Table of Contents

Torero Homepage

What is Torero ?

The basic concepts of Attribute Grammars were introduced by Donald E. Knuth in 1968 and have shown themselves to be a useful formalism for specifying the syntax and the static semantics of programming languages.

Actually, ***Torero*** is a tool to assist the design of Attribute Grammars and supports the extension of Yacc Grammars to Ox Attribute Grammars. It is an interactive Java application that allows to make attribuations to yacc grammars in a graphical environment and with a graphical user interface. By means of example derivation trees it demonstrates and eases the procedure of making attribuations. This tool helps beginners to get a better understanding for Attribute Grammars and experts to gain an overview of a new grammar quickly and easily.

News

2002-05-04 Released Version b0.9
2002-05-04 Published Source of Version b0.9

2002-03-08 Released Version b0.8

User Documentation

Guided Tutorial [online html]
Reference [online html]
User Documentation download [gnu-zipped ps] [gnu-zipped html]

Installation Guide
Requirements

Download

Torero b0.9 package download
Torero b0.9 source and javadoc avaliable.

Torero b0.8 package download

How to start Torero as an applet in a browser.

Copyright

Contact

If you’ve any questions about out Project feel free to contact us via email.
If you’re encountering any troubles using Torero please send us an email with a brief error description and the input file(s) used.

Authors

Two students at the Institute of Computer Languages just now completing their studies.

Paul Wögerer
Thomas Aichinger

Guided Torero Tutorial

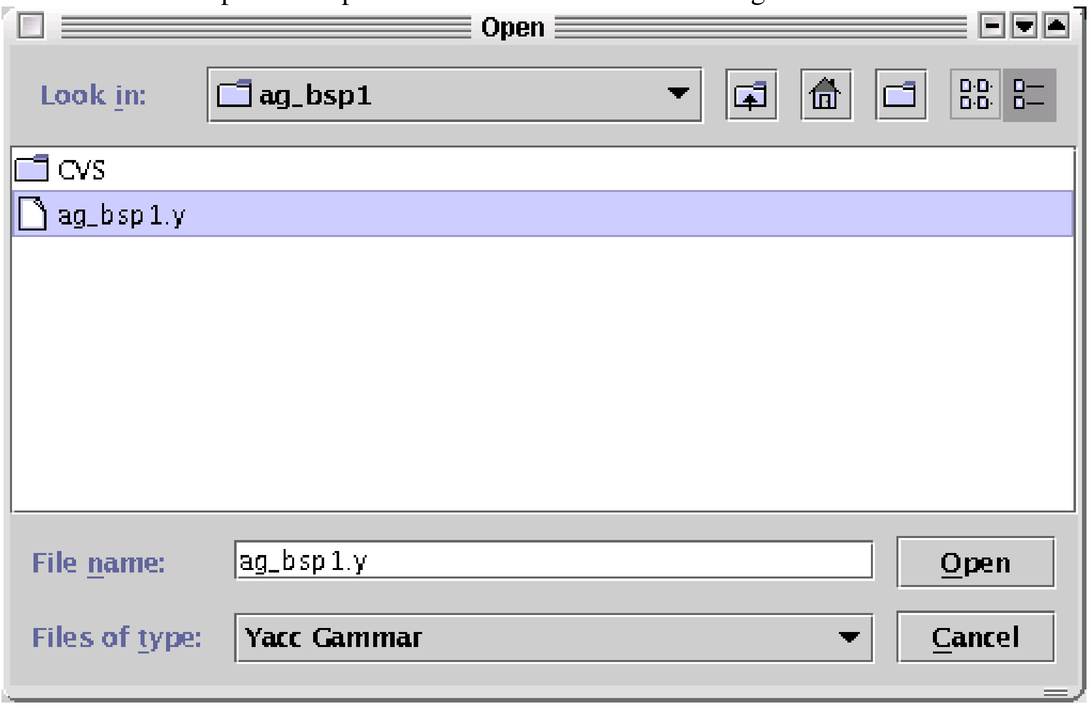
Introduction

This chapter explains the creation of an attribute gammar from a given yacc grammar and how to use Torero's tools. An attribute grammar for calculating the decimal value of a binary floating point number is used as example. Basic knowledge about Attribute Gammars is expected to understand the following steps. Being familiar with the ox notation for attribute references is also required because Torero is using the same syntax.
The example is also available as download under:

`http://www.complang.tuwien.ac.at/torero/download/examples/ag_bsp1.tar.gz`

Importing the yacc grammar

During the workflow a given yacc file is extended to an Attribute Grammar. After this is done it can be exported to the ox format for further processing.
Begin by loading the yacc file specifying the grammar of a binary floating point number. Click **File | Import Yacc Grammar** to open a file chooser dialog. Then select the yacc file **ag_bsp1.y** from the tutorial folder and push the open button in the file chooser dialog.



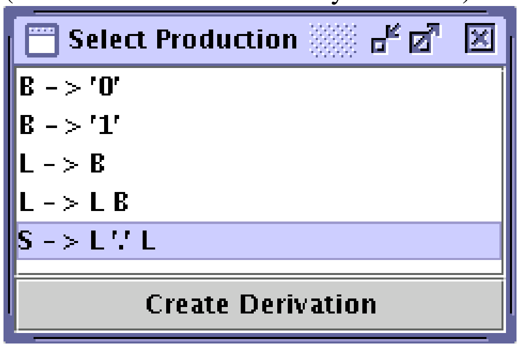
If the given yacc file is accurate, Torero shows an appropriate status message in the status line (Parsing and building ag_bsp1.y in 40 msec). If the yacc grammar contains some harmless faults, a parser log window will appear to inform about it. Torero tries to be as fault-tolerant as possible but has some limitations in comparison with yacc.

Browse the grammar

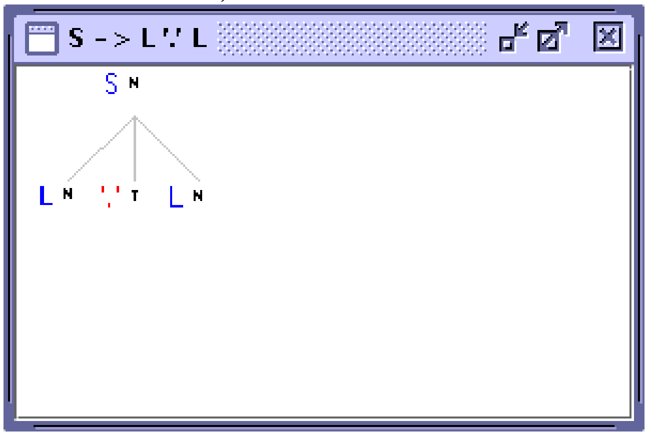
After loading the grammar it can be browsed by means of derivations. Explore the grammar by building some derivations. To start a new derivation click **Derivations | <New Derivation>**. This brings up a dialog for selecting a production as the root of the new derivation. Select the Production

```
S -> L '.' L
```

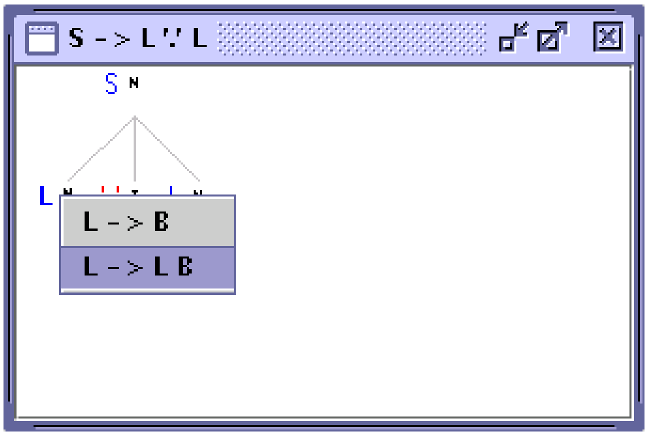
(This should be the last entry in the list).



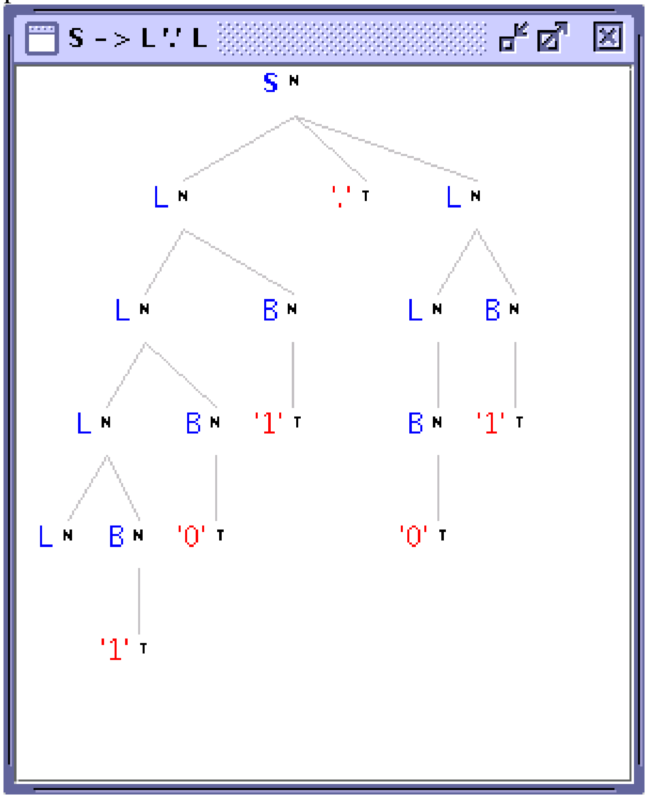
Press the **Create Derivation** button to move on. A new maximized derivation window appears on the screen. Now the previously selected production is visualized as a tree. It can be expanded easily. The tree is a graphical representation of a derivation in the current grammar. Now try to build a derivation for the word **L101,01**.



There are two kinds of nodes. The **red nodes** (with the little T on the right side) represent **terminal symbols**. The **blue nodes** (with the N) represent **non-terminal symbols**. This type of symbols can be used to extend the derivation. By pressing the right mouse button on it, a popup menu of productions which can be used to extend this non-terminal appears.



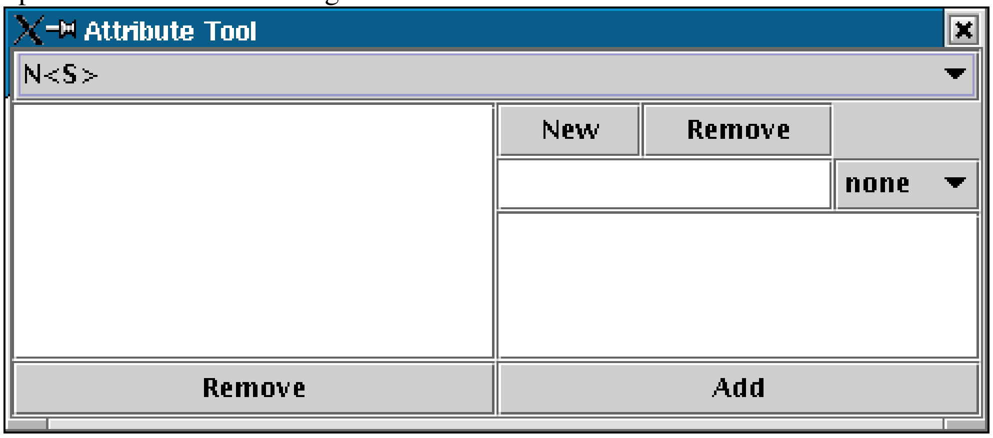
For the tutorial expand the non-terminal **L** on the left side of the tree with the production **L -> L B**. Apply the same production two times more on the leftmost non-terminal. Now try to apply productions standalone and build the derivation tree shown below.



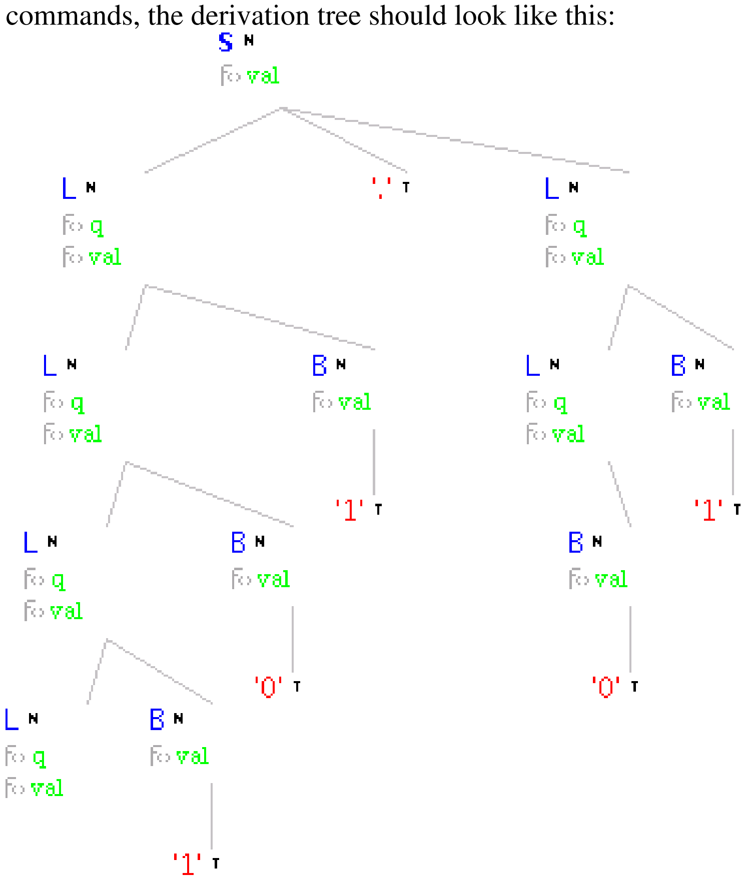
Define and assign attributes

Now define the attributes needed for calculating the decimal value from a parse tree representing the derivation of a word in the grammar of binary floating point decimals. For this two attributes are needed. One for calculating the decimal value of the binary number part of the binary float. Name it **val**. One more attribute for getting the information how often to shift-right (divide by two) the binary number part after the comma is needed. Name it **q**.

The Attribute Tool is designed to assist this procedure. Select the tool from the toolbar by clicking on it. (The one with the big **A** on it). Now click on the Symbol in the tree view you want to add an attribute. Klick now on the non-terminal **S** (which is the root Symbol in our Derivation Tree) to bring up the Attribute Tool Dialog.



Press the **New** button to create a new attribute. The text *New* appears in the textfield. Change it to **val** and commit it with enter. You just created a new Attribute named **val**. Now assign it to the non-terminal **S** by selecting **val** from the *Defined Attributes List* to the right and click on the **add** button. This assignment is immediately reflected in the active derivation window. Now select one of the other Symbols that need the **val** attribute in the derivation window. The active Nonterminal Combobox always shows the currently selected Symbol of the tree. Try to add the **val** attribute also to the symbols **L** and **B**. Finally add the **q** attribute to **L**. After having executed correctly all these commands, the derivation tree should look like this:



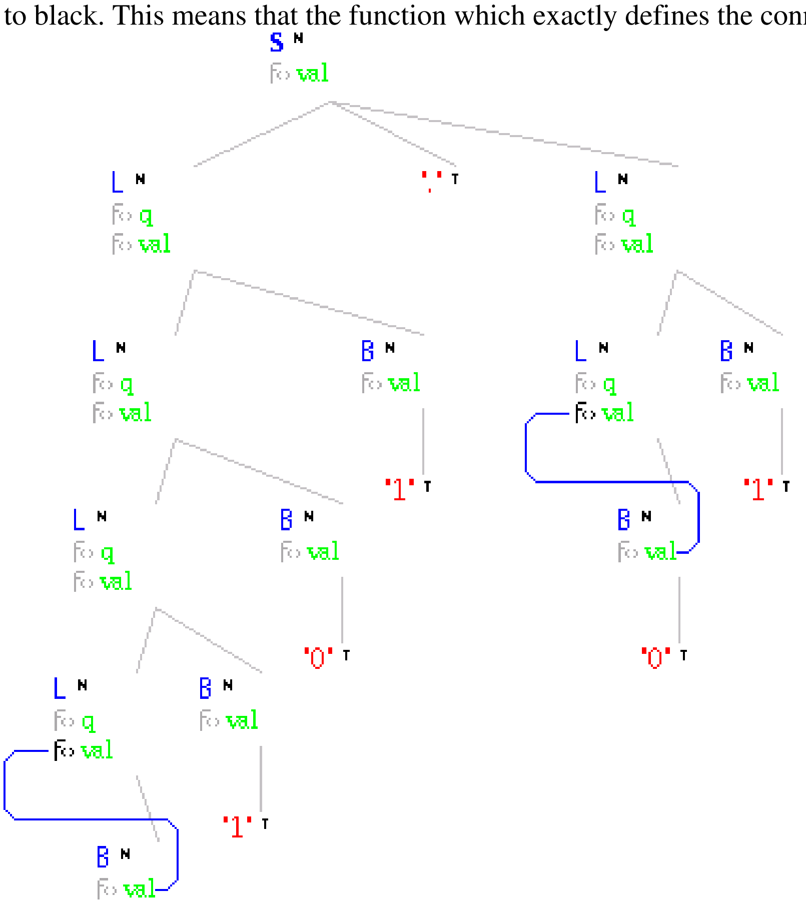
After this is done it's time to save the work. Do this by selecting **File | Save AG**. Use **.torero** or **.trr** as file extension for easy lookup at loading.

Make attribute connections & define functions

After having assigned all the neccessary attributes, connect them to each other with the Connect Tool. It's also neccessary to specify the mathematical relations between them using the Function Tool .

Begin by connecting the **val** attribute from the symbol **B** up to the symbol **L**. The tree in front shows two places where this connection can be declared. It doesn't matter which place is used for the next step, because it results in the same modification of the attibute grammar.

Select the Connect Tool from the toolbar. Now select the attribute which should be the **source** of your connection. Click on the **val** attribute of the **B** symbol which is under the **L** symbol. Notice that the mouse cursor changed it's state. This indicates to select the **destination** attribute of the connection. Do this by clicking on the **val** symbol above. The derivation tree immediately shows the connection everywhere this kind of production is used. At this moment the derivation tree should look like the one below. Notice that the little function icon at the left of the destination attribute has changed from grey to black. This means that the function which exactly defines the connection is defined and valid.



For this connection there is no need to edit the function. Whenever connections like this (exactly on source connected to one destination) are made, the representing function is generated automatically. However this doesn't mean that such functions cannot be changed.

Now use the function tool from the toolbar to make some attribute initializations. Select the tool from the toolbar and klick on the attribute **val** of a **B** symbol above an **'0'** symbol. Thus the function dialog appears and the little function icon shows that the function has become invalid. Now you can enter code for making the **val** attribute well defined on this location. Enter

```
@B.val@ = 0
```

into the function text field and press OK. If correctly entered the function state changes from invalid to valid. Now define

```
@B.val@ = 1
```

for the production

```
B -> '1'
```

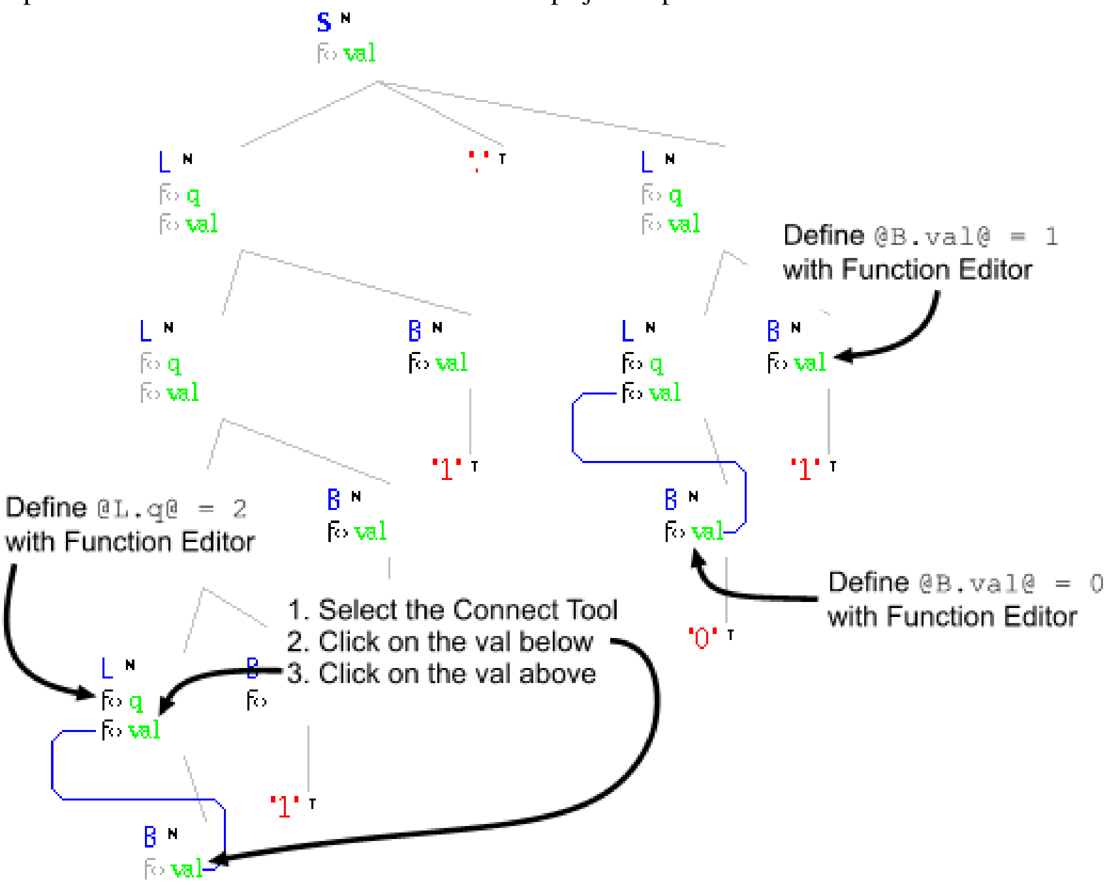
and

```
@L.q@ = 2
```

for the production

```
L -> B
```

The picture below is a brief instruction for the steps just explained.



Now complete the information flow of the **val** attribute in the production

```
L -> L B
```

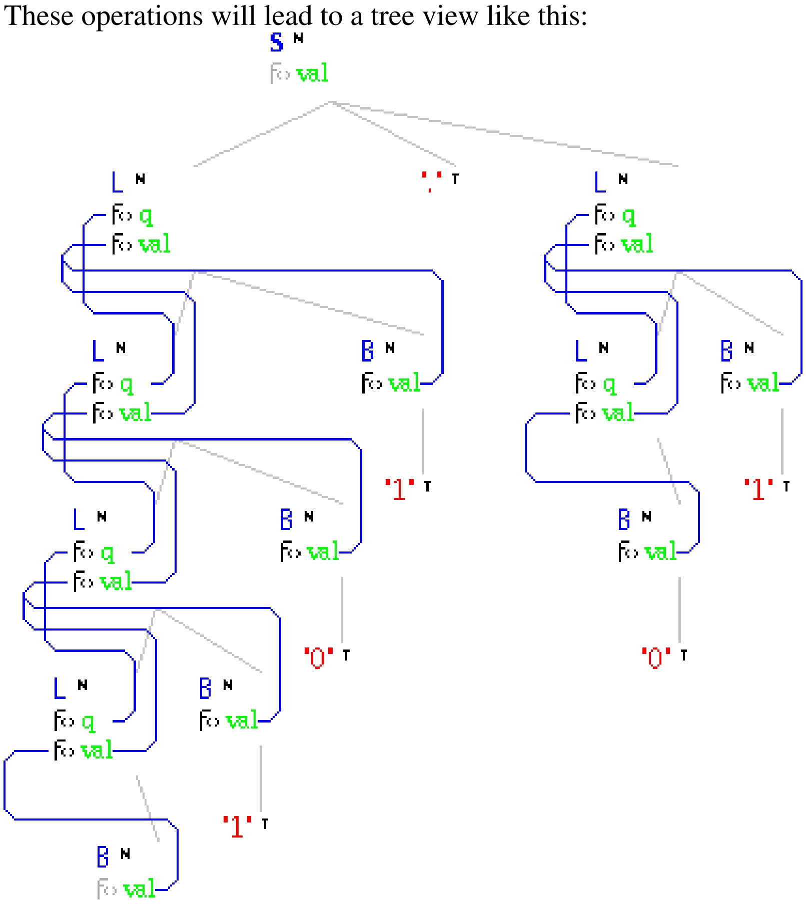
Therefore connect the **val** attributes from the symbols **L** and **B** to the **val** attribute on the **L** symbol above them. This time the corresponding function can't be generated automatically. It must exactly defined with the function tool:

```
@L.0.val@ = @L.1.val@ + @B.val@
```

Complete the information flow on the same production for the **q** attribute so that the relation between the **q** above and the **q** below is

```
@L.0.q@ = @L.1.q@ * 2
```

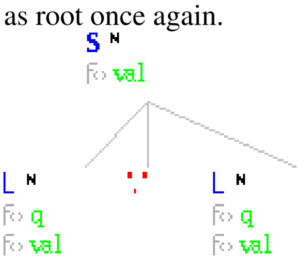
These operations will lead to a tree view like this:



Keep in mind that the tree view is completely dynamic. This means that changing the tree structure is allowed at any time. It's even possible to open a new derivation window at any time. Do so for completing the attribute connections. Just open a new Derivation Window via the Menubar and choose

```
S -> L '.' L
```

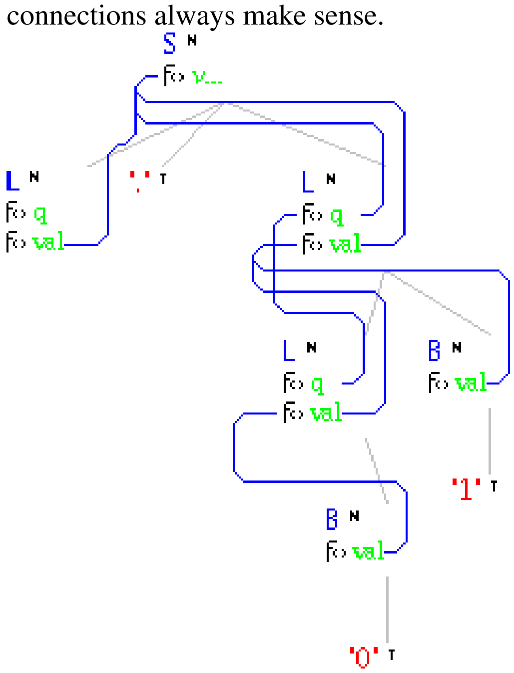
as root once again.



On this view define the last function as:

```
@S.val@ = @L.0.val@ + @L.1.val@ / @L.1.q@
```

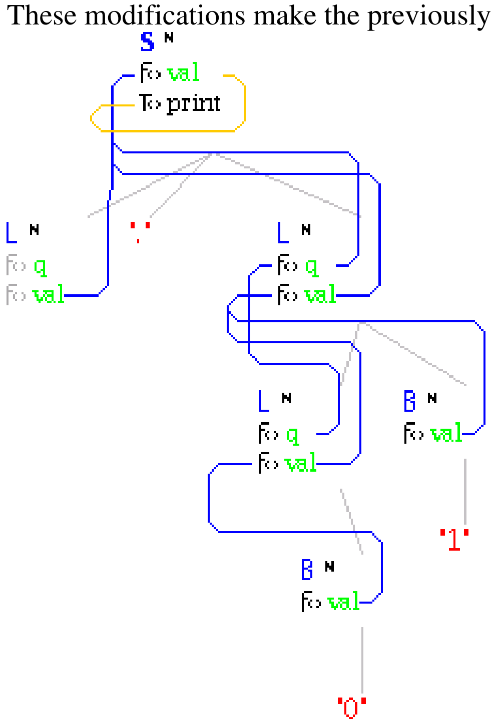
All done. Familiarize with this treeview and see that regardless which structure the tree has the connections always make sense.



There is finally one thing to do. Because a executable binary is made from this example a facility to inform the user about the result of the calculated value must be provided. This can be done by defining a traversal (name it **print**). It has to be added to the root production. This is reflected in the treeview with an "attribute like" entry on the symbol belonging to the left hand side of the production (it's the **S** symbol) the traversal is added to. All these procedures can be done with the Traversal Tool. To make the **val** attribute available in **print** connect the attibute to the traversal with the Connect Tool. Finally enter the source code with the Function tool:

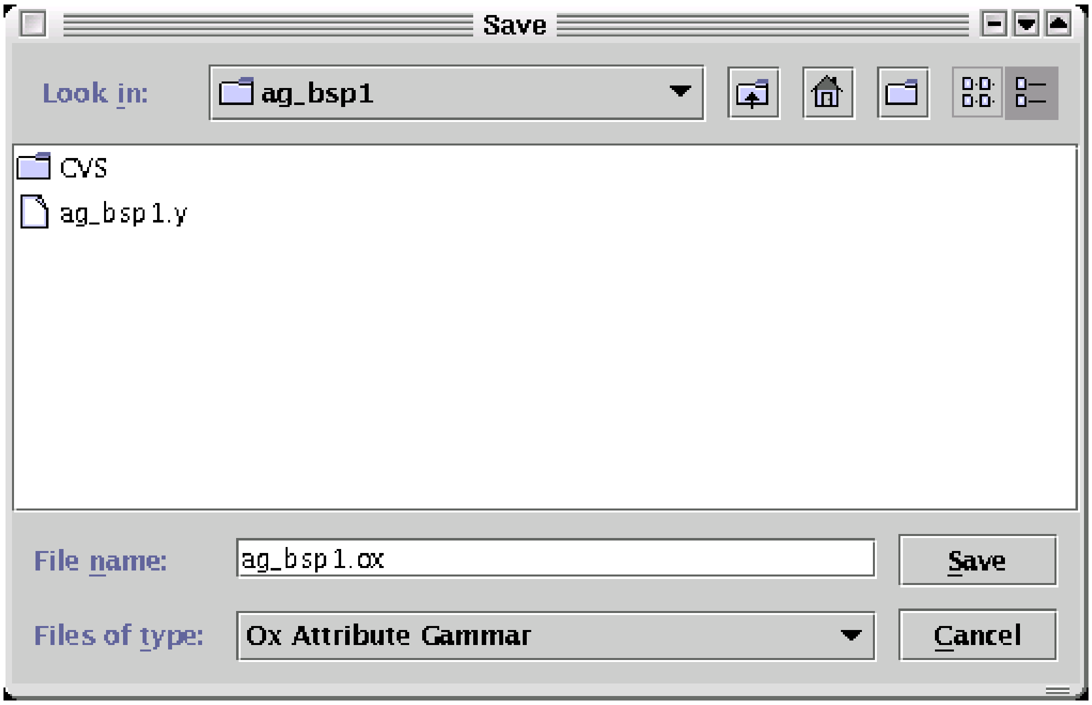
```
/* Print the calculated value to stdout */printf("%e\n",@S.val@);
```

These modifications make the previously shown picture look like this:



Exporting the Attribute Grammar to Ox

Now export the attribute grammar to the ox file format.



Define the Attribute Types

Until now the previously defined attributes have no data type. To provide this information a file named **attribute_types.h** have to be created . For this example it should contain:

```
#define T_VAL    float#define T_Q      float
```

because the used attributes have to be float values.

Compile & Run the Example

There is already a Makefile for building the example. Use it to get the binary from the source. The binary is named **calcdec** and can be tested with:

```
echo "110.001" | calcdec
```

Torero Installation Guide   Torero Home   Torero Reference

Torero Reference

The Toreo Reference provides detailed information concerning the following topics.

- Compatibility
- Menu Description
- Derivation Window
- Tool's Description

Torero Home Compatibility

Torero Compatibility

Input (yacc grammar) file compatibility

In Version b0.9 of Torero there are some restrictions on yacc grammars.
Torero always does a "best effort" import of the supplied yacc-grammars, if some parts of the input code are not supported by Torero it tries to ignore them and displays a log-window after importing a grammar, which contains info messages about the ignored code.
Torero does not make the same syntactic checks on grammars as yacc does. It is absolutely necessary that the input files conform to yacc's syntax, otherwise there could be situations where Torero will not accept the input grammar and won't display useful error messages.

- make sure that the input files have been compiled successfully by yacc and the c-compiler before importing them into Torero.
- the input files may not contain any attribuations.
- all c-code embedded in the rules section of yacc grammars (enclosed in curled brackets) has to be removed before importing (yacc "actions").
- yacc's %UNION and %TYPE specifications are not supported by Torero and should be removed from the input file before importing.
- all c-code embedded in the definition (before the first %%) or tail section (after the second %%) of yacc grammars is written to the output file "as it is".
- all c-code embedded in the definition section must not contain "%}" in strings or comments.
- literals have to be enclosed in single quotes.
- literals containing an escaped single quote (\') are not supported.
- key words (like %TOKEN, %LEFT) have to be UPPERCASE.
- in yacc-code, the semicolon should be used to mark the end of a rule only.

Yacc input syntax, with comments.

Output (ox attribute grammar) file compatibility

How Torero handels types (C-types) of attributes

Types of attributes cannot be specified in Torero directly, therefore the generated ox-file includes a headerfile where the user has to place the appropriate type definitions. The type names for attributes used by Torero are a simple extension of the attributes' names with a leading "T_", as shown below.

```
%{
#include "attribute_types.h"
%}

@attributes { T_VAL val; } S
@attributes { T_Q q; T_VAL val; } L
@attributes { T_VAL val; } B
```

All type names with the leading "T_" have to be defined by the user in the headerfile with the name "attribute_types.h". An example is shown below.

```
#define T_VAL   float
#define T_Q     float
```

Compatibility of different versions of Torero

Attribute grammars (.trr, .torero - files) saved with version b0.8 are not compatible to version b0.9 (and vice versa) because Torero (at this time) uses serialized JAVA object for saving and loading. There may also be problems when Torero is compiled using different JDK versions, so it’s suggested to use the torero.jar files provided in the download section.

Torero Reference   Torero Home   Menu Description

Menu Description

All menus in Torero are context sensitive, that means that it depends on the tool's state which menu items are usable at a time. Pictures below show states where all menu items are active.
Some menu items have shortcuts which are shown next to the menu items.

File menu

Import Yacc Grammar

A file chooser dialog opens to choose an input file which contains the yacc grammar. Per default all files with extensions .y and .yacc are shown.
Important: Depending on the version of Torero there are some restrictions on yacc grammars, which can be found in the compatibility section.
Only one instance of a grammar can be active at one time. When importing a new grammar, **the actual working grammar is discarded**, so save the working files first.

Export AG to Ox

Attribute grammars made with Torero can be written into an ox-input-file with this menu button. Note that the generated .ox file includes a C-Header file containing type definitions for attributes, which has to be supplied by the user. See the compatibility section for details.

Load AG

Previously saved attribute grammars made with Torero can be loaded with the menu item. Torero uses a binary format (JAVA serialised objects) for saving and loading attribute grammars.

Save AG

Save the actual attribute grammar. Torero uses a binary format (JAVA serialised objects) for saving and loading attribute grammars.

Export derivat to Dot

The active (focused) derivation window is written to a .dot file for use with Graphviz (formerly dotty).

Export derivat to JPG

The active (focused) derivation window is written to a .jpg file.

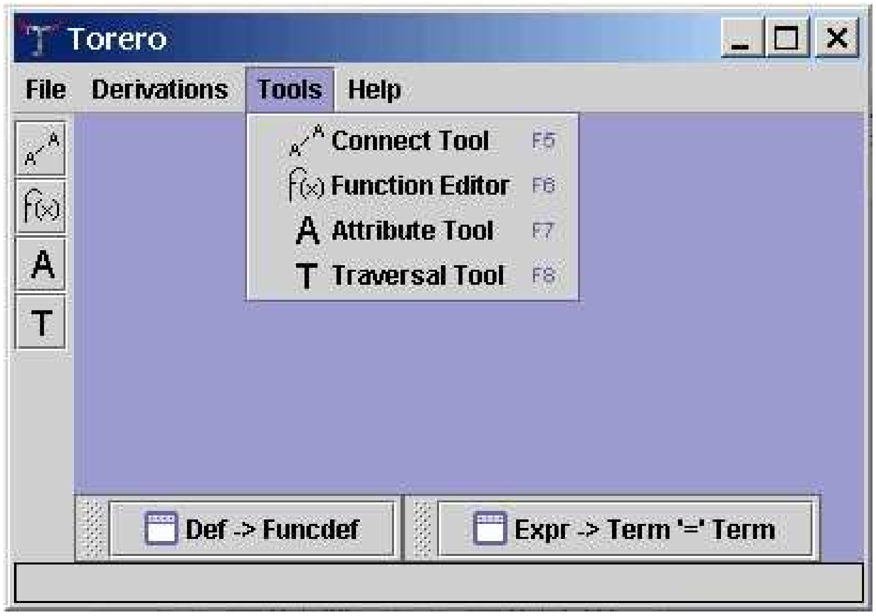
Derivation Menu

<New Derivation>

This menu item presents a dialog box, which allows to choose a production. Then a new derivation window with the selected production is opened.

All open derivation windows are listed below this menu item, they have the same names as the choosen production.
Previously created Derivation Windows are activated (focused) by selecting them from the list or clicking on the iconified derivation windows as shown in the example above.

Tool Menu



All available tools are listed, together with their shortcuts and can be activated by choosing from this list or using the Torero Toolbar. For detailed information about the tools see the Tool's Description.

Help Menu

Help

Displays a dialog with version information and a reference to the online documentation.

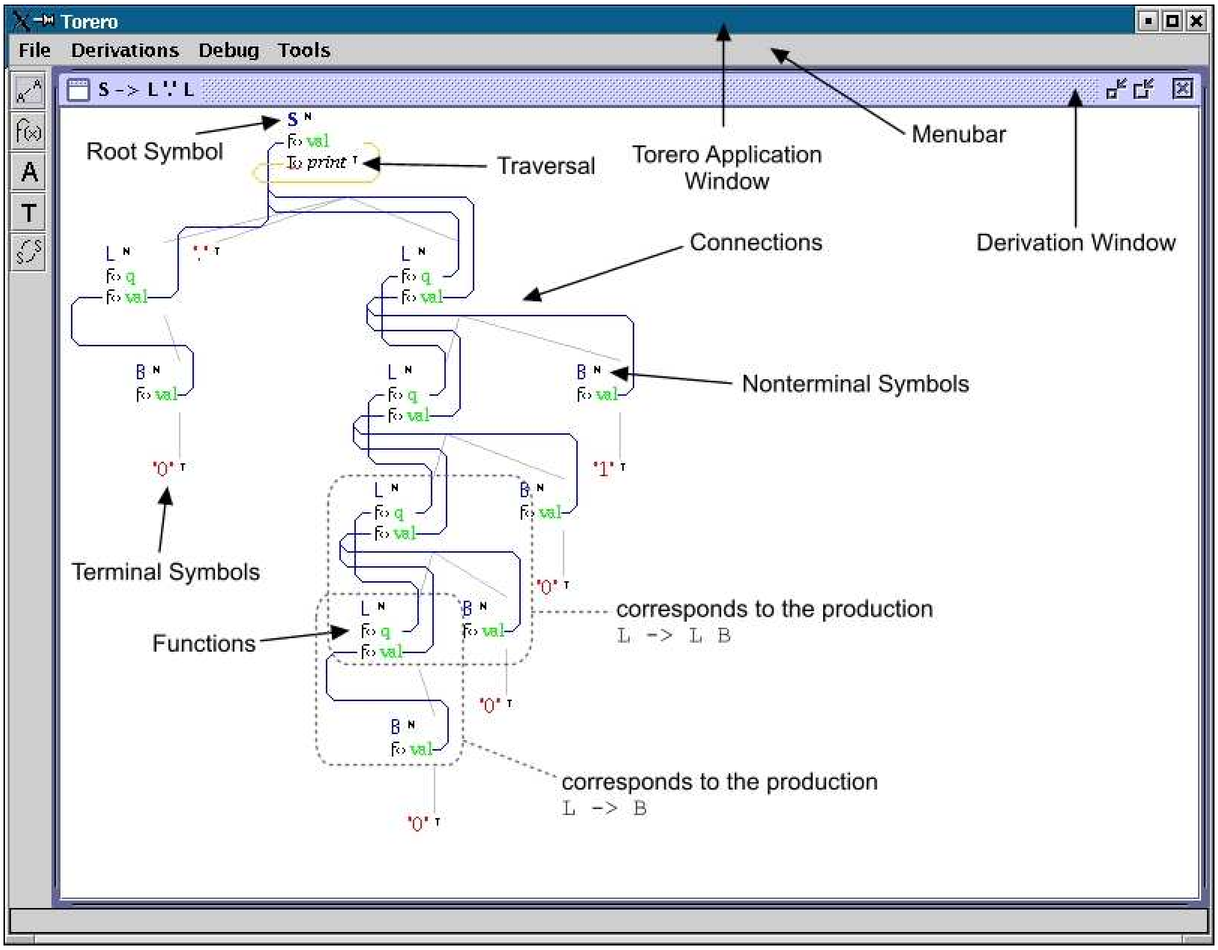
About Torero

Displays contact and licence information.

Torero Reference Torero Home Derivation Window

Derivation Window

Window description



Function and traversal symbols

Three different function symbols may appear next to an attribute's or traversal's name in a derivation window. They have specials meanings as described below.

symbol	description/meaning
	no connections are made and no function code is supplied.
	connections are defined, but either no function code is supplied or the function code is not valid.
	function code is valid.

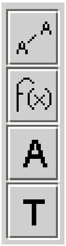
symbol	description/meaning
	no connections are made and no traversal code is supplied.
	connections are defined, but either no traversal code is supplied or the traversal code is not valid.
	traversal code is valid.

Note: invalid code means that either not all connected attributes were used, or that attribute names have been used which have not been connected with the connect tool.
If attribute grammars with invalid code are exported to ox, there will only be a comment in the ox-file in place of the invalid code sections.

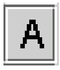
Torero Reference Torero Home Tool Description

Tool’s Description

Tools can be choosen from the Torero Toolbar as shown below. When a tool is active the cursor is changed to a small icon, showing the tool’s state. The cursor icon for each tool shown in the tool’s description.

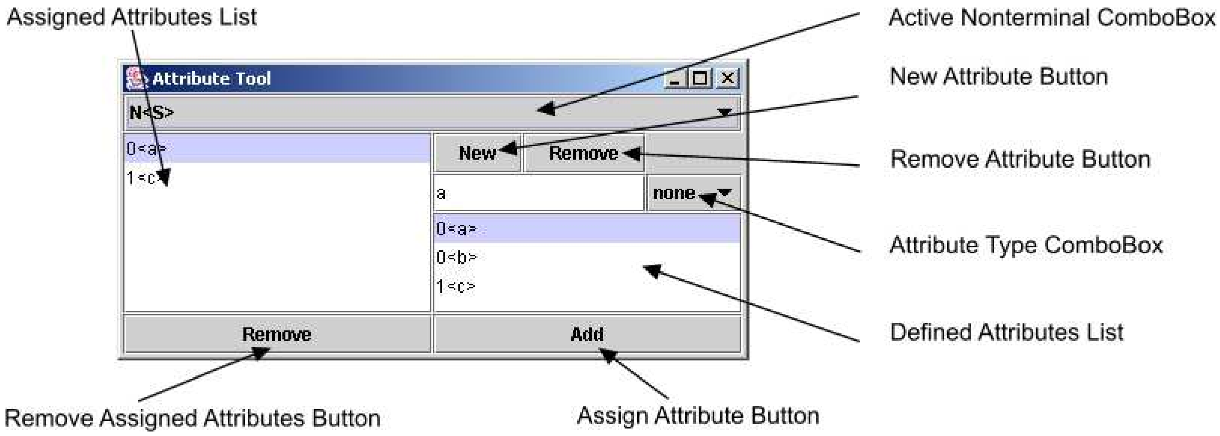


Please note that the Toolbar is only visible while editing an attribute grammar (after import of a yacc file, or after opening an attribute grammar).



Attribute Tool

After activating this tool in the toolbar, it can be used by clicking on a Nonterminal Symbol in a Derivation Window.



Creating new attributes

First use the *Active Nonterminal Combobox* to choose the desired nonterminal.
The *New Attribute Button* can be used to define new attribute names.
Use the *Attribute Type ComboBox* to specify the attribute’s type (none, autoinherit, autosynthesized). This can be done by simply selecting an attribute (the currently defined type is shown by the ComboBox) and then changing the combobox value.
All currently defined attributes are shown in the *Defined Attributes List,* this list supports multiple selection (use the Strg-Button).

Assigning attributes to symbols

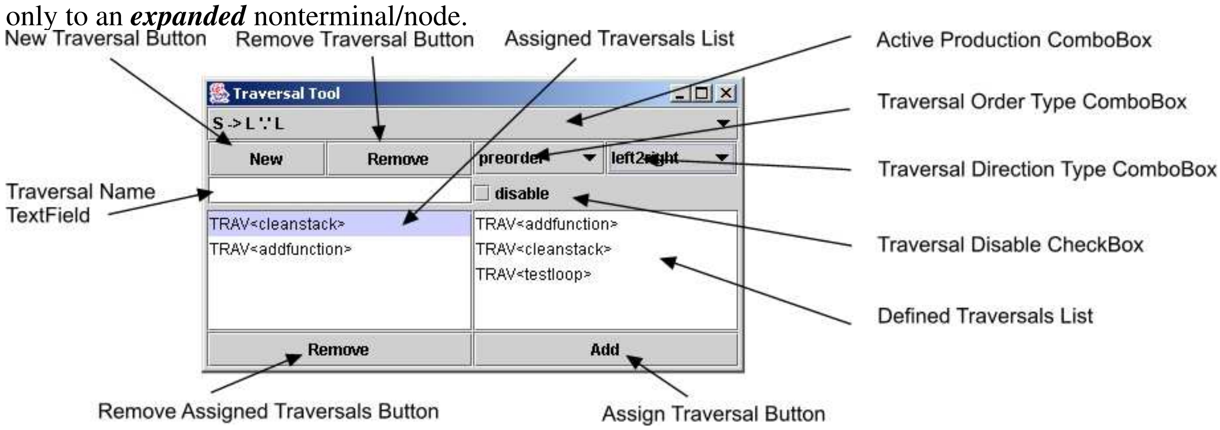
The *Assign Attribute Button* is used to assign all selected attributes to the active nonterminal, all currently assigned attributes are shown in the *Assigned Attributes List*. From there they can be removed using the *Remove Assigned Attributes Button*.

Selection Hints

The currently selected attribute from the *Defined Attributes List* ist shown in the textfield, if more than one is selected the first (in the list's ordering) is shown. There are some action that are only useful with one attribute at one time (renaming, changing the attribute's type), if there is a multiple selection only the first attribute is affected.
To remove a selection use Strg-Click (in both lists).

Traversal Tool

After activating this tool in the toolbar, it can be used by clicking on a nonterminal symbol in a *Derivation Window*. This tool is only used for definition of new traversal names and assigning them to productions. The source code for the traversal can be written with another tool (*Function Editor*) described later. Note that a traversal always is assigned to a whole production and can be assigned only to an ***expanded*** nonterminal/node.

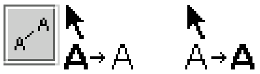


Creating new traversals

First use the *Active Production ComboBox* to select the active production. Click the *New Traversal Button,* overwrite the word "New" in the *Traversal Name Textfield* with the desired traversal name and hit the return key. Another way to create a new traversal is just typing the desired name into the (empty) *Traversal Name Textfield* and pressing the return key twice.
The traversals properties order type and direction type can be set using the Traversal Order Type ComboBox (default, preorder, postorder) and the Traversal Direction Type Combobox (default, lefttoright, righttoleft).
The *Disable Checkbox* temporarily disables the traversal and has the same meaning as the disable keyword in ox. It is mostly used for debugging purpose.
All currently defined traversals are listed in the *Defined Traversals List*. To remove a traversal completely select it in the *Defined Traversals List* and press the *Remove Traversal Button*.

Assigning traversals to productions

Select the traversal(s) in the *Defined Traversals List* and click the *Assign Traversal Button*. All traversals assigned to the active production are listed in the *Assigned Traversals List*. To remove a traversal from the active production select it in the *Assigned Traversals List* and press the *Remove Assigned Attributes Button*.

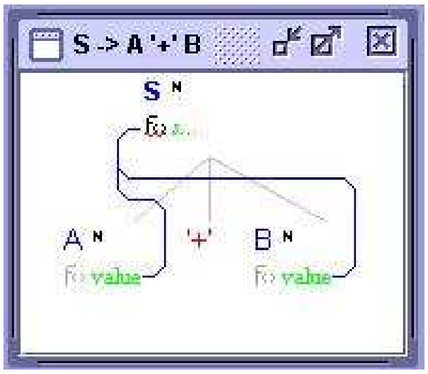


Connect Tool

This tool is used to connect attributes which is necessary before functions can be defined between them. The Connect tool works slightly different from other tools, it does not have an own window - only the cursor changes after activating it in the toolbar. The first icon shows the tool in the *Select Source State* and the second icon shows the *Select Destination State*.

What does a "connection" mean ?

A connection describes the relations between attributes, saying that (as in the example below) one attribute is **dependent** on one ore more other attributes. But it only defines dependencies and **not the exact** (mathematical) **function** between them.
Example: the derivation window below shows a connection between A.value, B.value and S.sum where S.sum is dependent on A.value and B.value. Therefore A.value and B.value are called *Sources* and S.sum is called *Destination* of this connection.

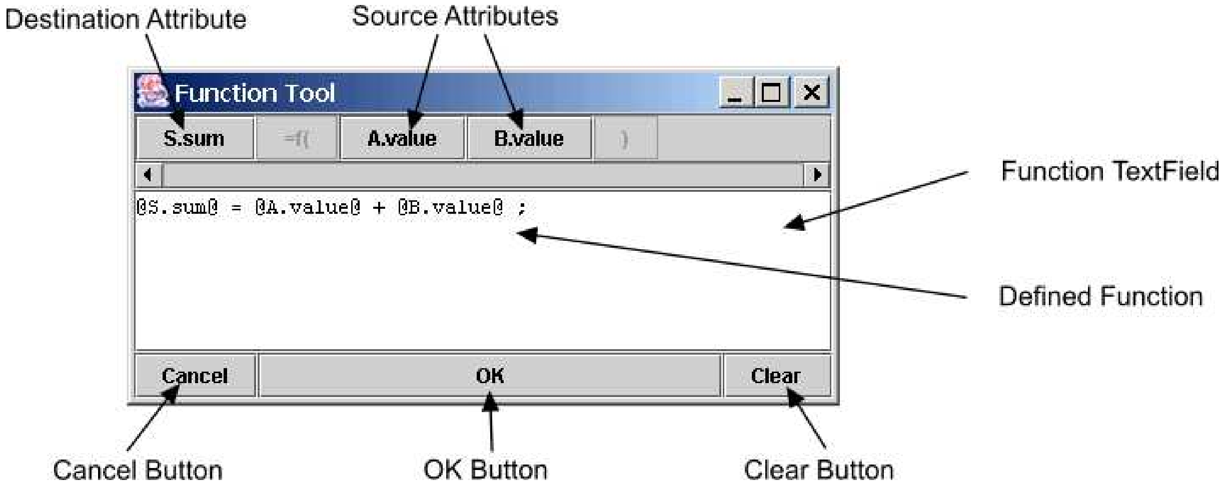


Making connections

After activating the tool it is in the *Select Source State*, now the source and destination are selected by clicking on the source and the destination attribute. A line is drawn between them and the f(x) symbol next to the destination attribute turns from grey to black (for details about f(x) see the function editor's section).
Multiple sources can be defined by simply repeating the actions above with different source attributes.
A connection can be removed in the same manner, first click the source that should be removed and then the destination attribute.

Function Editor

After activating this tool in the toolbar, it can be used by clicking on an attribute or a traversal (the *Destination Attribute*) in a Derivation Window. The tool can be used to define exact (mathematical) functions between attributes (traversals) previously connected with the connect tool and to define traversal code. The defined text is written "as it is" into the output file, therefore it has to be in the correct target language (ox and C in most cases).

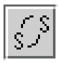


Inserting functiontext

Using this tool is simple. To insert the previously *"connected"* attributes into the ox-code, click on the *Destination Attribute* Button and/or the *Source Attribute* Buttons - a complete ox notation with the surrounding @ ... @ is inserted. As mentioned above the text is taken into the output file "as it is", after clicking the *OK Button* a few checks are done on the *Defined Function*: All used attribute (traversal) names must be either the *Destination Attribute* or be in the list of *Source Attributes*. The *Destination Attribute* and all *Source Attributes* have to be used to define a correct function. Note that any text enclosed by @'s is considered to be an *Attribute*.

Buttons

OK Button: The entered functiontext is checked and the window is closed if the check was successful.
Cancel Button: Modifications are discarded and the window is closed.
Clear Button: The whole functiontext and the connections to this destination attribute are cleared and therefore the Function Symbol (in the derivation window) changes to *no function defined*.
For the exact meanings of the different Function Symbols (in the derivation window) see the function symbols in the derivation window section.



Swap Tool (only in debug mode, -v -vv)

There is an additional tool that was implemented to illustrate the power of Torero's grammar model.That's the only tool that allows modifications to the underlying grammar. It is only available if Torero is started in debug mode with the command line option -v, -vv given.

After the Swap Tool is activated in the Torero Toolbar the cursor changes. With this tool you can swap two grammar symbols in a production by clicking them with the left mouse button.

Torero Reference Torero Home

Torero Installation Guide

Step by Step (standalone application)

1. Make sure that JAVA is installed properly and it’s in your path. Test it by typing "java -version" in a shell. On unix-systems java is usually installed in the "/usr/lib/java" or "/usr/lib/jdkXX" directory.
2. Go to your home directory and create a new directory. e.g. Torero.
3. Download and unpack the Torero package to the new directory. e.g. "/home/yourhome/Torero"
4. There you’ll find two start scripts: torero.sh (unix) and torero.bat (windows). Call the appropriate file to start Torero.

Command line arguments (standalone application)

If Torero is used as standalone application several command line arguments are supported:

```
torero.sh [-v | -vv | -y<yaccfile>]
```

- **-v** be verbose
- **-vv** be very verbose
- **-y<yaccfile>** parse yaccfile, print debug info & exit

Step by Step (applet version)

1. Follow steps 1. - 3. as described above.
2. Go to the Start Torero as an applet section and follow the instructions found there.
3. Call the appropriate "runapplet" script (runapplet.sh for unix, runapplet.bat for windows systems) found in the Torero directory.

Requirements | Torero Home | Howto Start Torero as an Applet

Starting Torero as a Java Applet

Running Torero as applet from your local harddisk

1. download the lastest torero package.
2. enter the root directory of the package and type "sh runapplet.sh" (unix) or "runapplet.bat" (windows). In this case you don't need the policytool.

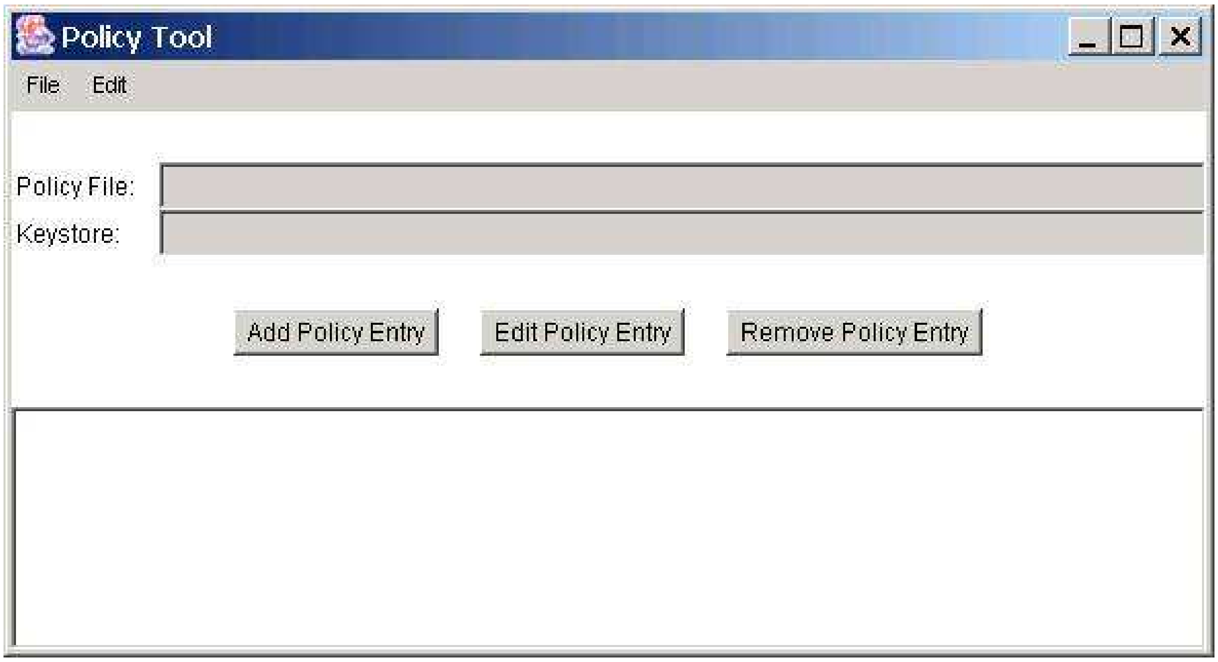
Running Torero in a Browser from the Web directly

Requirements

1. You'll need a JAVA enabled browser (most netscape, mozilla, opera, IE, ... are) with a JAVA-Plugin with version number equal or greater than 1.3.1. See the Requirements description section for details.
2. Torero needs some access-rights on the machine (they are defined with the java-policytool)

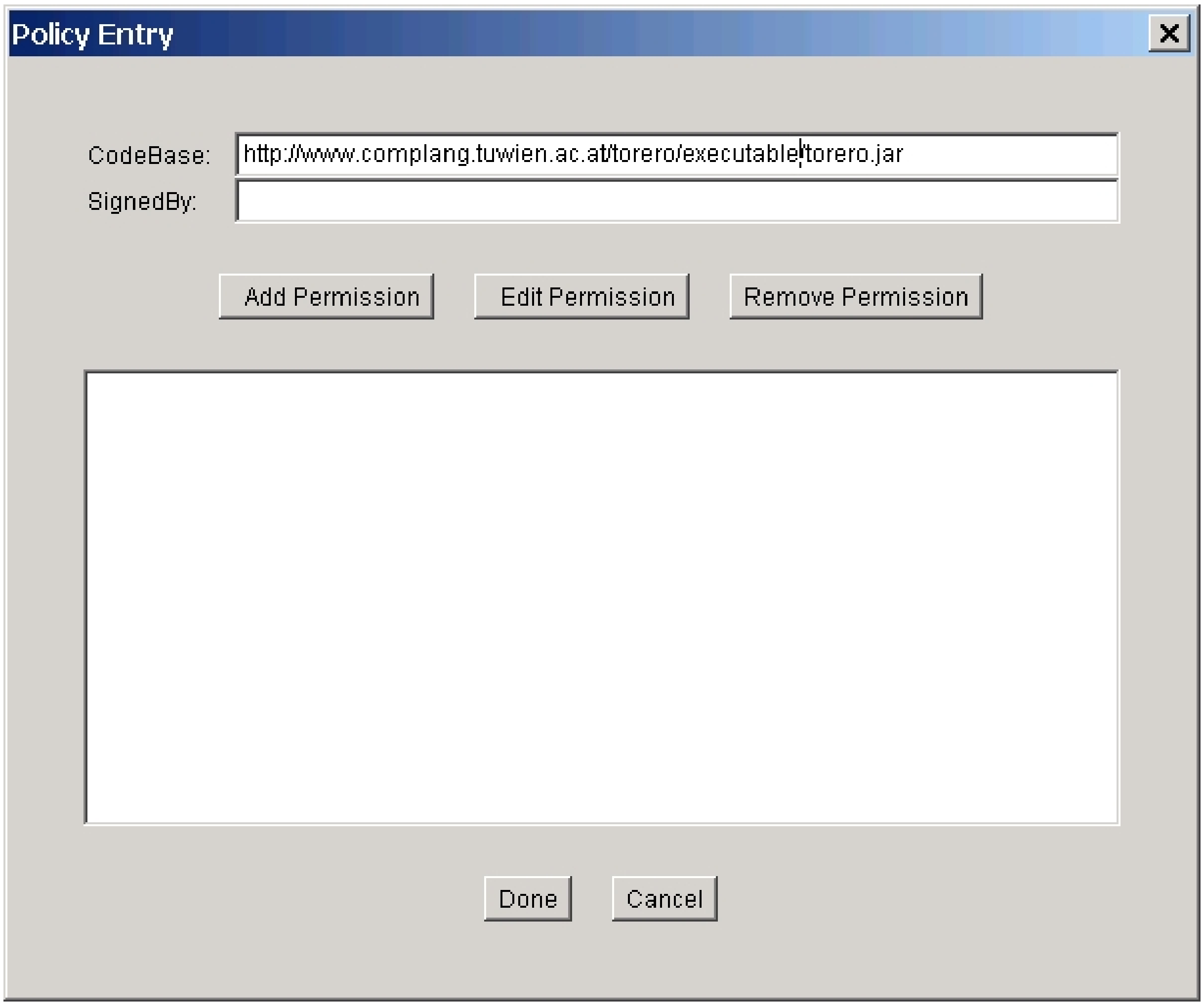
Defining access rights

Torero needs some access-rights (e.g. for reading input grammar files, writing ox files), they are defined using java's policytool. It is started by typing "policytool" on an command line/shell. If it is not in the *path*, locate the java directory and enter the "/bin" directory. Then type "policytool again. A dialog should open as shown below.

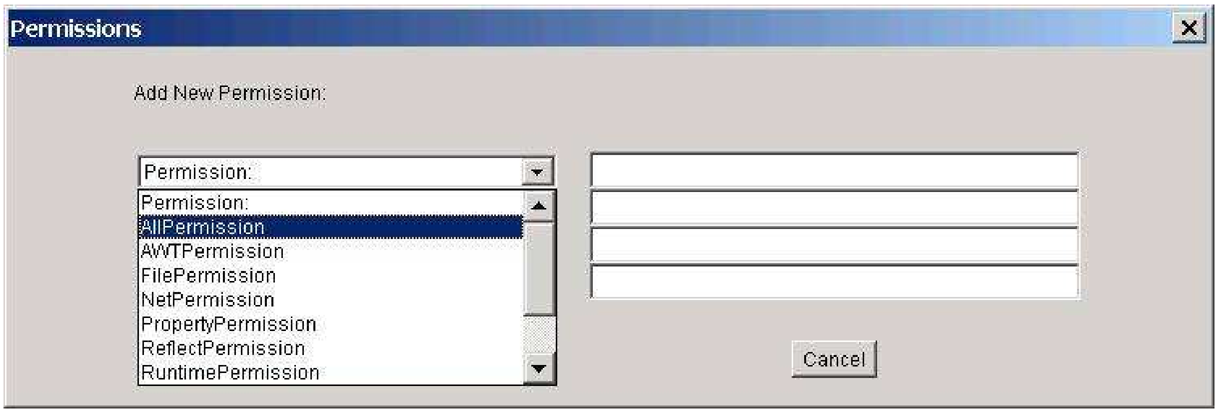


Then Click on the "Add Policy Entry" Button.

"Ok".



Enter the url and the name of the .jar file into the "CodeBase" field. If you start Torero directly from the web it is "http://www.complang.tuwien.ac.at/torero/executable/torero.jar".



Click on the "Add Permission Button", select AllPermissions in the Permission ComboBox and click "Ok".

If you want to save your settings use File|SaveAS and enter a filename for your "Policy File". Usually this file is named ".java.policy" and is located in the user's home directory. Note that in Windows2000/XP this is the directory beyond "Eigene Dateien". Close the policytool.

Now Torero can be started directly under Start Torero.

Back Torero Home

Torero Requirements Section

What do You need to run Torero ?

Torero is written in JAVA, therefore a Java Development Kit (JDK) or at least a Java Runtime Environment (JRE) is needed to run Torero. It is designed either to run as a standalone application or to be started as a JAVA Applet, in this case it is even possible to start it in a Browser with a JAVA-Plugin.

Torero b0.9 and b0.8

Java Version 1.3.1 or any newer 1.3.x version is required. Both the SUN and the IBM implementation will work, they are available for several operating systems.

Plattforms Torero has been tested on

Torero b0.9 and b0.8

Linux i386, JDK 1.3.1 (SUN)
Linux i386, JDK 1.4 (SUN) some minor problems, will be fixed in the next version.
Linux Alpha, JDK 1.3.1 (Compaq)
Windows 2000, JDK 1.3.1_02 (SUN)

How You can find out, which JAVA Version is installed

open a command shell and type

```
java -version
```

If you see something like

```
java version "1.3.1"
```

JAVA is installed correctly on your machine.

Back Torero Home