Vi-XFST

USER GUIDE

Version 1.2.0

by

YASİN YILMAZ

Sabancı University

Spring 2003

Contents

1 Introduction

This is a user guide documentation for Vi-XFST[1] project. This documentation introduces the software implementation of the XFST management model prepared by Prof. Kemal Oflazer[2] and Yasin Yılmaz[3] in Sabanci University. Please contact the designers of the project for the other related documents on the proposed model.

Intention of Vi-XFST project is to wrap the functionalities of XFST and to provide graphical editing, management and testing features to the researchers. XFST is a very powerful tool to manipulate finite-state networks, with its large set of commands. Vi-XFST is designed to make these commands invisible to the user and serve their functionalities through graphical components of its interface.

2 Features of Vi-XFST

Vi-XFST provides a simple and easy, yet powerful way to develop finite-state networks without involving developers in the complexities of a command line tool. With its set of innovative features, less experienced developers can quickly start testing with finite-state concepts seeing the actual picture on their workspace, while the advanced developers are freed from many manual tasks and controls that they had to cope with before. This means that they can focus on what to build, not on how to.

[1] Visual Interface for Xerox Finite-State toolkit

[2] oflazer@sabanciuniv.edu

[3] yyilmaz@su.sabanciuniv.edu

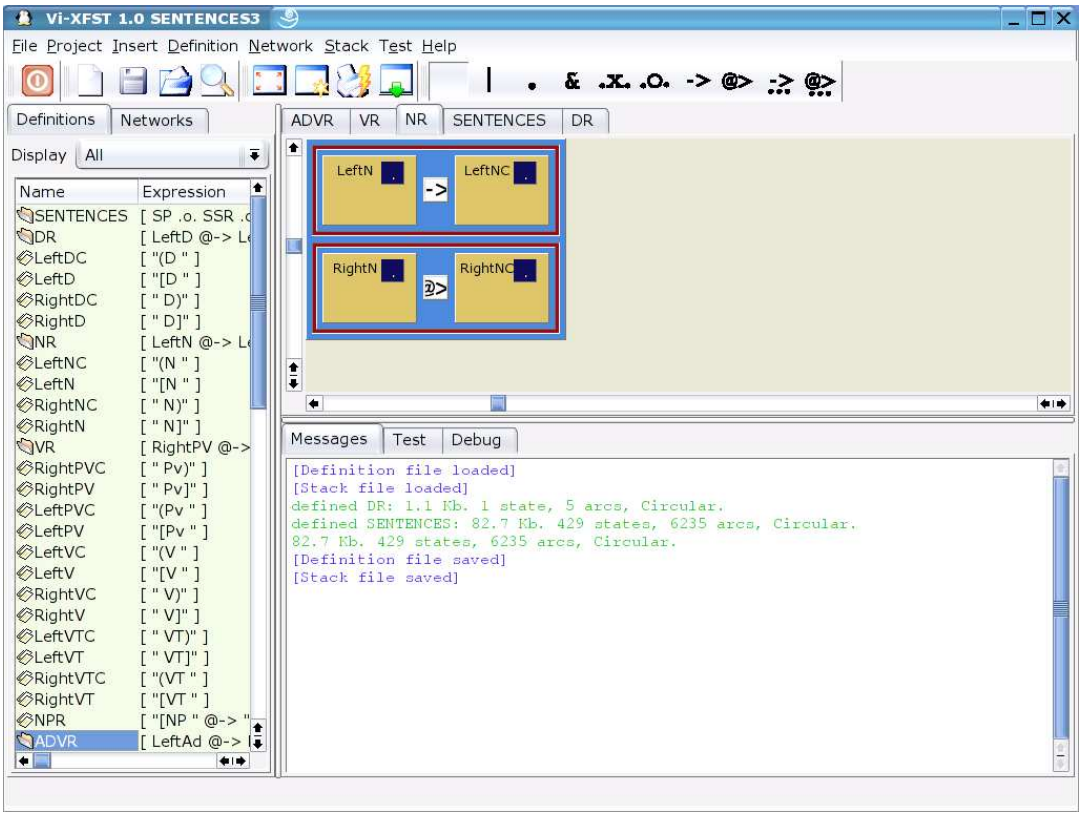

Figure 1: A sample screen-shot from the Vi-XFST IDE.

The following are important features of Vi-XFST:

- *A development environment designed for XFST:* Vi-XFST treats an XFST file as a development project that has to be managed on behalf of the user as he builds the regular expressions. The developer can move from the traditional way of editing an XFST source file, mostly done with a text editor like "*vi*" or "*emacs*", to a real integrated development environment, like Vi-XFST. He can see results of his work at design time, modify the code and retest any component.

- *Visual Regular Expression Development:* Vi-XFST's graphical regular expression construction tools allow developers to quickly build a visual model of their finite-state regular expressions. The developer quickly creates a topological model of the expression showing the relationship between expressions, as they are combined on the canvas of the visual editor. It is also easier to break down into the visual structure of large regular expressions. There is a hierarchical view of networks available, and that hierarchy is visible with in Vi-XFST. The user can zoom in a definition to see what is inside it, and go deeper, do tests at any level on a component, modify it, and go back to the top picture. This makes it possible to view networks at different levels of detail and make even large structures manageable and comprehensive.

- *Automatic definition and regular expression dependency checks and recompilation:* Vi-XFST watches modifications to a regular expression and recompiles any other definition that depends on it. Even the networks on the stack created with previous "*regex*" commands are recompiled if they depend on a modified regular expression. This is, in general, a difficult process for the developers to do manually. But with Vi-XFST, it is just transparent to the user at the background and automatically handled. Vi-XFST determines which definitions have to be recompiled. This selective recompilation of modified definitions is much efficient than recompiling the whole project.

- *A large set of supported XFST commands:* Beside expression operators, Vi-XFST supports many of the XFST's comprehensive command set and their options. They are hidden behind many easy to use dialog boxes, menu buttons and other graphical components of Vi-XFST. The developer will even use some of them without noticing, as he changes a project setting, clicks a button or updates an expression. Vi-XFST will send the appropriate commands to XFST on behalf of him to accomplish the requests.

- *Definition and Network Browsers:* These two browsers introduced in Vi-XFST display the list of defined regular expression definitions and available networks on the stack. The developer can access any one of the definitions or networks with just a mouse click. He can check or modify their properties, or use them in other parts of the project. For example, it is very easy to view which regular expression depends on a particular one. Without Vi-XFST, it is a quite difficult task. Also, properties of a network can be accessed with only a mouse click.

- *Drag and Drop:* Once a regular expression definition is defined, it is available in the Definition Browser. Then the user can drag and drop it with his mouse onto canvas to construct new expressions. Once basic definitions are defined, user can build new expressions without typing anything at all; create a definition base, drop previous definitions into it, and click *Push definition* and new definition is ready. Even a unique definition name is automatically created on behalf of the user. Vi-XFST provides a strongly typed development environment that reduces type errors while writing definition regular expressions.

- *History of input and output test strings:* Vi-XFST keeps track of strings applied to a network on the stack. User can always go back and test with his previous inputs with just one mouse click. He does not have to try to remember the inputs of last tests. They are saved inside the project file, and can be exported to any text file.

- *Message handling:* Vi-XFST handles every message from XFST program. They are never lost between user commands as before. Error messages, test outputs, normal XFST messages are all differentiated by Vi-XFST, parsed and indicated to the user.

- *XFST compatibility:* Vi-XFST project file can be used directly inside XFST as a script file. There is no Vi-XFST specific code inside the source file of the project that may be rejected by XFST.

- *Multi-platform IDE:* Vi-XFST runs on many Unix systems (Sun Solaris and all Linux distributions) and even on Microsoft Windows platforms with the same functionality. It is a fast pure C++ application, not a slow interpreted code like Java or TCL.

3 Installation and Requirements

Vi-XFST code can be compiled on any UNIX or Microsoft Windows platforms where QT library (version >= 3.0.0) is available. The installation process may vary according to the platform, but it is quite straight forward if these given steps are followed.

A script is prepared that uses QT tool *qmake* to compile and install the source code. This is useful where *autoconf* and *automake* tools are not installed on the target platform. The only requirement is the QT library. Just type:

```
# sh ./compile.qmake.sh
```

to start compilation, in the source directory. The default prefix value points to the users home director. Please adjust it prior to compilation by editing the "*DESTDIR*" variable in the "*project*" file in the "*infinity20/infinity20*" directory. The output binary file is named "*vixfst*" and installed in the prefix directory with the necessary auxiliary files, such as documentation (in *$prefix/docs*) and images (in *$prefix/images*).

The other way to build the code on Unix systems is mostly for development purposes. The original code is organized using *KDevelop IDE* and project source contains *autoconf* and *automake* compatible compilation and installation Makefiles. The *compile.autoconf.sh* script is prepared to automate this installation procedure, which generates a binary with *debug options on*. The output file name is "***infinity20***", which is the code name of the project during development phase. The binary is created in *infinity20/infinity20* directory. It is strongly recommended to use this compilation if you intent to debug your code.

Just gunzip and un-tar the source packet and enter the *infinity20* directory, type:

```
# sh ./compile.autoconf.sh
```

to start compilation. The installation prefix is "*/usr/local/*" by default. To install binaries type *"make install"* after compilation successfully ends. There are other options for the auto-generated "*configure*" script created during the compilation. These options may require advanced knowledge of GNU *autoconf* and *automake* scripts; therefore it is mostly used for development purposes only.

If you wish to debug or scroll through the classes and see the structure of the project code, you should load the project file for *KDevelop* "*infinity.kdeproj*" in *infinity20* directory. *KDevelop* also handles the projects with the above *autoconf* generated makefiles.

On Microsoft Windows platforms, Vi-XFST project file under *"Infinity20_Win32"* directory, should be loaded using *Microsoft Visual C++ (version >= 6.0)*. Once the compilation is done, the image and flush_network file should be in the same directory as the output executable file.

As it is mentioned above, the only requirement of Vi-XFST source code is the QT library. There are freeware versions for Unix systems. For Microsoft Windows platforms a freeware license is not available. But it is possible to obtain academic licenses for educational purposes at lower prices.

Our project code is not tested yet on other Unix operating systems (like FreeBSD, OpenBSD or HP Unix). But the design and implementation of the project avoids depending on any system specific libraries, calls or functions that may reduce the portability of the code. So there should not be any trouble porting Vi-XFST code on other UNIX platforms.

Of course, to build your own projects with Vi-XFST, XFST executable should be available in the target platform. For more information about availability of XFST binaries for your platform, please refer to *http://www.xrce.xerox.com*

4 The Integrated Development Environment

When you start Vi-XFST, you are immediately placed within the integrated development environment. This main window provides all the tools you need to design, compile and test your finite-state networks.

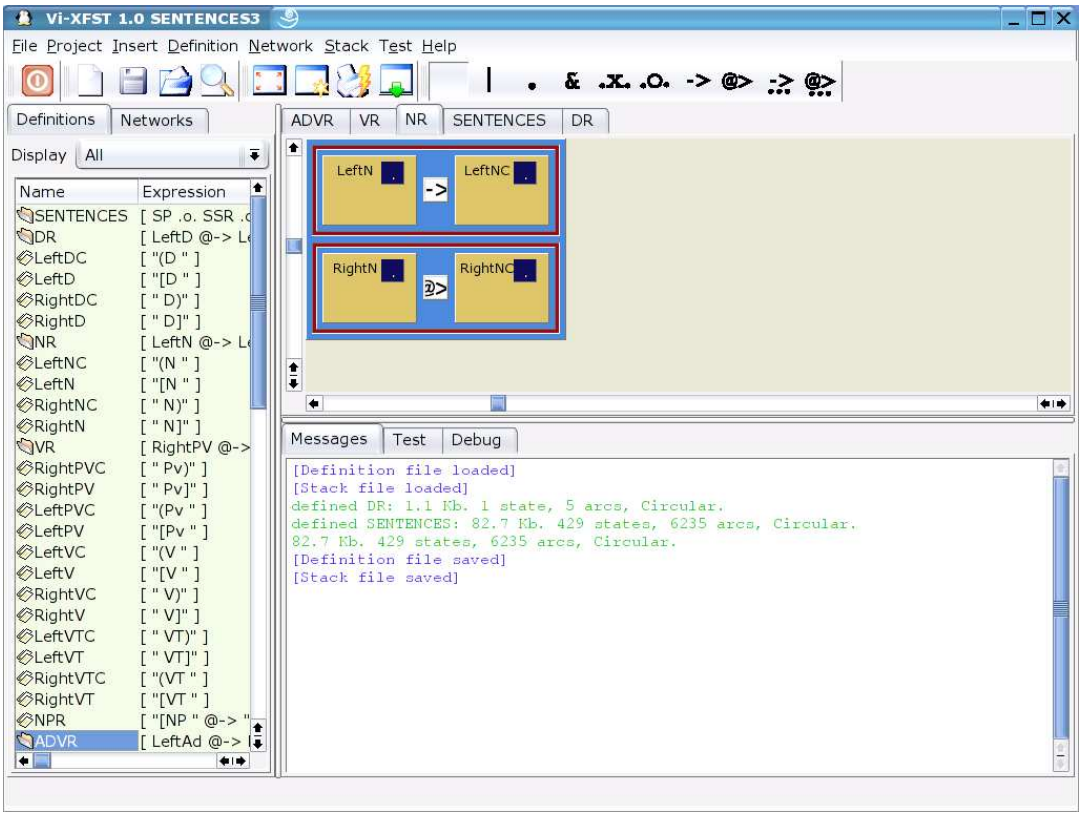

Figure 2: A sample screen-shot from the Vi-XFST IDE.

The development environment is composed of many graphical functional components; such as menu items, tabs, dialog boxes, which are used during different steps of finite-state project development. In the following sections these components are discussed in detail.

4.1 Main Window

Vi-XFST development environment main window is activated when the program is started. This window is the main control panel of the development process. For better understanding, main window can be detailed into the following components; *Definition Browser, Network Browser, Expression Canvas, Messages Tab, Test Tab Debug Tab and Menubar* components, each of which will be explained in more detail:

4.2 Definition Browser

The defined symbols in XFST that are constructed by "`define`" command are referred as regular expression definitions in Vi-XFST. Any regular expression created on Vi-XFST is added into the *Definition Browser*. It will remain there until it is undefined. *Definition Browser* is the main component to access a regular expression definition in Vi-XFST.

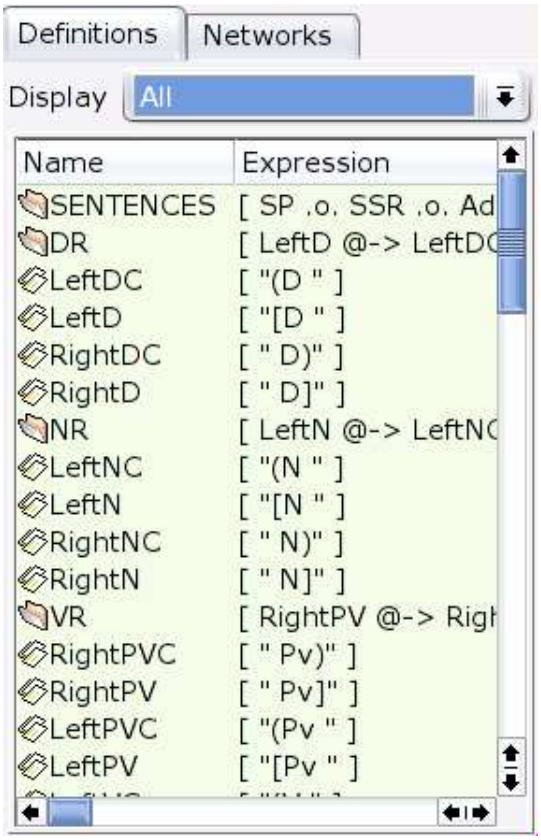

Figure 3: The Definition Browser

There are many functions available in the pop-up menu of *Definition Browser*. Select a definition item and right click to invoke the *Definition Browser* pop-up. The available functions are:

New Definition Invokes the *Definition Option* dialog to create a new definition. See Section 4.10 for more information.

Properties Displays the properties of a definition in *Definition Option* dialog. You can modify the definition properties in this dialog. See Section 4.10 for more information.

Read regex Creates the network for the definition and pushed it onto the stack. The *Network Browser* and *Test Tab* are activated after a successful compilation. The network appears in the *Network Browser* and becomes the top network. You can test it using the *Test Tab*.

Undefine Un-binds the symbol and removes the definition from the *Definition Browser*.

The items in the *Definition Browser* can be dragged and dropped into empty slots of an operator base on the *Expression Canvas*. See Section 5.2 for an example usage of drag and drop functions in Vi-XFST.

4.3 Network Browser

Every network on the XFST stack is listed in the *Network Browser*. The top network of the stack is the top item in the browser. Once a network is created, it is added to this browser where it will remain until it is popped up. *Network Browser* is the main component to access a network in Vi-XFST.

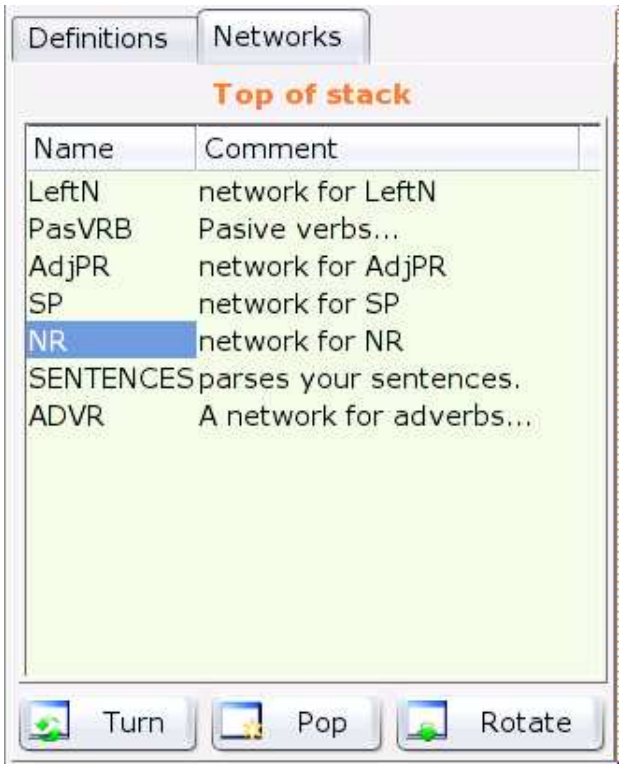

Figure 4: The Network Browser

To access the properties of a network, select and right click on it in the *Network Browser*. The *Network Options* dialog will be invoked in which various network settings can be adjusted. There are also other operations on the stack under **Stack** menu, which are *turn, rotate, pop, clear* and *print stack*. These operations modifies the stack in the XFST process, then Vi-XFST synchronizes its *Network Browser* with the XFST stack at the same time.

4.4 Expression Canvas and Workspace Tabs

Expression Canvas is where the visual regular expression development can be done when a project is active on Vi-XFST main window. There can be more than one *Expression Canvas* but only one of them is active at a time. These canvases are held in the *Workspace Tabs*. The user can switch between these tabs to activate the desired canvas and edit the regular expression inside it.

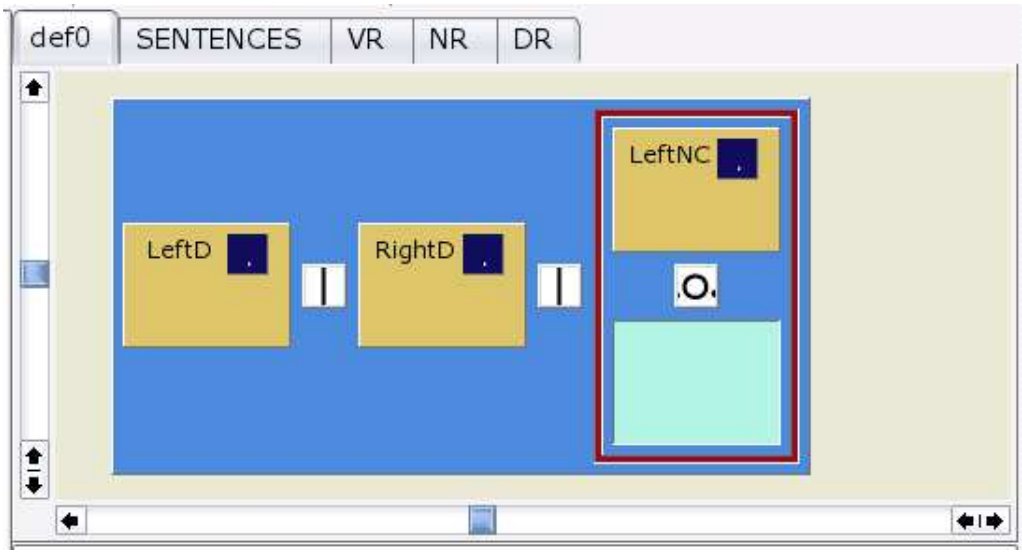

Figure 5: The Expression Canvas and Workspace Tabs

Each *Expression Canvas* can contain only one regular expression object. When this object is removed the associated canvas and workspace tab is also closed. To start a new regular expression in a new page, just select your operator from the tool bar and click on any *Expression Canvas*; Vi-XFST will open an empty workspace tab and canvas for you.

4.5 Message Tab

Vi-XFST also handles XFST messages on behalf of the user. They are filtered and displayed in the *Message Tab*.

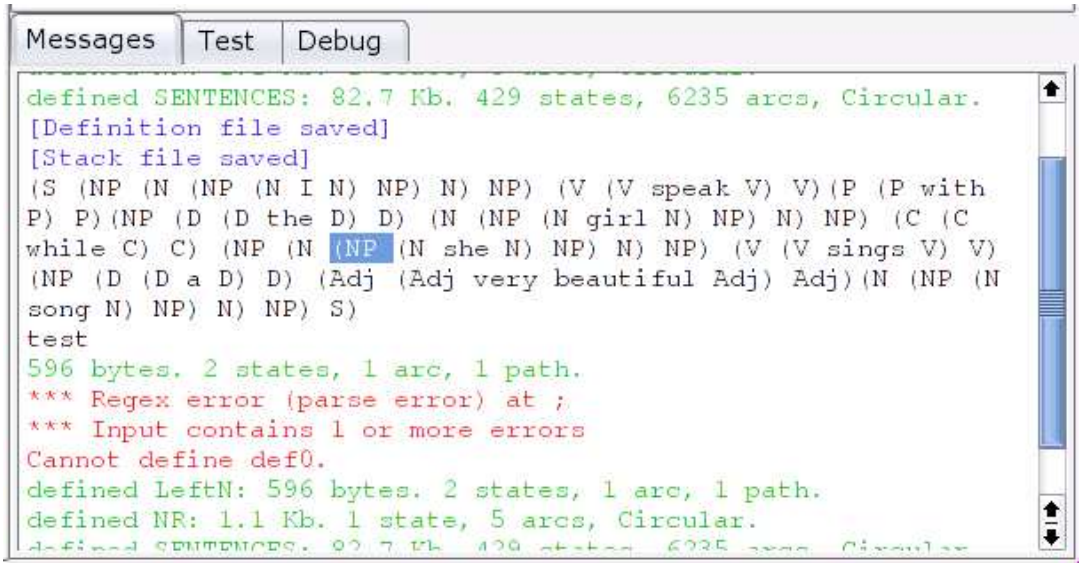

Figure 6: The Message Tab

The messages displayed in this tab can be grouped in to three:

XFST success messages They display successful command execution messages, such as a successful regular expression or network compilation. They are printed in green color.

XFST error messages They are printed in red, and inform error events. Most of the time, to take the attention of the user, a pop-up information dialog box may be invoked about the error.

Print Messages These are the outputs of *"print"* commands of XFST, such as *"print defined, print stack, print random-lower etc"*. They are displayed in blue.

Test Results Test result messages are generated by the "*apply*" commands. They are displayed in both *Test Tab* and *Message Tab*. These messages are printed in black fonts.

Vi-XFST Messages These messages are not XFST generated. Some of the actions of Vi-XFST produces these outputs, such as loading a project, saving or loading the stack file etc. These messages are in dark blue.

4.6 Test Tab

Test Tab is where you can apply strings to your networks on the stack.

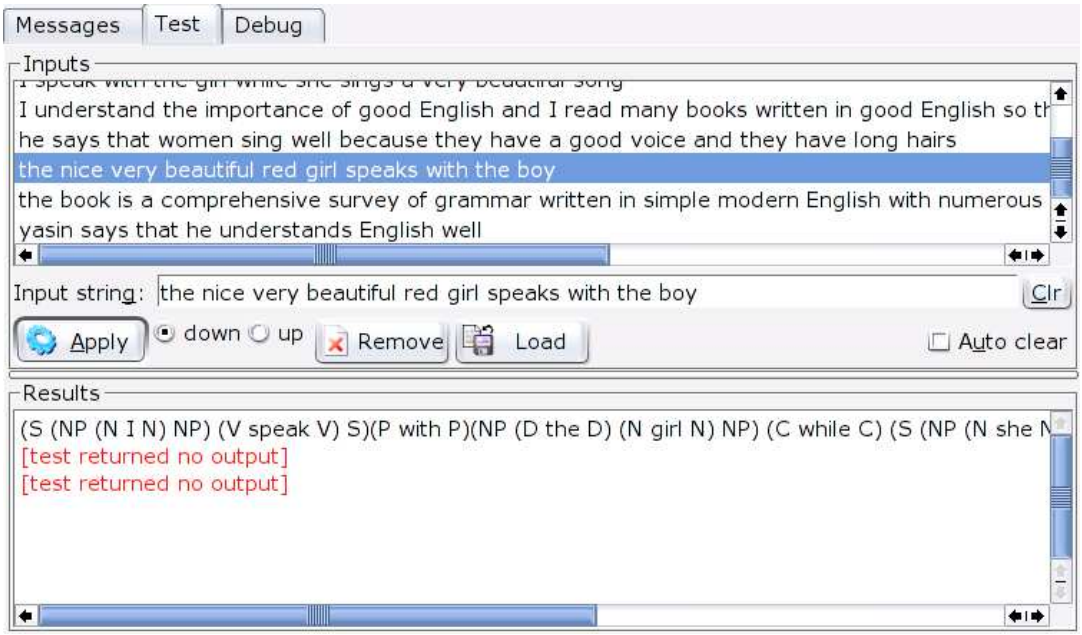

Figure 7: The Test Tab

When a network is compiled onto stack, Vi-XFST will automatically activate *Test Tab* and place the cursor in the *Input String Edit Box*. Enter your inputs into the *Input String Edit Box*, and press enter or click the *Apply* button just below. The direction of apply command can be set by *down* and *up* radio buttons near the *Apply* button. The results will be displayed in the *Results Edit Box*, and your input string will be added to the *Inputs* list. You can adjust

maximum number of input strings kept in the Inputs list and some other settings about *Test Tab* in the *Preferences* dialog.

Items in the Inputs lists can be removed, cleared all, loaded from, or saved to a text file by the buttons on this tab. These actions are also associated with menu items in the *Test* menu.

4.7 Debug Tab

The Vi-XFST project is still under development. Inevitably, despite our hard efforts on debugging the code, there may still be software bugs or logic errors. Therefore the *debugging window*, which is heavily used in the development process, is not removed from this release version. This window, when the message handler is installed and debug option is enabled during compilation, displays various debug messages from Vi-XFST execution flow.

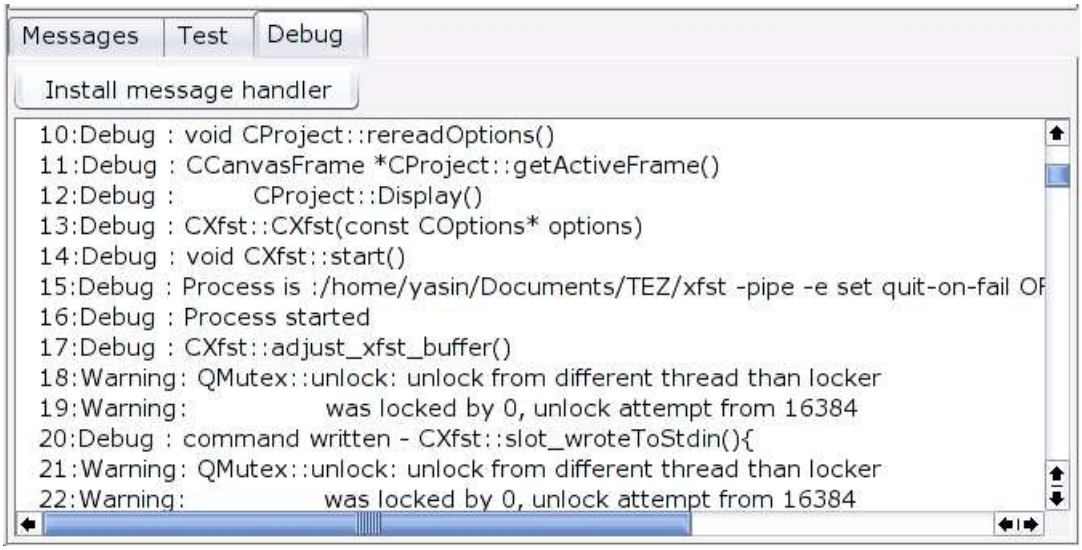

Figure 8: The debugging window is useful only when the debug option is set during compilation.

The user can include the outputs of this window to report bugs to the project developer that will be used to track down the cause of the problem.

4.8 Menubar Commands

File Menu

There are functions for editing Vi-XFST preferences and shutting down Vi-XFST, under file menu item.

Preferences Opens the *Preferences* dialog. See Section 4.12 on page 23 for more information.

Exit (Ctrl-Q) Initiates the shutdown sequence of Vi-XFST, which can be canceled if the user does not confirm the invoked confirmation dialog box. If there are modifications in the active project, the user will be prompted to save the changes. The Vi-XFST will close all open files, try to close and then try to kill XFST process, and save all Vi-XFST options and history to the operating system settings structure. For more information about Vi-XFST Settings refer to Section **??**.

Project Menu

New (Ctrl+N) Opens the *New Project* dialog, allowing to create a new project workspace and associated project file.

Save (Ctrl+S) Saves the active project file and associated binary definitions and network files to the project directory.

Load (Ctrl+L) Opens the *Load Project* dialog, allowing the user to select an existing project file to load into Vi-XFST.

Close (Ctrl+W) Closes the active project.

View & print project file (Ctrl+P) Opens the *Project Preview* dialog, where the project source file can be viewed, saved or printed in various formats.

Open recent project Contains a sub-menu with the last 10 opened projects. The user can open a project more easily using this recent project menu.

Save xfst messages Saves the *Message Tab* content to a text file. Invokes a *Save File* dialog.

Clear xfst messages Clears the contents of *Message Tab.*

Properties (F2) Opens the *Project Options* dialog that lets the user change various settings for the project. This could be a project name to save the project with another name or some user interface options.

Insert Menu

Release selection (F2) Click this menu option to de-select any selected insert toolbar button. This stops the pointer to insert new operators on the *Expression Canvas*.

Concatenation Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Concatenation operator.

Crossproduct Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Crossproduct operator.

Intersection Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place an Intersection operator.

Longest match replacement Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Longest Match Replacement operator.

Longest match markup Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Longest Match Markup operator.

Replacement Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Replacement operator.

Simple markup Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Simple markup operator.

Composition Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Composition operator.

Crossproduct Click this menu option and then click on the *Expression Canvas* or an open slot of an expression to place a Crossproduct operator.

Definition Menu

New definition (F4) Invokes the *Definition Options* dialog to get definition properties and then pushes a new definition into the stack with these values. The created definition will be added to the *Definition Browser*.

Push (F5) Defines the active definition on the *Expression Canvas* and adds it to the *Definition Browser*.

Read regex (F7) Compiles the active definition on the *Expression Canvas* adds it to the *Network Browser*.

Undefine (F8) Undefines the active definition on the *Expression Canvas* and removes it from the *Definition Browser*.

Properties (F9) Opens the *Definition Options* dialog box for the active definition on the *Expression Canvas*. Definition name, expression and comment are some of the editable values of a definition through this dialog box.

Save print outputs to file This is a toggle-menu item. If it is selected, the print command outputs will be written to a file instead of the *Message Tab*. This option invokes a *Save File* dialog.

Do not print to file This is a toggle-menu item. If it is selected, the print command outputs will be written to the *Message Tab* instead of a file.

Print-Defined prints each defined symbol and the size of the network it stands for.

Network Menu

Save print outputs to file This is a toggle-menu item. If it is selected, the print command outputs will be written to a file instead of the *Message Tab*.

Print first N outputs This is a toggle-menu item. If it is selected, the print command outputs will be written to the *Message Tab* instead of a file.

Print Random-lower Generates random words from the lower side of top network on the stack.

Print Random-upper Generates random words from the upper side of top network on the stack.

Print Words Prints the paths in the top network of the stack.

Print Lower Words Displays the words in the lower side language of the top network on the stack.

Print Upper Words Displays the words in the upper side language of the top network on the stack.

Print Net Prints a text representation of the top network on the stack.

Print Sigma Prints the sigma alphabet of the top network on the stack.

Print Random Words Generates random words from the top network on the stack.

Print Size Prints the size of the top network on the stack.

Tests Equivalent Returns 1 if the topmost two networks contain the same language.

Tests Lower-Bounded Returns 1 if the lower side of the top network has no epsilon cycles.

Tests Lower-Universal Returns 1 if the lower side of the top-level network represents the universal language.

Tests Overlap Returns 1 if the languages of the two topmost networks have a non-empty intersection.

Tests Sublanguage Returns 1 if the language of the topmost network is a sublanguage of the second network on the stack.

Tests Upper-Bounded Returns 1 if the upper side of the top level network contains no epsilon cycles.

Tests Upper-Universal Returns 1 if the upper side of the top-level network contains the universal language.

Prune Removes all paths leading to non-final states in the top network on the stack. This operation is not included in the project file.

Reverse Replaces the top network on the stack with the one that accepts the reverse language. This operation is not included in the project file.

Sigma Replaces the top network on the stack with a network that encodes the "sigma language" of the original network, that is, the union of all symbols in the sigma alphabet. This operation is not included in the project file.

Sort Reorders the arcs of the top network on the stack in a canonical order. This operation is not included in the project file.

Substring Replaces the top network on the stack with a network that accepts every string that is a substring of some string in the language of the original network. This operation is not included in the project file.

Optimize Runs a heuristic algorithm that tries to reduce the number of arcs. This operation is not included in the project file.

Unoptimize Reverses the effect of *optimize* command. This operation is not included in the project file.

Complete Extends the top network until it has a transition for every symbol in sigma in every state. This operation is not included in the project file.

Determinize Replaces the top network with an equivalent deterministic network. This operation is not included in the project file.

Epsilon-remove Replaces the top network with an equivalent one that has no epsilon transition. This operation is not included in the project file.

Invert Exchanges the two sides of the top network on the stack.

Lower side Extracts the lower language of the top network on the stack.

Minimize Replaces the top network with an equivalent one that has minimal number of states.

Negate Replaces the top network with a network that accepts all and only those strings rejected by the original.

Stack Menu

Clear Removes all networks on the stack.

Pop Removes the top network on the stack.

Print Displays the content of the stack.

Rotate Pushes the top element of the stack to the bottom.

Turn Reverses the order of the networks on the stack.

Test Menu

Clear outputs Clears the outputs generated by previous tests.

Save outputs Saves the output messages in the *Test Messages Window* into a text file. This option invokes a Save File dialog box.

Apply up Simulates the composition of the input string with the lower side of the top network on the stack and extracts the result from the upper side. Any output message is displayed in the *Test Messages Window*.

Apply down Simulates the composition of the input string with the upper side of the top network on the stack and extracts the result from the lower side. Any output message is displayed in the *Test Messages Window*.

Load text file as inputs Invokes a File Open dialog box, and loads the selected ASCII file as a list of input strings into the *Test Inputs Listbox*.

Save test inputs Saves the test inputs in the *Test Inputs Listbox* into a text file. This option invokes a Save File dialog box.

Clear test inputs Clears the entries in the *Test Inputs Listbox*.

Help Menu

About Vi-XFST Invokes the About Vi-XFST dialog, which gives the author contact information, version number, and some licensing information about Vi-XFST.

4.9 Project Options Dialog

Click **Project|Properties** or **Project|New** to invoke the *Project Options* dialog.

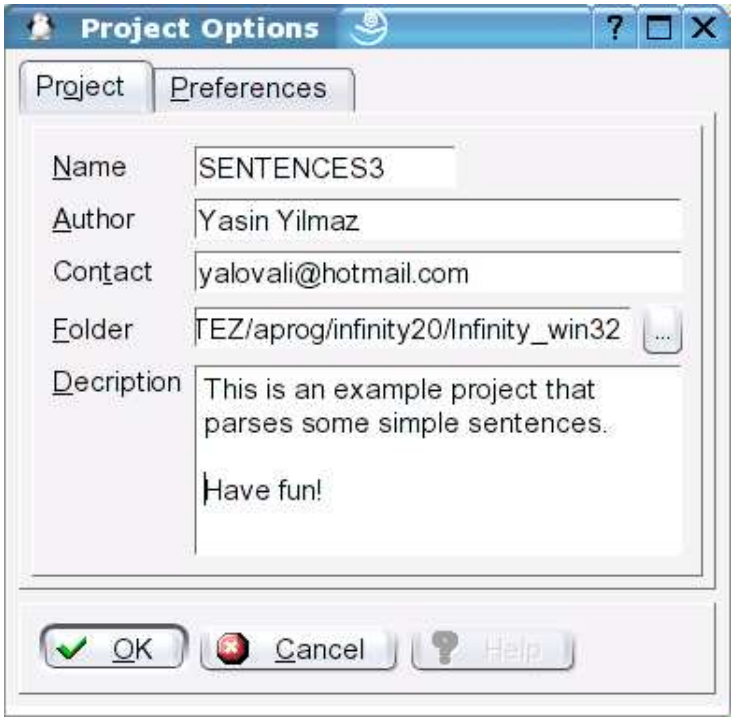

Figure 9: Project Options dialog

This dialog is used to edit properties a project session in Vi-XFST. The values that can be set in this dialog are:

Project Name This the descriptive name for the active project. It is also used to generate the filename of the source file with the extension "**.infproj**". Changing the project name, functions as a ”*save as*” operation. The next save command will create all project files according to this new name.

Author The author of the project file.

Contact The contact information for the project, probably an email address or a web site.

Folder The folder in which the project files will be saved. Default value is the current directory.

Description Intention of this field is a short description for the project purpose, structure or any other useful information to the others.

Canvas Color This option sets the *Expression Canvas* background color for this project.

4.10 Definition Options Dialog

Click **Definition|Properties** or **Definition|New** to invoke the *Definition Options* dialog.

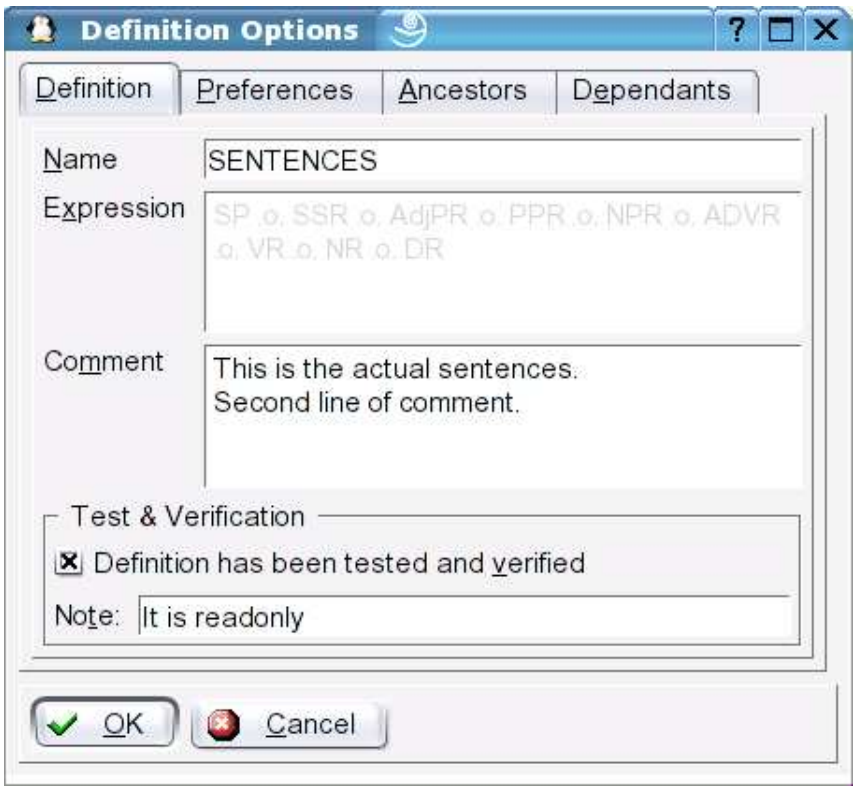

Figure 10: Definition Options dialog

This dialog is used to edit properties of an existing or a newly created definition. The values that can be set in this dialog are:

Name This is the name of the definition. If it is a new definition Vi-XFST will generate a random definition name for the user. In Vi-XFST definition names are kept unique and overriding a definition name is not allowed. If a definition has dependents, changing its name is also not allowed.

Expression It is the regular expression that this definition is composed of. Definitions in Vi-XFST can be created in two ways; either by typing an expression in this *Definition Options* dialog, or by using graphical components on the *Expression Canvas*. The first kind of definitions are marked as *non-visual*, they cannot be displayed on the *Expression Canvas* graphically. Their expressions can only be edited in this dialog. But the second type of definitions are called visual definitions, and their expression is extracted from their graphical representation on the canvas. These definitions can only be edited on the *Expression Canvas*. Therefore their expressions are read-only on in this dialog.

Comment Any comment on the definition can be typed in here.

Canvas Color This option sets the *definition rectangle* background color for this definition only.

Tested & Verified A checkbox to indicate that this definition is verified and tested, and possibly bug-free.

Verify Note A short note on verification of definition.

Ancestors A tree view of the ancestors of this definition. The root node is this definition and items below are the ones, which it depends on.

Dependents A tree view of definitions that depends on this definition. The root node is this definition and items below are the ones that depend on it.

4.11 Network Options Dialog

Click **Properties** menu item of the pop-up menu in the Network Browser to invoke the *Network Options* dialog.

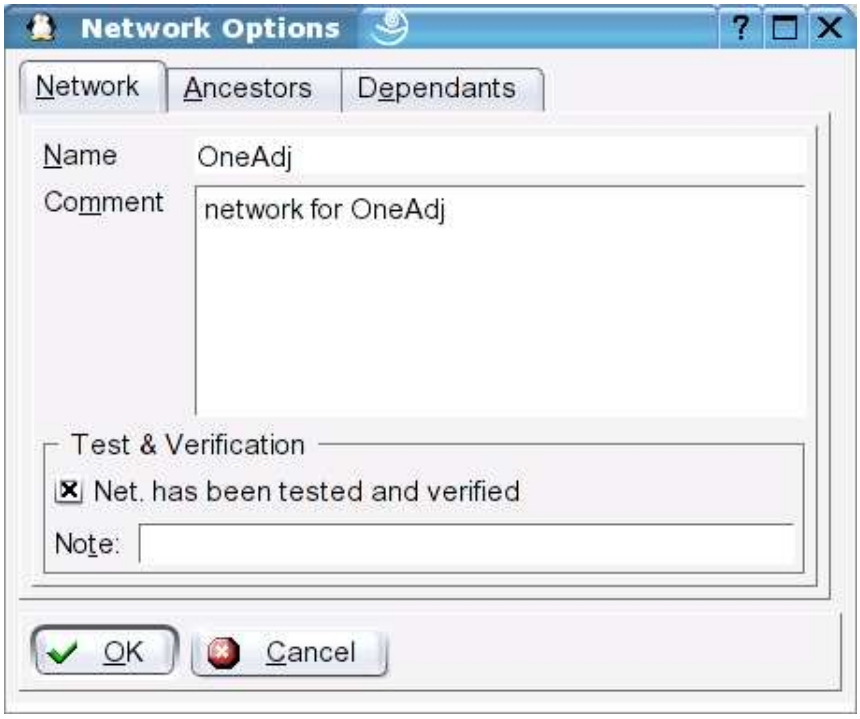

Figure 11: Network Options dialog

This dialog is used to edit and view properties of a network on the stack. The values that can be set in this dialog are:

Name This is the name of the network, which is the same as the definition that this network is compiled from. This value is read only.

Comment Any comment on the network can be typed in here.

Tested & Verified A checkbox to indicate that this network is verified and tested, and possibly bug-free.

Verify Note A short note on verification of this network.

Ancestors A tree view of the ancestors of this network, which is the same as ancestors of the definition of the network. The root node is this network and items below are the ones, which this network depends on.

dependents A tree view of definitions that depends on the definition of the network. The root node is the network and items below are the ones those depend on it.

4.12 Preferences Dialog

Click **File|Preferences** to invoke the *Preferences* dialog. The following dialog will appear, to let various Vi-XFST settings to be changed.

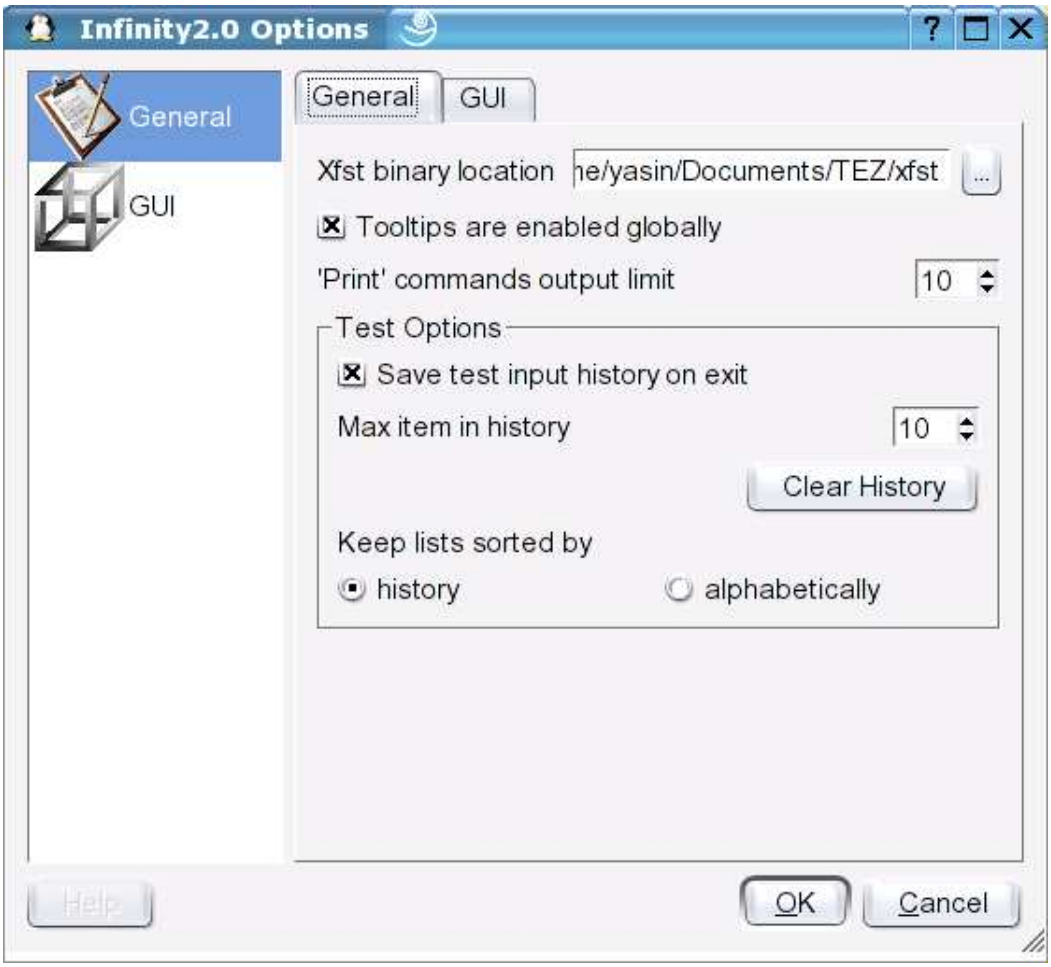

Figure 12: Preferences dialog

The setting in this dialog box are saved in *"$HOME/.qt/infinityrc"* file on UNIX systems. On Microsoft Windows operating systems it is in the registry database under *'infinityrc'* key. For the first execution of Vi-XFST, some default values will be set to these options. The most

important of them that has to be reset by the user is the XFST (or XFST.exe on Microsoft Windows systems) binary location. If this location is not entered or invalid, Vi-XFST will not be able to load XFST process.

The options that can be set in this dialog are:

XFST Binary Location This is the location of the executable XFST binary file on the system. User should have proper access rights to execute this file.

Enable Tooltips Enables/disables the tooltips available for many components on Vi-XFST. These are little help messages displayed in a small yellow box below the mouse cursor, that appears when the user points to a menu item, definition box on the canvas or toolbar items.

Print Commands Output Limit Sets a limit on the upper limit of output lines generated by print commands available in *definition* and *network* menus.

Save Test Inputs On Exit Enables/disables automatic saving of test inputs list into your project file.

Max Number of Inputs Puts an upper limit on the number of test inputs that will be kept in your project file.

Clear History This button clears the test input list.

Keep Inputs Sorted by History|Alphabetically Sorts the input list according to the given criteria.

Definition Canvas Color This option sets the *definition rectangle* background color for this definition.

Project Canvas Color This option sets the *Expression Canvas* background color for this project.

Font Options These are the font settings used in the canvas of the Vi-XFST. They can be changed to whatever settings are available on the underlying operating system.

4.13 Project Preview Dialog

Click **Project|View & Print** menu to invoke the *Project Preview* dialog. The following dialog will display the source code of your project.

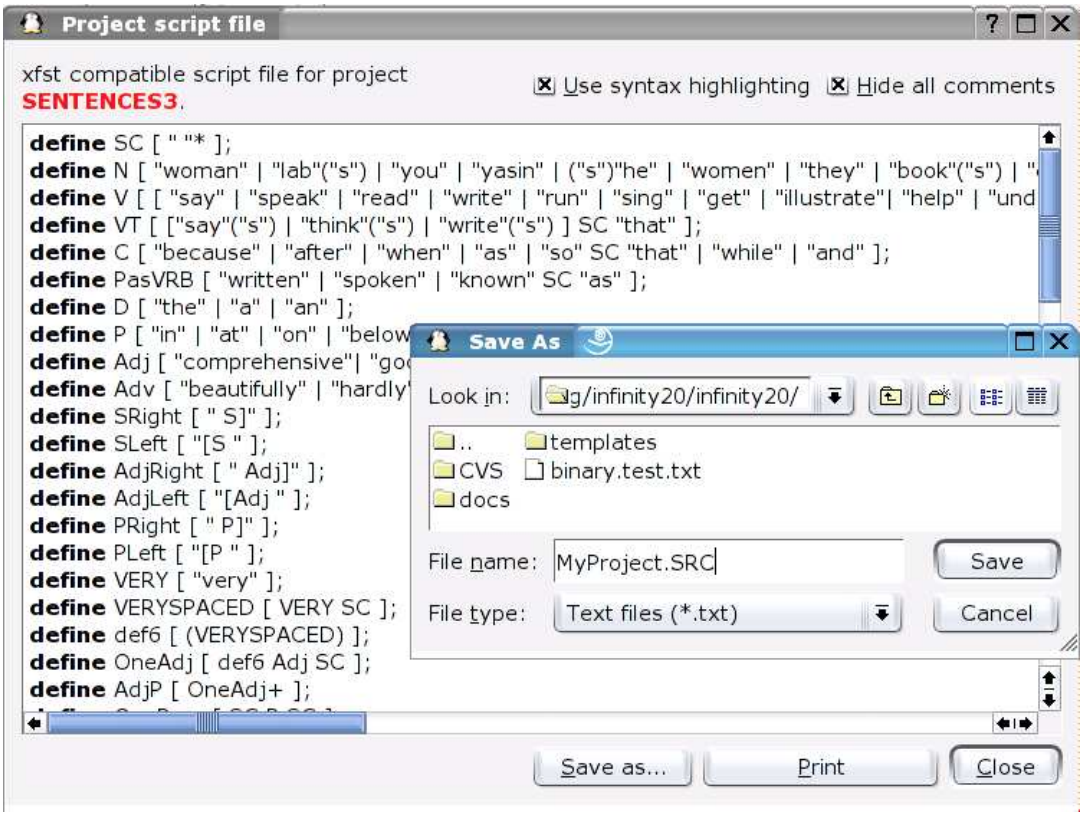

Figure 13: Project Preview dialog

You can use this dialog to export the project to a text file or print in various formats. Click *Hide all comments* checkbox to hide/unhide Vi-XFST in-line control comments. Or you can enable/disable syntax highlighting by the *Use syntax highlighting* checkbox. The **Print** button will call the system print dialog box and let you choose printing preferences and get a hardcopy of your project. If your underlying system permits, a postscript copy can also be generated with this printing dialog.

The **Save** button lets you export a copy of the project into a text file, according to the display criteria set in this dialog.

5 Project Development Process

5.1 Starting a New Project

Vi-XFST handles a development session with XFST in a *Project Workspace*. For each project workspace, a project file is created. Also when the project is saved, a binary definition and/or network file will be created in the same directory. Definition and network settings, regular expressions, test inputs and user comments are kept in the project file which has a name created by concatenation of project name and "*.infproj*" file extension.

To start a project workspace, click **Project|New** menu item, or the associated tool button. The *Project Options* dialog will be invoked. Enter a descriptive name for your project, and

select a project directory. The default value points to the current directory, but it is probable that you will want your project files saved in a more reasonable location.

You can also enter some values for *Author*, *Contact information* and for *project description*. These are optional fields and you can change these values at any time later, just select **Project|Properties** menu to bring this dialog back.

When you click the OK button, the *Project Options* dialog will be closed and a new project workspace will be initiated. A XFST process will be loaded and menu items, browsers and workspace canvas will be initialized. After the initializations, you can start adding definitions to your workspace.

5.2 Building Regular Expressions

There are two ways to define a regular expression definition in Vi-XFST. The quickest way is to type in a regular expression using the keyboard. Simply click **Definition|New Definition** menu item or associated keyboard shortcut (F4) to open *Definition Options* dialog.

In the dialog a definition name is already generated for you. Just type a regular expression, -some comment is optional but recommended- and click **OK**. The *XFST Progress Dialog* will appear and try to define your expression. If no error occurs, your regular expression definition will appear in the *Definition Browser*. The *Message Tab* will be popped up if it is not visible. Check these messages for your definition. If there has been an error, Vi-XFST would have noticed that and display the error message generated by XFST.

The other way to create your definition, is to use the *Expression Canvas* to build your regular expression in a more controlled and user friendly way. Vi-XFST offers a powerful visual interface for regular expression construction.

Select an *operator base* type from the toolbar and click on the *Expression Canvas*. Vi-XFST will draw the operator base with empty slots. These slots are where you will insert existing regular expression definitions. You can select and drag a definition from the *Definition Browser*. Or write click the empty slot and select **Insert Definition** menu item and select an existing definition from the list. The selected definition will be added into this empty slot you have right-clicked.

You can double click an empty slot to create a new definition more quickly. *Definition Options* dialog will be invoked and the created definition will be inserted into this slot automatically.

For more information about working with graphical representation of regular expression see Section 6.

5.3 Compiling a Regular Expression

Once a regular expression is defined in XFST and added to the *Definition Browser*, now it can be compiled as a network onto the stack. There are many ways of compiling a regular expression, you can just right click a definition in the *Definition Browser*, and select **read regex** in the invoked pop-up menu. Or if your regular expression definition is on the expression canvas, right click the definition there and select **read regex** menu item in the pop-up. There is also another menu item to do same task under the *Definition* menu.

During the regular expression definition compilation, *XFST Progress* dialog will be displayed. If there is no error, your network item will appear in the *Network Browser*. The *Test Window* will be popped up if it is not visible. If there has been an error, Vi-XFST would have noticed that display the *Message Window* instead of the test window.

Compiling a regular expression is just one mouse click as described above, and you can quickly start entering inputs to the network. Now, please proceed to the next section.

5.4 Testing a Network

To apply input strings on the transducer at the top of the stack, switch to the *Test Tab* if it is not already activated. Enter your inputs into the *Input String* edit box, and press enter or click the *Apply* button just below. The direction of apply command can be set by *down* and *up* radio buttons near the *Apply* button. The results will be displayed in the *Results* editbox, and your input string will be added to the *Inputs* list.

Items in the inputs lists can be removed, cleared and loaded from or saved to a text file. These operations are available both through the buttons on the *Test Tab* and menu items under the **Test** menu. If auto-save option is set in the Vi-XFST settings, input strings are kept inside the source file when the active project is saved. They are also loaded when the project is opened back.

5.5 Modifying The Stack

Items on the stack are the networks compiled by Vi-XFST. You can remove (pop-up), change position and ordering (turn, rotate) of these items with buttons below the *Network Browser*. There are various operations over a network on the stack. Most of them are available under the **Network** menu. For this version of Vi-XFST these network modification commands are not kept in your project file. But the binary network file saved by "`save stack`" command, will contain your most recent stack including these modifications. So keep in mind this issue and beware that Vi-XFST will not re-run these modification commands when it recompiles the networks on the stack.

5.6 Printing and Viewing the Source Code

The project source file can be viewed within the *Project Preview* dialog. Click **Project|View & Print** menu to invoke the dialog that will display the source code of your project.

You can use this dialog to export the project to a text file or print in various formats. Click *Hide all comments* checkbox to hide/unhide Vi-XFST inline control comments. Or you can enable/disable syntax highlighting by the *Use syntax highlighting* checkbox. The **Print** button will call the system print dialog box and let you choose the printing preferences and get a hardcopy of your project. If your underlying system permits, a postscript copy can also be generated from this printing dialog.

Click the **Save** button to export a copy of the project into a text file, according to the display criteria set in this dialog.

5.7 Exporting the Code and Binary Files

Under the project directory (see *Project Options* dialog), there are three files related to a project. These are:

<ProjectName>.infproj The source file for your project. It contains project information, options, network definitions and input strings. This file can be loaded into XFST with "-l" parameter. All the Vi-XFST generated codes are marked with "##Vi-XFST##" comment markers. But it is strongly advised not to edit this file manually. Instead, use the *Project View & Print* dialog described in Section 4.13 to generate a user copy of the project source file.

<ProjectName>.infdef This binary file is created by the XFST "`save defined <filename>`" command automatically by Vi-XFST whenever the active project is saved. The binary file contains networks for all defined symbols in the project workspace. You can use this file in XFST with "`load defined <filename>`" command. Vi-XFST will try to locate this file when the project is loaded, but if it is not available, all definitions will be rebuild from the regular expression source file. But it cannot detect if this file is modified outside Vi-XFST, therefore you should not change the content of this file. You must work on your own copy of this binary file.

<ProjectName>.infstack This binary file is created by the XFST "`save stack <filename>`" command automatically by Vi-XFST whenever the active project is saved. The binary file contains networks on the stack of the project workspace. You can use this file in XFST with "`load stack <filename>`" command. Vi-XFST will try to locate this file when the project is loaded, but if it is not available, all networks will be rebuild from the source file. But it cannot detect if this file is modified outside Vi-XFST, therefore you should not change the content of this file. You must work on your own copy of this binary file.

Any modification on the stack will be effective in this binary file. So if you want to prepare a binary transducer file to distribute without the source code, you can freely do any modification with the operators in Network menu. But remember that these modifications are not saved into project source file.

All of the files listed above, are compatible with XFST program. Any of them can be distributed to other users. But only the project file (with extension *.infproj*) can be loaded back to Vi-XFST.

If the project file seems confusing with many inline comment blocks put by Vi-XFST, you can get a tidier file by *Project Preview* dialog as described above.

5.8 Bug Reporting and Debugging Vi-XFST

The Vi-XFST project is still under development. It lacks many features of a comprehensive integrated development environment. Inevitably, despite our hard efforts on debugging the code, there may still be bugs, logic errors, or even crashes while using Vi-XFST. Therefore the debugging window, which is heavily used in the development process, is not removed from this release version. This window, when the message handler is installed and debugging enabled during compilation, displays various debug messages from Vi-XFST execution flow.

If you experience a bug to report to the authors, please send a copy of these messages that will let to track down the bug. We appreciate every bit of help to improve the code.

6 Graphical Representation Of a Regular Expression

One of the most powerful features of Vi-XFST is the graphical representation of regular expressions. On the *Expression Canvas*, it is possible to build complex regular expression with simple mouse clicks.

When a project is opened on Vi-XFST, there is always an active workspace tab that contains an empty expression canvas. You can use this canvas to place and construct expressions on. First step is to select an operator from the toolbar that will be the main operator base. Then operands should be added to slots of this base. Each slot on Vi-XFST's operator bases is a like a pair of brackets (`"[" ... "]"`) in regular expression text. Vi-XFST places your definitions in these boxes. Figure 14 shows an example definition and the corresponding expression.

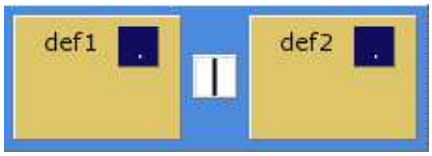

Figure 14: A definition base with two open slots: `[ def1 | def1 ]`

Operator bases like union, concatenation or composition may take more than default number of operands. To add additional slots select **New Slot** from the pop-up menu of the base.

Figure 15: A definition base with two open slots: `[ def0 def1 def4 ]`

Just the same way an empty slot may be removed. Right click an empty slot and select **Remove** from the invoked pop-up menu to remove it from the operator base.

There are three ways to insert a definition into an empty slot. If you want to create and add a new definition, just double click into an empty slot. Vi-XFST will create a definition for you and pop-up the *Definition Options* dialog. Freely enter any expression you like, there is no restriction in regular expressions for definitions created using this dialog. Click **OK** to accept your changes. Vi-XFST will automatically push your expression into XFST, add it to the *Definition Browser* and replace the empty slot with this definition. This is a very fast way to build up complex expressions.

Another way is to use an existing definition from the *Definition Browser*. Select it with mouse, drag and drop it into an empty slot. This is also another comfortable way of building expression within Vi-XFST.

The last way is to select the definition to be inserted, from the pop-up menu of the empty slot under **Insert Definition** item. Sub-menu of this item is the list of definition available in Vi-XFST. The selected definition will be inserted into the empty slot.

It is also possible to insert an operator base into an empty slot of another base. This feature enables to build complex regular expressions. It is important to remember that each slot is a pair bracket. Therefore your new operator base is enclosed within a pair of brackets as show in Figure 16.

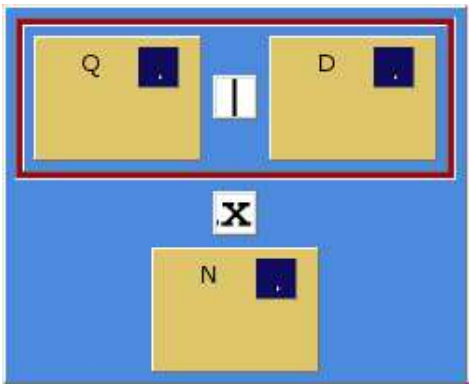

Figure 16: Nesting operator bases in each other: `[ [ Q | D ] .x. N ]`

There is no limit in the depth of nesting operators inside each other. You can freely build large regular expressions. If things get confusing on the canvas, try the **Minimize** option in the pop-up menus of operators to shrink the operator you are working on.

Once you are done with your regular expression, it has to be defined in XFST process. Select **Push definition** in the pop-up menu of the operator base. Vi-XFST will define and store it in the *Definition Browser*. For more information about defining a definition see Section 5.2.

6.1 Operator Base Object

The basic component of an expression is the base rectangle that defines a pair of brackets (¨[¨ ... ¨]¨) and the selected operator. You add other components (definitions) onto this base.

You can give it a name that will be used to refer to this network once it is compiled. When a base is added to the workspace canvas a unique definition name is already generated for you. Also the name of a base can be modified from the *Definition Dialog* that is invoked with the Properties item inside pop-up menu of the base. When this definition is compiled, it will appear in the definitions browser of Vi-XFST with this new name. The following drawing is a simple definition object:

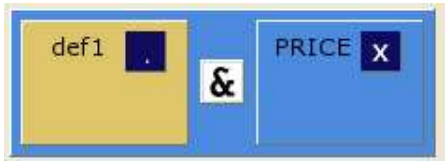

Figure 17: A sample operator base

In Figure 14 the main operator is an intersection. This base has two operands; `def1` and `PRICE`, which are also previously defined definitions. These definitions can be removed from

the base by selecting **Remove** operator from the pull-down menu invoked by right clicking on them.

The symbols on the upper right corner of these definition rectangles denote if the definition can be enlarged inside the operator base. A "." means that this definition was not built using the visual expression canvas; therefore it cannot be enlarged using graphical components on the screen. Also double clicking on the definition rectangle only opens this definition's properties.

The "x" sign on the upper right corner of a definition, as for the "PRICE" in Figure 14, means that the definition can be enlarged into its components. When a definition is enlarged by clicking on this icon, this symbol changes into an "O". Clicking again in this symbol shrinks the definition back to its original state. Here is an example of viewing a definition in enlarged form:

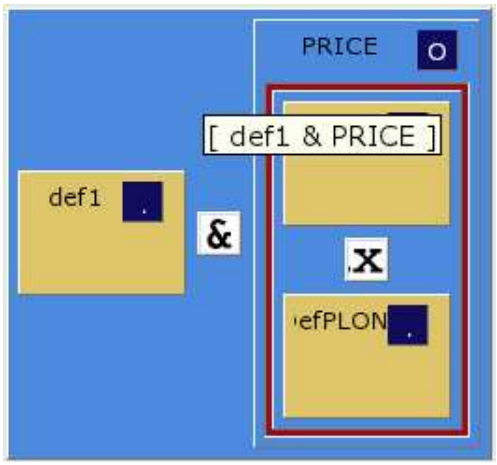

Figure 18: The PRICE definition is enlarged inside another definition.

By using this feature of Vi-XFST, it is possible to view a transducer back into its sub-components. It is also possible to compile an enlarged definition, and apply input string to this network on the stack to debug it.

7 Graphical Regular Expression Components

In this version of Vi-XFST, only most commonly used regular expression operators are supported on the expression canvas. Regular expressions that require the other operators can be still built using the *Definition Options* dialog that is invoked by **Definition|New** menu. We hope to release support for these excluded operators in the next version.

Once an operation is defined you can still change it, replace operands, or delete it later. It is possible to add new slots to an operator if it can take more than default number of operands.

For example the default Union operator comes with two empty slots. But you can always add new slots for additional operands. Also some operators, such as markup and replacement, have "*conditional*" operator bases that have special meanings for them. You can add a conditional base to them from the pull down menu by right clicking on these operators. All of these features are accessible through the pop-up menus of the operator slots.

The following sections in this chapter are a list of available operators in Vi-XFST.

7.1 Union

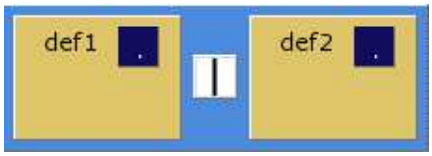

Figure 19: `Union` operator base. Displayed regular expression:
`[ def1 | def2 ]`

Opens with 2 default open slots and more slots can be added. Operator icon can be changed into: Concatenation, Intersection.

7.2 Concatenation

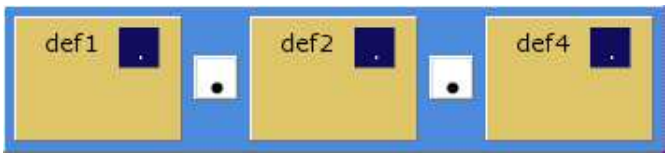

Figure 20: `Concatenation` operator base. Displayed regular expression:
`[ def1 def2 def4 ]`

Opens with 2 default open slots and more slots can be added. Operator icon can be changed into: Union, Intersection.

7.3 Intersection

Figure 21: `Intersection` operator base. Displayed regular expression:
`[ [def13 | def8 ] & def2 ]`

Opens with 2 default open slots and more slots can be added. Operator icon can be changed into: Union, Concatenation.

7.4 Composition

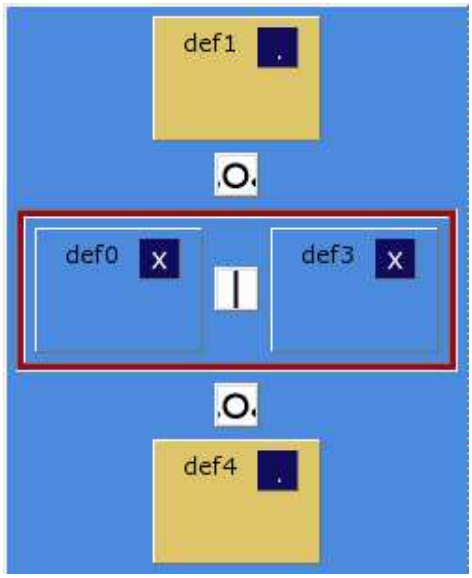

Figure 22: `Composition` operator base. Displayed regular expression:
`[ def1 .o. [ def0 | def3 ] .o. def4 ]`

Opens with 2 default open slots and more slots can be added. Operator icon cannot be changed into another operator.

7.5 Crossproduct

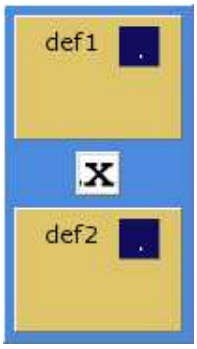

Figure 23: `Crossproduct` operator base. Displayed regular expression:
`[ def1 .x. def2 ]`

Opens with 2 default open slots and no more slots can be added. Operator icon cannot be changed into another operator.

7.6 Replacement

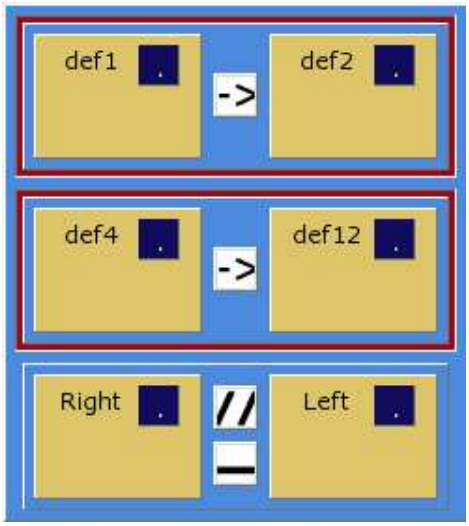

Figure 24: `Replacement` operator base. Displayed regular expression:
`[[ def1 -> def2 ],[def4 -> def12 ] // Right _ Left ]`

Opens with 2 default open slots. Possible to add another pair of slots for parallel replacement. Operator icon can be changed into: Longest-match Replacement.

Special option: *New Condition* adds a condition with two open slots to the replacement operation. Only one condition can be defined per operator base.

7.7 Left-to-right, Longest-Match Replacement

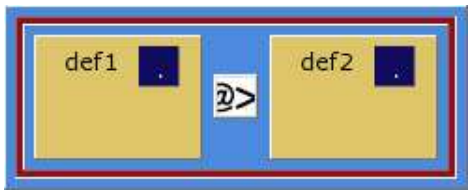

Figure 25: `Left-to-right,Longest Match Replacement` operator base. Displayed regular expression:

`[ def1 @-> def2 ]`

Opens with 2 default open slots. Possible to add another pair of slots for parallel replacement. Operator icon can be changed into: Replacement.

Special option: *New Condition* adds a condition with two open slots to the replacement operation. Only one condition can be defined per operator base.

7.8 Simple Markup

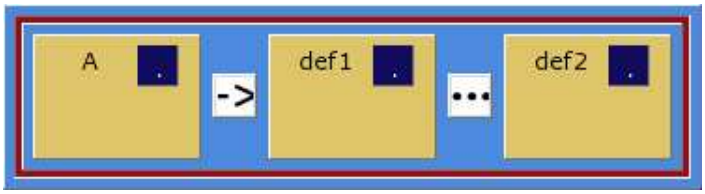

Figure 26: `Markup` operator base. Displayed regular expression:
`[ A -> def1 ... def2 ]`

Opens with 3 default open slots. Possible to add another triple of slots for parallel markup. Operator icon can be changed into: Longest-match markup.

Special option: *New Condition* adds a condition with two open slots to the markup operation. Only one condition can be defined per operator base.

7.9 Left-to-right, Longest-match Markup

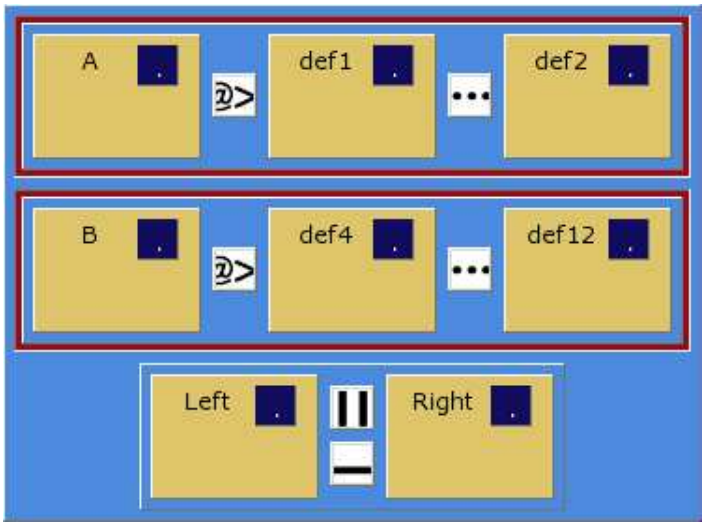

Figure 27: `Left-to-right, Longest-match Markup` operator base. Displayed regular expression:
`[[A @-> def1 ... def2],[B @-> def4 ... def12] || Left_Right ]`

Opens with 3 default open slots. Possible to add another triple of slots for parallel markup. Operator icon can be changed into: Simple markup.

Special option: *New Condition* adds a condition with two open slots to the markup operation. Only one condition can be defined per operator base.

8 Bug Reporting

The Vi-XFST project is still under development. Inevitably, despite our hard efforts on debugging the code, there may still be bugs. Therefore the debugging window, which is heavily used in the development process, is not removed from this release version. This window, when the message handler is installed and debugging enabled during compilation, displays various debug messages from Vi-XFST execution flow.

If you experience a bug to report to the authors, please send a copy of these messages that will let us track down the bug. We appreciate every bit of help to improve the code.

9 Authors

Vi-XFST is developed as part of a master thesis study at Sabancı University Computer Science department. About the avialability of the code and other documentation please contact the following authors.

Project Supervisor:

Prof. Kemal Oflazer <oflazer@sabanciuniv.edu>

Main Developer:

Yasin Yılmaz <yyilmaz@su.sabanciuniv.edu>