User Manual of LambdaGraph

The graphical interface of the functional
synchronous data flow language Lambda Flow
Version 0.1 (alpha)
26 July 1996
Number 95-33

by G. de WAILLY

1 What is LambdaGraph?

LambdaGraph (LG) is the graphical interface for the *LambdaFlow* (LF) language. Its allows a graphical programming of functional synchronous data flow (FSDF) for a parallel implementation. Its semantics is well defined with the *LambdaMatrices* (LM) abstract language.

LG is written in STk, the Scheme TK package written by Erik Gallesio, and available at kaolin.unice.fr:/pub. The used version of STk is 1.7.

This text describe how to install LG and it is an user manual.

I3S - 41 bd Napoleon III. 06100, Nice - FRANCE G. de Wailly: gdw@unice.fr

2 Initializing LambdaGraph

2.1 Requirement to run LambdaGraph

LambdaGraph require the interpreter `STk` written by Erik Gallesio. This package is available at '`kaolin.unice.fr:/pub/STk-2.1.7.tar.gz`'. It is easy to install. This package can be executed on

- Sparc (SunOs 4.1.x & Solaris 2.34)
- Dec 5xxx (Ultrix 4.2)
- SGI (IRIX 4.05, 5.1.1, 5.2)
- DEC Alpha 3000/400 (OSF-1 V1.3)
- RS6000 AIX 3.2.5
- HP 9000/735 (HP-UX 9.01)
- PC (Linux 1.0 -> 1.2)
- PC (FreeBSD 1.1)
- PC (SCO)
- PC (NetBSD-1.0)
- Sony WS (Sony NEWS, NEWSOS 4.2R)

On PCs, I think it is unreasonable to use LambdaGraph with less than 12Mb of memory (but possible, the author write a large part of the program with 8Mb).

2.2 Installing LambdaGraph

In order to install LambdaGraph, copy the file '`lg-01.tgz`' in the place you want to install the directory that will contain LambdaGraph (for example `cp lg-01.tgz $HOME/usr`).

Then untargz the file with the command `tar xvfz lg-01.tgz`. This command creates the directory tree of LambdaGraph.

Edit the file '`$HOME/usr/.graphe.stk`' and set the value of *root-path* to '`$HOME/usr/graph-0.1`'. The line will become: `(define root-path $HOME/usr/graph-0.1)`.

Add the directory '`$HOME/usr/graph-0.1`' to your *$PATH*: Edit the file '`$HOME/.profile`' and add the line `export PATH="$HOME/usr/graph-0.1:$PATH"` at the end of the file. (Perhaps you have to change `export` to `setenv`, according to your system shell).

Then type `graph`, and the marvelous world of LambdaGraph become a true reality for you (ie: Now the problems begin).

2.3 Getting started

2.4 Configuring

LambdaGraph supports some static options read at start-time. Some of these options concern the used color. Some, more important concern the paths of the different parts of the program..

```
;; .graph.stk
(define background-color         "PeachPuff")
(define grid-color               "gray")
(define actor-color              "White")
(define select-color             "IndianRed")
(define segment-color            "SaddleBrown")
(define handle-color             "blue")
(define segment-color-actif      "red")
(define input-color              "green")
(define output-color             "blue")
(define active-reliable-color    "red")
(define menu-color               "RosyBrown")
(define scroll-color             "RosyBrown")
(define command-background-color "PeachPuff")
(define command-text-color       "Black")
(define selected-command-color   "Red")
(define background-canvas-color  "darkgray")
(define canvas-color             "PeachPuff")
(define write-reliable-name      #t)
(define root-path                "/user/gdw/thesis/programs/graph"
(define library-path             ($ root-path "lib/"))
(define algebra-path             (& root-path "algebra/"))
(define compiler-path            (& root-path "compiler/"))
(define info-command (format #f "env INFOPATH=~adocs xjed -f info -f onewindow"
     root-path))
(define default-font
        "-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1")
```

3 Graphical interface layout

The graphical interface of LambdaGraph is built with three different parts: A menu, a command panel and a canvas. The menu gather commands that cannot be interface easily with the mouse or that are not be used frequently. The command panel allows actors (object of a program) to be created into the canvas. The canvas is the area where program are drawn by the user.

3.1 Menus

The menu allows some particular command that are difficult to handle with the mouse or that are not frequently used. The file menu entry deals with file management such as opening, saving, print... The Edit menu entry interfaces the clipboard operation such as copy/cut/paste. The Compile menu entry deals with the tool kit that can compiles/runs programs. The Option menu entry configures the editor and the help menu entry is supposed to help you.

3.1.1 File menu entry

The file menu entry deals with file operations such as open/save.

- *New* opens a new empty editor. The default name of the edited program is `unnamed.lg`. The editor will ask an other name on the first save.
- *Open:* opens a LambdaGraph file with a dialog-box. This dialog-box allows the user to open a file everywhere in the file-system. A `Double-<1>` in the file list will select the corresponding file to be open, as the button `OK`. The current directory can be changed with the use of the right directory list. A press on the `ALL-FILE` shows all the files in the current directory, such as directories, hidden file, ...
- *Save:* Saves the current file. If the name of the current file is '`unnamed.lg`', this action asks you a new file name before saves it (see `Save as`. If the user has opened a program of a library and modifies it, LambdaGraph update all the corresponding module of all the opened editors on saving. For example, if an editor uses the module `nand` of the library `logic` and if the user opens the corresponding file found in the library path, alters and saves it, LambdaGraph will updates the module used in the first editor, according to the modifications (perhaps, there will have no visible changes).
- *Save as:* saves the current edited program with a new name. Asks the new name with a dialog-box such as the one described in the `File.open` menu entry.
- *Save in library:* saves the current file in one of the libraries of the *library-path* option. The corresponding library will be updated with adding the new module of its list of modules.
- *Print:* prints the content of the canvas with the command `lpr postscript file`.
- *Postscript:* saves the content of the canvas in a postscript file. The name of the postscript file is asked with a dialog-box.
- *Close:* closes the editor. If there is no more editor opened, this command terminated the program and returns to the shell. *This command does not check if the current edited program is saved before close the editor.*
- *Exit:* closes all the opened editors and returns to the shell. *This command does not check if the current edited programs are saved before close the editors.*

3.1.2 Edit menu entry

The following commands use the clipboard (see Section 3.5 [clipboard], page 6).

- *Copy:* copies selected actors and handles with their inner links into the clipboard.
- *Paste:* copies the content of the clipboard in the canvas.

- *Cut:* does the same thing than the copy command, but also removes selected actors and handles from the canvas.
- *Select all:* selects all actors and handles in the canvas. This can be done also with `<2>` in the canvas.
- *Unselect all:* unselects all the selected actors and handle in the canvas. This can done also with `<3>` in the canvas.
- *Delete all:* deletes all the objects in the canvas. Deleted objects are copied in the clipboard.

3.1.3 Compile menu entry

LambdaGraph cannot directly executes drawn programs. It uses a compiler library to do that. A *compiler* is register into LambdaGraph at init-time (see Chapter 11 [compiler], page iii). When the user first activates one of these menu entries, a dialog-box asks him what compiler he wants to uses.

- *Compile:* compiles the current edited program with the current compiler. An error window may be appear in order to inform the user about inconsistencies of its program.
- *Run:* executes the current edited program with the currently selected compiler.

3.1.4 Option menu entry

- *Geometry:* change the size of the canvas of the editor. A dialog-box with two sliders is drawn in the screen. Moving the slider button dynamically changes the size of the canvas. If the `OK` button is pressed, then the changes is kept. If the `CANCEL` button is pressed, the old value is reactivated.
- *Grid:* draws a dialog-box that can change the grid aspect of the canvas. Chose the value of the space between lines of the grid by moving the button of the slider. Then `APPLY` applies this value to the grid without hide the dialog-box, and `OK` applies the change and hide the dialog-box. `UNGRID` hides the grid of the canvas. `CANCEL` restores the old mode.
- *Reliable:* show a dialog box that can changes the diameter of the reliable (inputs and outputs of actors, and handles). Big diameter is suitable for easy editing and small diameter is suitable for printing. Click with `<1>` on the slider button in order to chose the diameter. Then press `APPLY` to change the diameter of the reliable in the canvas without hide the dialog-box, or press `OK` to change the diameter and hide the dialog-box. Pressing `CANCEL` restores the old value.

3.1.5 Help menu entry

- *Info:* launches the info file viewer defined by the variable *info-command* of the file '`.graph.stk`'. By default the command is `"env INFOPATH=docs xjed -f info -f onewindow"`.

3.2 Command panel

The command panel is in the left of the editor window. It is composed by some buttons and it allows easy creating of actors in the canvas. Clicking `<1>` on one of the button select the inserting mode. The button becomes "sunken" and the text of the button is rewritten with the color red. Clicking `<1>` in the canvas (see Section 3.3 [canvas], page 5) creates the corresponding actors (see Chapter 4 [actor], page 6) at the position where the mouse cursor is. Click `<1>` in another button of the command panel unselected the previously selected button if it exists and select the pressed one. If the same button is twice pressed, no-mode is selected and a click `<1>` action on the canvas does not have any effects.

3.3 Canvas for drawing applications

The canvas is an area where data flow applications can be graphically edited. It is in the middle of the editor, with two scrollbar, one at its right and one at its bottom. The size of the canvas does not depend of the size of the editor window: if the editor window takes the whole screen, the canvas area appears into a gray inactive rectangle. The size of the canvas can be dynamically changed when big programs are edited (see Section 3.1.4 [option], page 4).

If a part of the canvas is hidden by the editor window border, the scrollbar become active.

When an insertion mode is selected in the command panel (see Section 3.2 [panel], page 5), the `<1>` event in the canvas creates the corresponding actor (see Section 4.2 [create-actor], page 7).

A `<2>` click selects all the objects of the canvas (see Section 4.6 [select-actor], page 9, see Section 6.5 [select-handle], page 11). A `<3>` event unselected all the selected object of the canvas.

3.4 Mouse event notation

The mouse is plainly used in LambdaGraph. It can be used to select a menu entry (see Section 3.1 [menu], page 3), an insertion mode in the command panel (see Section 3.2 [panel], page 5). It also used for more specific action in the program such as move, configure and destroy object.

Mouse event are noted `<1>` for a click with the left button of the mouse, `<2>` for a click with the middle button and `<3>` for a click with the right button. Specialization mouse event are noted `Shift-<1>` for a click with the left button with the `Shift` key pressed, or `Double-<1>` for a double click with the left button.

Generally, the left button `<1>` is used for direct action such as creation of actor, the middle button `<2>` is used for configuration such as the configuration of actors and the right button `<3>` is used for destruction such as the destruction of actors.

3.5 Clipboard operations

A clipboard is an invisible area where some object can be copied. This area is shared with all the editors. The clipboard is very useful for repeat a piece of program in the same editor, or to exchange a piece of program between two editors. It also can be used to temporary keep a deleted piece of program in order to retrieve it later if necessary.

Selected objects (see Section 4.6 [select-actor], page 9, see Section 6.5 [select-handle], page 11)can be copied into the clipboard with a *copy* command, a *cut* command, a *select all* command or with a *delete all* command (see Section 3.1.2 [edit], page 3).

The content of the clipboard can be copied into the canvas with the *paste* command (see Section 3.1.2 [edit], page 3). Note that the pasted objects become the selected in the canvas.

4 Actors

An actor is an element of the data flow language. Each actor has its own graphical representation, according to its type. Actors are drawn in a module, that is itself an actor. Actors have several inputs and several output (in fact, only modules can have several outputs, the other actors only have one output).

4.1 Differents type of actors

- *Atoms* are the basic expressions of the language. Natural integers and their associated operators, user's defined data and their specific operators, symbols, some comparators are atoms. The user can specify its built upon his data and operators. The syntax of atoms is the usual one.
- *Alternative* is a choice between two expressions, *then* and *else* depending on a *condition*. The evaluation result of an alternative is the evaluation result of either *then* if the evaluation result of *cond* is not zero or *else* otherwise.

  Because of the functional feature of the model, side-effects are impossible in an evaluation. So, it is not specified if both the blocs *then* and *else* are computed, or if only one of them is selected by the result of the evaluation of *cond*.
- *Application* acts as a filters of its arguments according to its operator semantics. The application evaluation returns the reduction result of the operator application to its evaluated arguments.
- *Stream* allows to write in a functional way a recurrent equation. This equation can be seen in different ways: In automatic engineering, it is a *delay* operator and in software engineering, it is a *state variable*. A stream has two parts: *State* which contains the initial value and *contract* for computing its next values by evaluation. Note that this streams semantics differs with the semantic of the streams of *Lucid*, *Lustre*.
- *Module* is a structured object that could gather other objects.

Actors are created with the command panel (see Section 3.2 [panel], page 5). Atoms and Applications depend on the current used algebra (see Chapter 8 [algebra], page 11). Their value can be changed with the configuration command (see Section 4.4 [configure-actor], page 7). Modules depend on the current used library (see Chapter 9 [library], page 12). They can also be configured (see Section 4.4 [configure-actor], page 7). In addition, a module can be edited (see Section 3.1 [menu], page 3)

4.2 Create an actor

An actor is created with the command panel (see Section 3.2 [panel], page 5). If a creation command is selected (the button text is red), a `<1>` event (see Section 3.4 [mouse], page 5) create the corresponding actor in place.

Then, the created actor can be moved (see Section 4.5 [move-actor], page 8), linked (see Chapter 5 [link], page 9), configured (see Section 4.4 [configure-actor], page 7) and destroyed (see Section 4.3 [destroy-actor], page 7). In addition, it can be selected (see Section 4.6 [select-actor], page 9) for clipboard operations (see Section 3.5 [clipboard], page 6).

4.3 Destroy an actor

Every actor in the canvas can be destroyed the a `<3>` (see Section 3.4 [mouse], page 5) event on it. All links (see Chapter 5 [link], page 9) which depend on this actor are destroyed too. Note that the destroyed actor is not saved into the clipboard (see Section 3.5 [clipboard], page 6), so the destruction is irreversible.

If actors are selected (see Section 4.6 [select-actor], page 9), they are destroyed too.

4.4 Configure an actor

Atoms, applications an d module can be configured. The first two are configured according to the selected algebra (see Chapter 8 [algebra], page 11). The last of the three is configured according to the selected library (see Chapter 9 [library], page 12). If there is not current algebra or library selected, the configuration command let the user first chose one.

4.4.1 Configure an input or an output of an actor

Inputs and outputs of actors can be configured with a `<2>` click one the wanted IO. This action has the same effect than the configuration of a handle See Section 6.3 [configure-handle], page 10.

4.4.2 Configure an atom

An atom is configured with a `<2>` event on it. The configuration of an atom draws a dialog box in the screen. Then, the user is waited to hit the value he want for this atom in the edit zone.

When the user hit the value, he press the `OK` button. Then, the entered value is checked according to the selected algebra check function. For example, with the *integer* algebra, the value `123` is correct and the value `123a` is incorrect.

If the value is correct, the atom is redrawn with the user value and the dialog box is hidden. If the value is incorrect, the dialog box is not hidden.

In addition, the user can press the `APPLY` button in order to redraw the atom without hide the dialog box, and he can press the `CANCEL` button in order to cancel the operation.

4.4.3 Configure an application

An application configured with a `<2>` event on it. The configuration action `<2>` on an application draws a dialog box. This dialog box contains the list of operator of the selected algebra (see Chapter 8 [algebra], page 11), the signature of the current selected operator and the associated comment.

The user can use the `<1>` event on the dialog box list of operator in order to select another operator and see its features. Note that a `Double-<1>` has the same effect than press on the `OK` button.

When the wanted operator is selected, the user can press the `OK` button. The application box is redrawn and the dialog box is hidden.

The user can also press the `APPLY` button in order to change the application box without hide the dialog box.

He can press the `CANCEL` button in order to cancel the operation.

He can choose another algebra with the `ALGEBRA` button. This action draw the algebra chooser dialog box.

4.4.4 Configure a module

An module is configured with a `<2>` event on it. The configuration action on an module draws a dialog box. This dialog box contains the list of module of the selected library (see Chapter 9 [library], page 12), the fan-in and the fan-out of the current selected module and the associated comment.

The user can use the `<1>` event on the dialog box list of module in order to select another module and see its features. Note that a `Double-<1>` has the same effect than press on the `OK` button.

When the wanted module is selected, the user can press the `OK` button. The module box is redrawn and the dialog box is hidden.

The user can also press the `APPLY` button in order to change the module box without hide the dialog box.

He can press the `CANCEL` button in order to cancel the operation.

He can choose another library with the `LIBRARY` button. This action draw the library chooser dialog box.

In addition, used module can be edited in a separate window. In order to edit a module, click on it with `Double-<1>`. You can also edit a module with opening its file (see Section 3.1.1 [file], page 3).

4.5 Move an actor

An actor can be moved on a `Press-<1>` event, without release the button, and with move the mouse. If actors (see Section 4.6 [select-actor], page 9) are selected, they are moved too. When an actor is moved, all its links are dynamically redrawn.

4.6 Select an actor

The `Shift-<1>` event toggle the selection of an actor. A selected actor appear in the selection color.

Selected actors can be moved (see Section 4.5 [move-actor], page 8), destroyed (see Section 4.3 [destroy-actor], page 7), copied into the clipboard (see Section 3.5 [clipboard], page 6).

5 Link

A link is a communication channel between either an output of an actor (see Chapter 4 [actor], page 6)or a handle (see Chapter 6 [handle], page 10) and an input of another actor. Handles, outputs and inputs are called *reliables*.

5.1 Create links

In order to enter in the link mode, first move the mouse cursor on a reliable (a reliable is an input, an output or a handle). If the reliable can be linked, its color change (by default, the linkable color is red). The only reliable that cannot be linked is an input already linked, because an input can has only one link.

Then click with `<1>` on the reliable named *start-reliable*. The mouse cursor may be changed, according the the nature of the start-reliable: On an input, the cursor becomes an arrow pointed on the top, otherwise it is an arrow pointed on the bottom.

At this stage, you can either chose another reliable named *target-reliable*, or cancel the link operation (see Section 5.3 [cancel-link], page 10). In order to chose the target-reliable, move the mouse cursor on it. Reliable that can be linked to the start-reliable has to change its color. If the wanted target-reliable does not change its color, it cannot be linked to the start-reliable. For example, if the start-reliable is an output, the target-reliable can only be an unlinked input reliable.

If the chosen target-reliable has changed its color, click on it with `<1>`. A line is drawn between the start-reliable and the target-reliable. This link can be destroyed (see Section 5.2 [destroy-link], page 9) or you can create a handle on it (see Chapter 6 [handle], page 10).

5.2 Destroy links

A link between two reliables can be destroyed with a click `<3>` on it. The selected link is destroyed, and all its relative link. For example, if an output is linked to an input via a handle (see Chapter 6 [handle], page 10), first consider the fact that the handle only has tow link, output-handle and handle-input. If you destroy the handle-input link, the output-handle and the handle-input link are destroyed, as the handle. If the handle has another link handle-input2, if you destroy the output-handle link, the links output-handle, handle-input and handle-output2 are destroyed as the handle. If you destroy the handle-input link, it is destroyed, but the handle and the other links are not altered.

5.3 Cancel a link operation

When you are in the link-mode (the mouse cursor is either an up-arrow or a down-arrow), you can click `<3>` on the canvas (see Section 3.3 [canvas], page 5) to cancel the link operation. The cursor is restored.

6 Handles

A handle is a deferred output: It is always linked to only one output, and it can be linked to at least one input or handles. Handles allows the user to reorganize its graph with beautiful links because handles can be move anywhere in the canvas.

6.1 Create an handle

An handle can only be created on a existent link: If you want a handle, first create a link and then create the handle. An handle is create with a click on a link with `<1>`.

An handle can be moved (see Section 6.4 [move-handle], page 10), configured (see Section 6.3 [configure-handle], page 10), selected (see Section 6.5 [select-handle], page 11) and destroyed (see Section 6.2 [destroy-handle], page 10).

6.2 Destroy a handle

A handle can be destroy with a `<3>` click on it. The handle is destroyed, and all its relative link. For example, if an output is linked to an input via a handle (see Chapter 6 [handle], page 10), first consider the fact that the handle only has tow link, output-handle and handle-input. If you destroy the handle-input link, the output-handle and the handle-input link are destroyed, as the handle. If the handle has another link handle-input2, if you destroy the output-handle link, the links output-handle, handle-input and handle-output2 are destroyed as the handle. If you destroy the handle-input link, it is destroyed, but the handle and the other links are not altered.

6.3 Configure a handle

A name can be associated to a Handle. In order to edit this name, `<2>` click on it. A dialog-box appears. You can enter the wanted name in the edition area. The `APPLY` button applies the name to the handle, without hides the dialog-box. The button `OK` applies the name to the handle and hides the dialog-box. The `CANCEL` button restores the original name of the handle and hide the dialog-box.

6.4 Move a handle

If the user `<1>` click on a handle without release the button and move the mouse, the handle moves. All the links attached to the handle move too.

Note that only the handle moves, even other objects are selected.

6.5 Select a handle

A handle can be selected with a `Shift-<1>` on it. Select a handle allows to move/destroy it when an actor is moved/destroyed.

7 Program

A program is the "thing" currently edited in the canvas of LambdaGraph. By default, programs are considered as top-level programs that use some actors and some modules of some libraries (see Chapter 9 [library], page 12): It cannot become a module used by another program. In order to allow the program to used as a module, it is needed to move it into a library. Then, it is a part of a library and it can be used into another program. A program can also bea module of library when the user explicitely open it (see Section 4.4.4 [configure-module], page 8)

8 Algebra

An algebra is a set of one data-type and some operators. It is used to configure atoms and applications (see Section 4.4 [configure-actor], page 7). Mixing algebra is allowed in the same module. Algebra are located in '`algebra`' directory of the *algebra-path* option variable of the configuration file '`.graph.stk`'.

```
;; comment of the algebra
(*comment* "Algebra for the logic computations")
```

The data-type is characterized by a full and a short name, and a check function. This check function has one string parameter. It returns `#t` if the string contains a valid data, `#f` otherwise. On the following type-definition, the boolean data-type is defined. Its long name is `boolean` and its short name is `b`. Validates string representing booleans are `"0"`, `"1"`, `"T"`, `"t"`, `"F"`, `"f"`. So the check-function compares its parameter to these strings and returns the result of this comparison:

```
;; boolean type definition
;;         name      short check
(*deftype* "boolean" "b"   (lambda (string)
                              (or (string=? string "0")
                                  (string=? string "F")
                                  (string=? string "f")
                                  (string=? string "1")
                                  (string=? string "T")
                                  (string=? string "t"))))
```

An algebra is also defined by a set of operators. An operator can configure applications of the program. It is featured with a name. This name will appears in the box of the applications drawn in the canvas. It also has a signature. This signature is written ala denotational `t1->t2->...->tm->tn`: the type of the returned value of the operator is `t1`, and the type of the parameters are `t1, ..., tm`. Such signature is used to know the arity of the operator. It can also be used to check in the program correct links (with type checking), but it is not yet implemented.

```
;; special operators and modules
;;   type   name  signature comment
(*operator* "and" "b->b->b" "logic and")
(*operator* "or"  "b->b->b" "logic or" )
(*operator* "not" "b->b"    "logic not")
```

9 Library

```
;;         comment   inputs   outputs geometry body
(*defmod* "nand"    '(e1 e2) '(s)    '175x205 '(

  ;; actors
  (define ad399 (*make* <InputPort>   :name "e2"  :x 133 :y 14 ))
  (define ad289 (*make* <OutputPort>  :name "s"   :x 82  :y 173))
  (define ad257 (*make* <InputPort>   :name "e1"  :x 38  :y 15 ))
  (define ad276 (*make* <Application> :name "not" :x 82  :y 115
                        :algebra "boolean"                    ))
  (define ad261 (*make* <Application> :name "and" :x 82  :y 62
                        :algebra "boolean"                    ))

  ;; links
  (define ad443 (*make* <Segment> :name ""
                                  :upstream (getIO ad399 0 <Output>)
                                  :backing  (getIO ad261 0 <Input> )))
  (define ad296 (*make* <Segment> :name ""
                                  :upstream (getIO ad276 0 <Output>)
                                  :backing  (getIO ad289 0 <Input> )))
  (define ad295 (*make* <Segment> :name ""
                                  :upstream (getIO ad261 0 <Output>)
                                  :backing  (getIO ad276 0 <Input> )))
  (define ad294 (*make* <Segment> :name ""
                                  :upstream (getIO ad257 0 <Output>)
                                  :backing  (getIO ad261 1 <Input> )))
  ))
```

10 Compiler

Compilers are function that can be called ...

```
;;            name
(*defcompiler "simulator"
              ;; compile command
              (lambda (inputfile)
                (system (format #f "stk -f check ~a" inputfile)))
              ;; run command
              (lambda (inputfile)
                (system (format #f "stk -f simul ~a" inputfile))))
```

Table of Contents

11 Index

(Index is nonexistent)