2LT - Two level transform GUI

A. Ribeiro D. Lapa

December 4, 2007

Abstract

This document contains instructions to how to install and use 2LT Graphical User Interface

Contents

1 Installation

These are the installation steps and requirements for Mac Os X.

1.1 Requirements

These are the requirements needed to run the 2LT GUI

X11 if you didn't install it when installing your Mac Os X then it is available at the Apple site

GHC v6.6 available at GHC site or through the MacPorts package manager

gtk2hs available at the gtk2hs site or through the MacPorts package manager

sdf-bundle v2.0.1 available at VooDooM project site. After downloading this program, his bin directory should be added to the PATH.

1.2 Installation

After all requirements have been fulfilled, download the 2LT tarball available at 2LT SVN site and unpack it.

Now lets call **$2lt** the root directory of 2LT after unpacked. Start a terminal and do these commands:

```
> cd $2LT/2lt-gui
$2LT/2lt-gui> make
$2LT/2lt-gui> chmod +x run
```

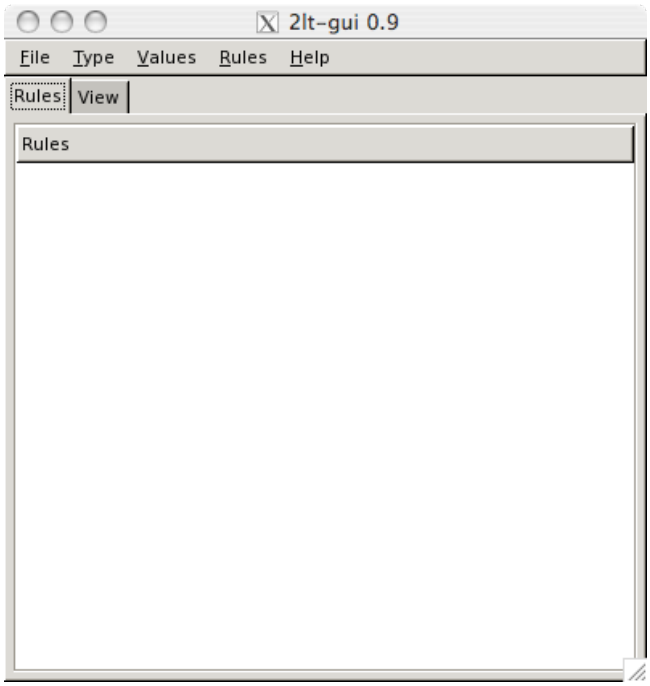
1.3 Running

To run 2LT GUI you must start **X11**, and normally it will start a terminal by itself. If it doesn't start a terminal you should start it yourself by clicking Applications > Terminal in the **X11** menu. Let **$2lt** be the root directory of 2LT and do the following commands.

```
> cd $2LT/2lt-gui
$2LT/2lt-gui> ./run
```

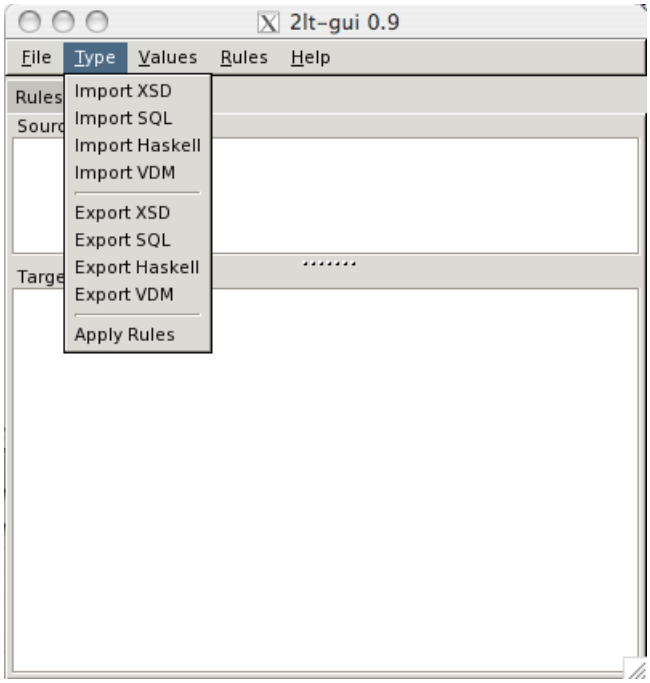
2 Using 2LT GUI

When 2LT GUI is started a window like this will appear:



2.1 Loading a model

First of all we need to load a VDM, XSD or SQL file. To do this change to the *View* tab and using the *Type* menu, import the file that you want. A popup window to select which file is to be loaded will appear.



After a file is loaded correctly, the source frame of the *View* tab will show the internal data type used to represent the type contained in the loaded file.

A little description of the symbols used to show the internal data type.

1 represents the unit type

A × **B** represents a product of the types A and B

A + **B** represents a co-product of the types A and B

A -\ B represents a map from A to B

[A] represents a list of A

{A} represents a set of A

u (A) represents the least fix point of type A

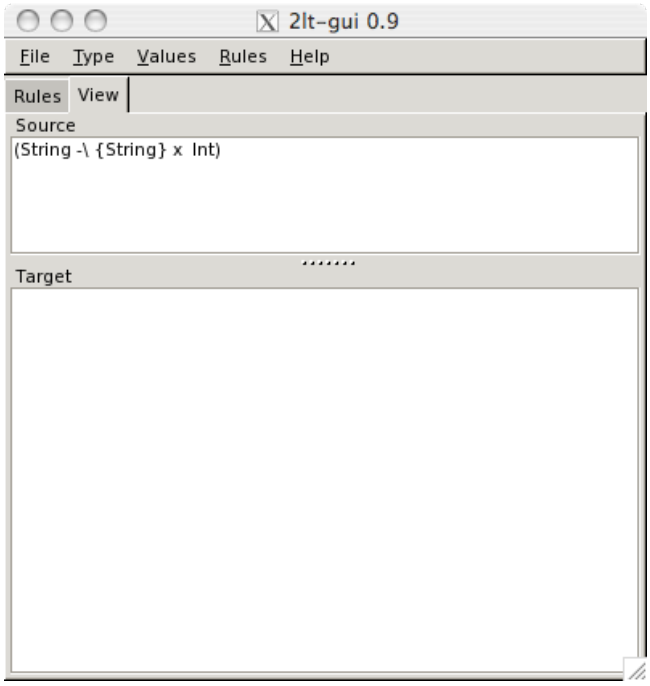
If we load *Bams_ types.vdm*, that is located in the examples directory which has the following contents:

```
types

BAMS = map AccId to Account;
Account :: H: set of AccHolder
           B: Amount;
AccId = seq of char;
```

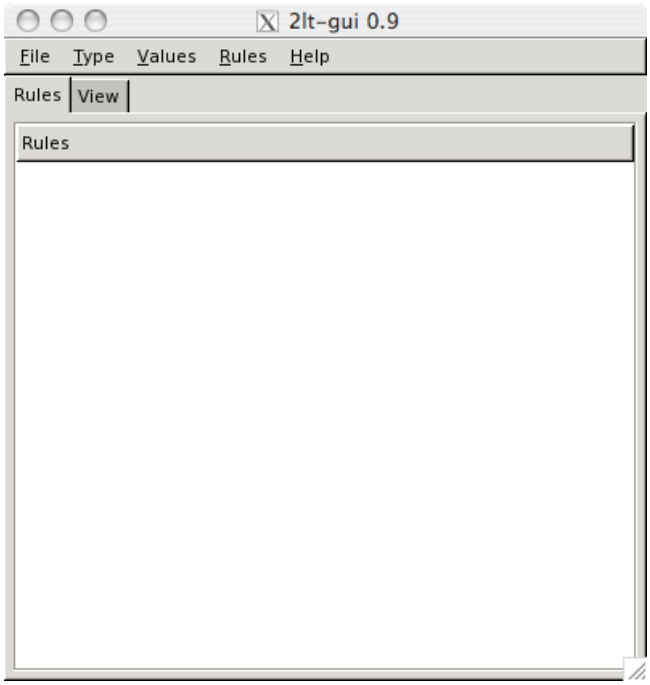
```
AccHolder = seq of char;
Amount = int
```

We will get something like this:

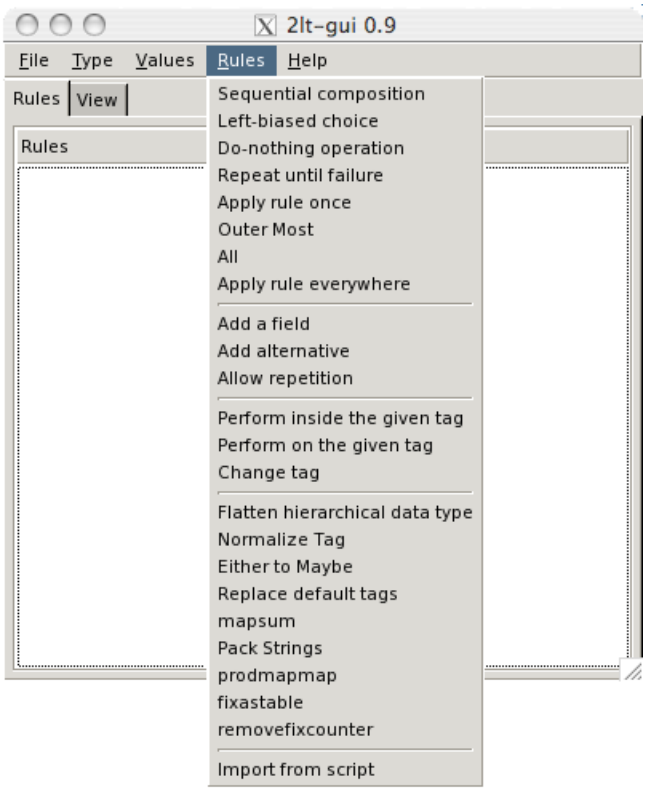


2.2 Choosing the rules to apply

2LT is not completely automatic, you'll have to choose which rules to apply to the model so it could be transformed according with that rules. There are two ways of changing the rules to apply, but to use either of them you have to change to the *Rules* tab.



When the *Rules* tab is selected you could either choose what rules to apply via *Rules* menu or use the option to import a script that is also in the *Rules* menu.



2.2.1 Writing a rule script

Rule scripts are very easy to be written, just use these rules that are shown bellow. The *tag* field must be a *String* encapsulated by with " or ' and the *rule* field could be any of the rules. Nested strings are represented like in some script languages, you have to change between one notation or another. When using sequential composition, you should use parenthesis around the rules in both sides or the result might not be the expected.

```
>>> : Used to compose other rules
||| : Used to compose other rules but with a left-biased choice
nop : Do-nothing operation
many rule: Repeat until failure
once rule : Apply argument rule exactly once, at arbitrary depth
addfield s1 s2 : Add a field s1 to the type s2
addalt atype : Adds an alternative
allowrep : Allow repetition
inside tag rule : Perform the argument rule inside the tag
at_tag tag rule :  Perform the argument rule on the tag
flatten : Flatten a hierarchical data type
all rule :  All
everywhere rule :  Apply argument rule everywhere
normalizetag :  Makes sure that the annotations on a type representation
                satisfy the requirements
eithertomaybe : Replaces Either based optional fields with Maybe
                based optionals
replacedefaulttags :  Replaces Either and Maybe based optionals by
                      outermost tagged Maybe optionals
changetag a b : Change tag from a  to  b
mapsum : " mapsum "
try rule:  Applies the argument rule if possible, but never fails.
packstr : Packs lists of char into strings
prodmapmap : Joins similar tables
fixastable : Eliminate recursivity
removefixcounter : Remove Fixastable conter
```

2.2.2 Some script examples

Bellow are some script examples:

- Script 1

```
(once nop) >>> (add_alt  "One")
```

- Script 2

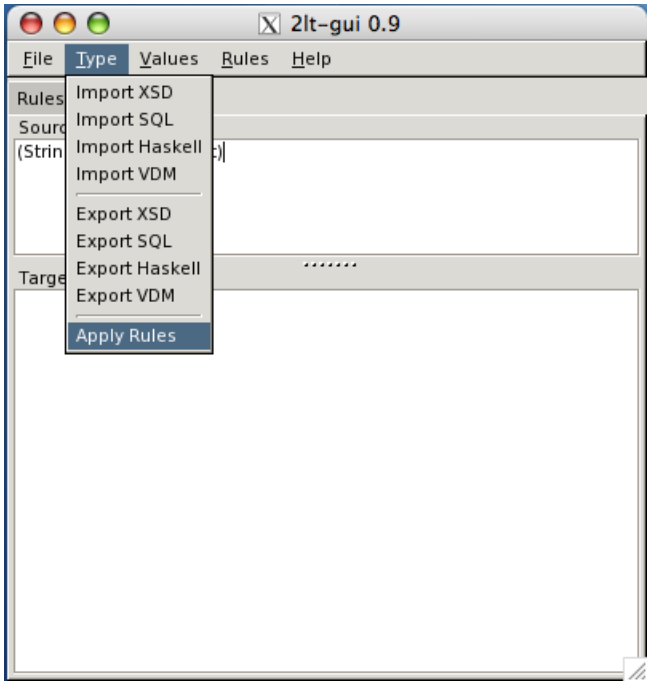
```
inside "Format" (add_alt "Tag 'Hello' String")
```

- Script 3

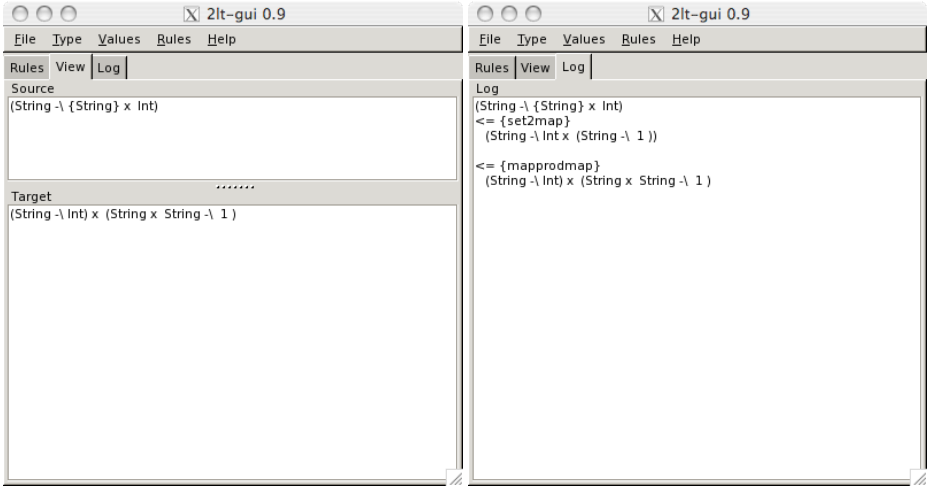
```
(everywhere (try packStr)) >>> normalizeTag >>>
(outerMost eitherToMaybe) >>> flatten >>> (outerMost mapsum)
>>> (many (once replaceDefaultTags))
```

2.3 Transforming types

After choosing which rules to apply and loading a model, you can find in the *Type* menu the option to apply the rules to the data type in the current *View* tab.



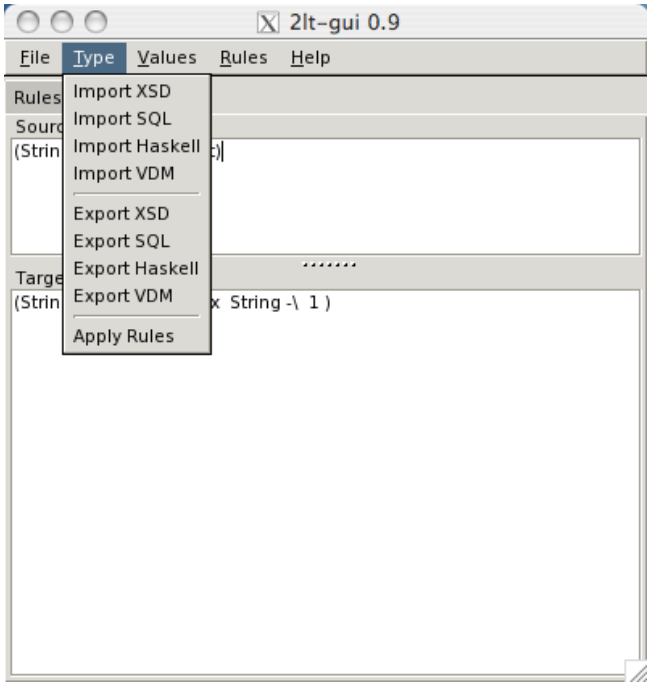
If all was done correctly, the *Target* frame of the selected *View* tab should be filled with a new type and a *Log* tab should appear, indicating which rules have been applied.



2.3.1 Exporting transformed types

When a *Source* type was properly transformed, it's possible to export *Target* type as a XSD, SQL or VDM file, the option is in the *Type* menu.

Warning: the type may not be exportable as some of the file types.

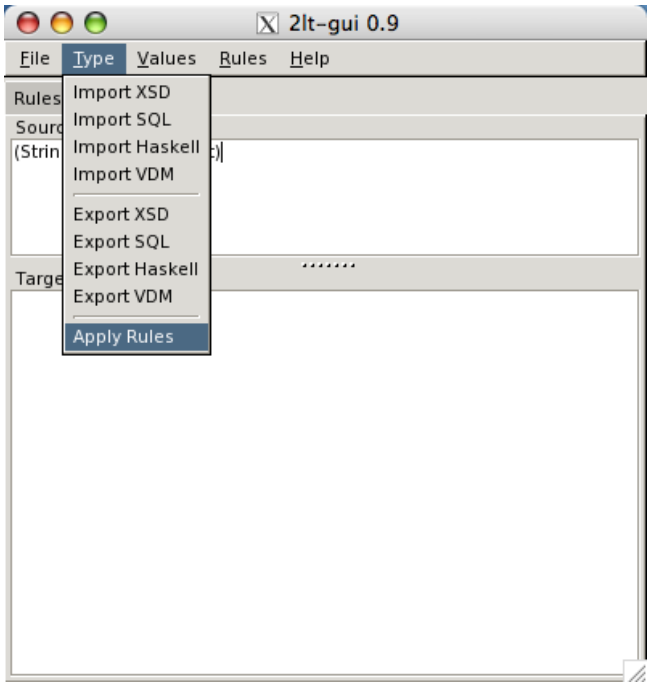


2.4 Migrating values

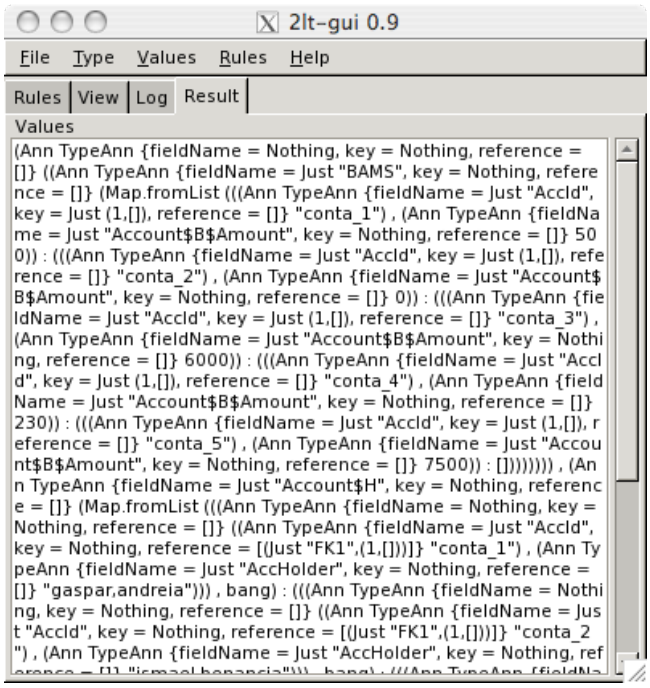
This section shows how to use 2LT to migrate values that are inhabitants of a certain type to values inhabitants of another type that is reached when applying the rules. The migration could go both ways, forward from *Source* type to *Target* type or backwards from *Target* type to *Source* type. To do this migration it is necessary that previously a type was already loaded and transformed.

2.4.1 Transform forward

If a value that is inhabitant of the *Source* type and you want to migrate it to *Target* type then the option *Transform forward* is under the *Values* menu. You must choose the correct file type to import and then a popup dialog to choose the file will appear.

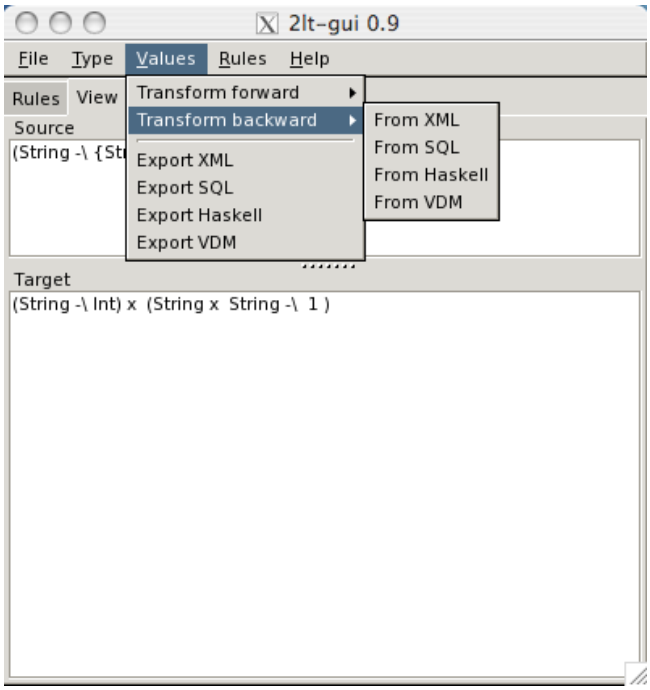


After the file is chosen, if the value on it is truly an inhabitant of *Source* type, then a *Result* tab will show up, containing the internal representation of the migrated value.

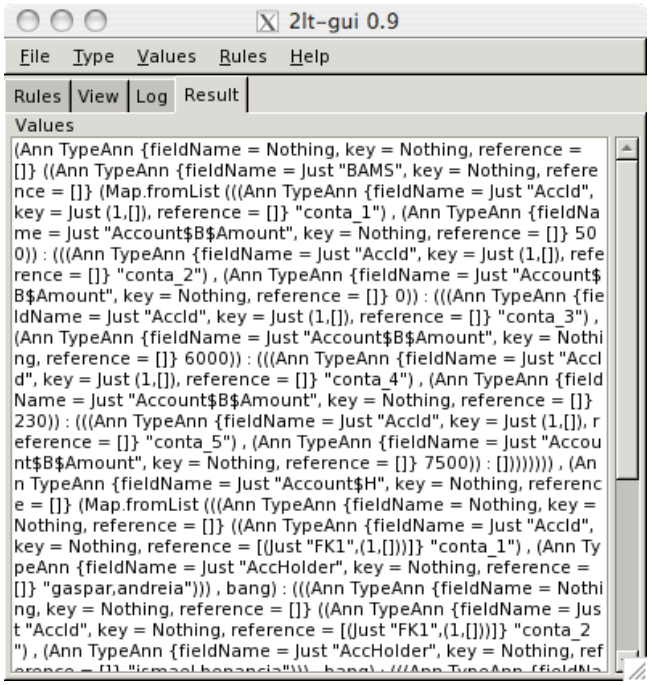


2.4.2 Transform backward

If a value that is inhabitant of the *Target* type and you want to migrate it to *Source* type then the option *Transform backward* is under the *Values* menu. You must choose the correct file type to import and then a popup dialog to choose the file will appear.



After the file is chosen, if the value on it is truly an inhabitant of *Target* type, then a *Result* tab will show up, containing the internal representation of the migrated value.



2.4.3 Exporting the migrated values

If a *Result* tab is select, the migrated value can be exported using the *Values* menu and choosing which is the file type (VDM, SQL or XML).

Warning: values may not be exportable as some file types and the VDM export is not yet implemented.