
After displaying the current category, this option will allow you to choose either to test all objects in the category, or to test one specific object. The program will then show you whether or not an object is an initial object in the category. If it is initial, the display will read, for example:

```
A is an initial object!
```

An object may be not initial for two different reasons. It may have no path to one or more objects in the category. In this case, the message might read:

```
NOT INITIAL...no path to object C
```

An object might also have two or more (unequal) paths to a particular object, in which case the message might read:

```
NOT INITIAL...gh and f are both paths to object B
```

You will then be prompted to type <*CR*> to return to the Category Tools Menu.

`(3) Equality of Composites`

This feature will determine if two composable paths in the current category are equal. **Warning:** *Before choosing this option, be sure that the current category is confluent; choose option (1) if unsure.* The category will be displayed on the screen, and you will be prompted to enter two paths. If these paths are equal, given the current equations, the message (for example)

```
ghkm and fhm1 are equal paths.
```

will be displayed. If the two paths are not equal, you will see (for example)

```
ghkm and fhm are NOT equal paths.
```

Then type <*CR*> to return to the Category Tools Menu.

`(4) Make Dual`

This selection will create the dual (opposite) of the current category and store it on disk. You will be prompted:

```
Name of file for dual category:
```

At this prompt, enter the name of the file in which you wish to save the dual category. You will then be asked to

```
Enter name of DUAL category>
```

Here you should enter the name that you wish to give to the dual category (not the filename, but the name of the category itself). Once this has been done, the dual category will automatically be loaded into memory. Control is then returned to the Category Tools Menu.

`(5) Sum?`

Choosing this item will determine if an object and two paths , the candidate injections, into the object (which we will call α and β) represent a sum in the category. **Warning:** *Before choosing this option, be sure that the current category is confluent; choose option (1) if unsure.* You will be prompted first for the object, then for α, and then β. The program will not accept entries for α and β which do not have the given object as codomain, nor will it accept objects or paths which are not in the category. The final prompt will be similar to this:

```
Enter maximum number of loops (default = 2):
```

This variable controls the order of endomorphisms the program will test when checking the sum. The default value is the same number which appears in the Main Menu as the maximum order of endomorphisms. If you were to enter 3, the program will not test any paths which visit an object more than 3 times. Once you have entered this value, you will informed as to whether or not the object and paths are a sum (assuming the number of visits permitted is sufficient to generate all distinct arrows of the current category.)

`(6) Display Category`
This selection will display the active category on the screen.

`(0) Exit to Main Menu`
This returns control to the program's Main Menu.

7 Kan Extensions of $\mathbf{set}_0$-valued Functors

Warning: *If you intend to use a particular functor in a right or left Kan extension, and you have edited one of the categories involved in the functor, the functor itself* **must** *be re-created.*

7.1 Right Kan Extension

Option 4 from the Main Menu, `Right Kan Extension`, will allow the user to have the program compute a right Kan extension. Before choosing this, the user must have loaded all categories and functors which are to be used in the desired right Kan extension. If you wish to cancel the operation at any time, simply enter -1 at one of the prompts.

The user will first see a list of the currently loaded categories, and will be asked to select category $\mathbf{A}$ and then category $\mathbf{B}$ for the right Kan extension. Then, a list of currently loaded functors will be displayed and the user will be asked to select a functor F from category $\mathbf{A}$ to category $\mathbf{B}$, followed by a functor from $\mathbf{A}$ to the category of sets. The user will then be asked to enter a file name in which to store the output. Simply hit $<CR>$ if you do not wish to save the output.

Before proceeding with the right Kan calculation, the program will automatically ensure that both category $\mathbf{A}$ and category $\mathbf{B}$ contain a confluent set of relations.

The output for the right Kan extension is somewhat complex. Consider an object B in category $\mathbf{B}$. For this object B, the category B/F is calculated, where F is the functor from $\mathbf{A}$ to $\mathbf{B}$. The objects of this category B/F are all of the form $\beta : B \to FA$, where β is a path of arrows in category **B** and A is some object in category **A**. Now suppose that $\beta : B \to FA$ and $\beta' : B \to FA'$ are two objects in B/F. Then, an arrow f from β to β' would be an arrow $g : A \to A'$ in category **A** such that $F(g) = f$. We then construct a full sub-category $J(B/F)$ of B/F whose inclusion in B/F is initial.

The objects of this newly-constructed category $J(B/F)$ are the first thing displayed for an object B in the right Kan output. $R(B)$ is computed as a subset of the product:

$$\prod_{\substack{\beta : B \to FA \\ \beta \text{ in } J(B/F)}} X(A)$$

See the *Guide to Data Structures and Algorithms* for more details about how these tuples are computed. After the objects of $J(B/F)$ are displayed, the tuples in R(B) are listed. If R(B) is empty, the number 0 will appear. If there were no objects in $J(B/F)$, R(B) will be displayed as 1.

To the right of each tuple will be a column for each object A in category **A** with $B = FA$. This column provides information about the A component $\rho_A : RF(A) \to X(A)$ of the natural transformation $\rho : RF \to X$ required to complete the specification of the right Kan extension If R(B) was 0, ρ_A will appear as 1 if $X(A) = 0$ or as ! (the unique arrow) if $X(A) \neq 0$.

After all of the tuples have been displayed with the ρ information, they will be redisplayed along with information about the action of the right Kan extension on arrows of category **B**. The user will first see the objects of $J(B/F)$ again, and then the objects of $J(B'/F)$ for each object B' which is the codomain of an arrow out of B. Then, the tuples of R(B) are redisplayed. This time, to the right of each tuple, for each arrow $f : B \to B'$, you will see where R(f) takes that tuple in R(B'). **Warning:** *The program may not have enough information to fully compute the right Kan extension. This might occur, for example, in a finitely-presented infinite category.* If any information cannot be successfully calculated, a ? will be displayed.

This entire procedure will be repeated for all objects in category **B**.

Consider the following example. Suppose category **A** consists of three objects A, B, C with arrows $f : A \to B, g : A \to B, h : C \to B$, and suppose category **B** contains two objects D, E with one arrow $k : D \to E$. Suppose also that $F(A) = F(C) = D, F(B) = E, F(f) = F(g) = F(h) = k, X(A) = [4], X(B) = [3], X(C) = [3], X(f) = (1,3,2,3), X(g) = (2,3,2,2), X(h) = (2,3,2)$. The output would similar to the following:

```
Right Kan Extension ... object D

a1 = (1,A)     a2 = (1,C)

R(D)       rho(A)   rho(C)
~~~~       ~~~~~~   ~~~~~~

a1 a2
~~ ~~

 2  2         2        2
 3  1         3        1
 3  3         3        3


Right Kan Extension ... object D

R(D):  a1 = (1,A)     a2 = (1,C)
R(E):  b1 = (1,B)


          R(k)
R(D)      R(E)
~~~~      ~~~~

a1 a2     b1
~~ ~~     ~~

 2  2      3
 3  1      2
 3  3      2



Right Kan Extension ... object E

a1 = (1,B)

R(E)      rho(B)
~~~~      ~~~~~~

a1
```

```
~~
 1          1
 2          2
 3          3


Right Kan Extension ... object E

R(E): a1 = (1,B)

R(E)
~~~~

a1
~~
 1
 2
 3
```

8 Left Kan Extension

The algorithm used below is an implementation of the Todd-Coxeter algorithm as described in Chapter 7 of Walters' book [2].

The user input for a left Kan extension is identical to that for a right Kan extension. The user must provide the program with categories **A** and **B** , a functor $F : \mathbf{A} \to \mathbf{B}$, a functor $X : \mathbf{A} \to \mathbf{Set}$, and a file name for output.

The left Kan extension is calculated using the generalized Todd-Coxeter procedure. See the *Guide to Data Structures and Algorithms* for more information.

The output of the left Kan extension begins with information about the natural transformation $\epsilon : X \to LF$. For each object A in category $\mathbf{A}$, $\epsilon_A : XA \to LFA$ is shown. This might appear something like the following (where $B = FA$ and $XA = [3]$):

```
  eA
XA-->LB
 1    2
```

```
 2     3
 3     4
```

After this, you will see the action of the left Kan extension L on the objects and arrows of **B**. Each object B in **B** will be displayed with all of the elements of $L(B)$ below it. To the right will appear all generating arrows out of B with their action under L below. For example, if B is an object which is the domain of two arrows, $f : B \to C$ and $g : B \to D$, the output might appear as follows:

```
      Lf  Lg
LB    LC  LD
 1     2   1
 2     3   3
 3     1   2
 4     4   4
```

References

[1] S. Carmody, M. Leeming and R.F.C. Walters, The Todd-Coxeter Procedure and left Kan extensions, *Journal of Symbolic Computation* 19(1995), 459-488.

[2] R.F.C. Walters, *Categories and Computer Science*, Cambridge University Press, 1991.