Thistle: diagram display engines and editors

Jo Calder
Language Technology Group, Human Communication Research Centre
and Centre for Cognitive Science
University of Edinburgh
email: `J.Calder@ed.ac.uk`

July 24, 1998

Introduction

Warning: The software described here is under development and liable to change. You are reminded that it comes with no warranty.[1]

This document constitutes the reference documentation for the diagram display engine and editor *Thistle*. There is a home page[2] for Thistle on the world wide web. The version described here is 1.5. The integration of regular expressions §A.1 has status *beta*, otherwise the status of the program is *full release*.

We are very keen to hear any comments, criticisms *et cetera* you might have on the system. If you are reporting something you deem to be an error in programming, please read the list of known problems in §B below, before contacting us. Items in that list are referred to in the documentation below using the word *problem* followed by a paragraph reference to §B.

Conventions used in this document

[Adapted from the conventions used in [2, 3].]

`typewriter font`	This is used for all UNIX facilities, switches, directories, filenames and anything else the user may type as part of a command, e.g. `thistle -c hpsg94.spec`.
`italic typewriter`	This is used for variables in the above. For example `thistle -c hpsg94.spec` *`filename`*. Here the variable *`filename`* needs to be replaced by some actual filename.
`[ ]`	Optional elements of program syntax. For example: `thistle -c hpsg94.spec [-startHidden]`. This indicates that `-startHidden` is an optional element in this command.
sans serif	Used to indicate operations that can be invoked by the user. (Note that in the hypertext version of this document, this font may be rendered as a serif font.)
`typewriter font`	Used to display fragments of code.
italic	Used to indicate terms being defined.

[1] http://www.ltg.ed.ac.uk/software/thistle/no_warranty.html
[2] http://www.ltg.ed.ac.uk/software/thistle/home/index.html

1 General description

We present here some novel software for the manipulation of diagrams. Our principal motivation is that of supporting the creation of diagrams representing linguistic analyses, and for the mechanical interpretation of such diagrams. Accordingly, we deal not only with aspects of the layout of diagrams but also with their *content*.

Each display engine or editor is parameterized by a *diagram class* which specifies the layout of diagrams and how layout and content are related. Every diagram class provides a variety of *types* of diagram and a grammar which indicates how a larger diagram can be made up of smaller diagrams of particular types. Many examples of diagram classes and particular diagrams are provided.

2 Display engine

Diagrams may be displayed on Web pages and there is a growing collection of collection[3] of examples available. Examples there include trees automatically derived from a parsed corpus (from the HCRC Map Task Corpus[4]). For details of how to add a diagram to a Web page, see §6.2.

Some displayed diagrams may be edited, one allowing trees and AVMs, for example[5]. If editing is enabled for a diagram, clicking anywhere within the pane the diagram is displayed on will bring up an editor. (This can happen only once for any instance of a diagram, unless the corresponding Web page is reloaded.) A new window is created with a menubar allowing control of the editor. Sequences of diagrams may also be displayed, useful for diagrams which, taken in sequence, represent the changing state of some process, for example the state of a parser.[6]

3 Editing diagrams

The editor is started either by clicking on a diagram in an Applet (if editing of the diagram is enabled, see §6.2 below), or by invocation from a command line.

3.1 Command line options

Under UNIX systems, Thistle may be invoked as followed:

`thistle -c` *`classfile`* `[-s` *`scriptfile`*`]`

or

`thistle -c` *`classfile`* `[`*`file`*`]`

In the second case, *`file`* must refer to an *existing* diagram file. If Thistle successfully loads the diagram contained in *`file`*, that file then becomes the current file. If loading is not successful, the association with that file is broken.

The following options are understood by Thistle:

`-c` ***`classfile`*** specifies the class of diagrams to be edited

`-s` ***`scriptfile`*** specifies file to be run as a script (see §5.3)

`-startHidden` requests that continuation points are not shown on start up.

The environment variable `SPECSPATHS`, if defined, is assumed to contain a set of path names of directories which are searched sequentially for the diagram class file of the name supplied. (The character `:` should be used to separate directory names.)

[3] http://www.ltg.ed.ac.uk/software/thistle/demos/
[4] http://www.cogsci.ed.ac.uk/hcrc/wgs/dialogue/dialog/maptask.html
[5] http://www.ltg.ed.ac.uk/software/thistle/demos/example4.html
[6] http://www.ltg.ed.ac.uk/software/thistle/demos/example_parser1.html

3.2 Window layout and interaction

The window layout of the editor is largely constant across diagram classes. At the top of the window is a menu bar (see §4) allowing operations of different kinds. Beneath that is a line containing a text input box. Below that is the *working area*. At the top of the working area, you will see the *control panel*, which shows the types of diagram available within the editor. On starting an editor, no types will be available. As different subparts of a diagram are selected (see §3.3), the types of diagram available at that point are indicated.

The title bar of the window indicates the diagram class in use, and the name of the file associated with the diagram (or `(no file)` if no file is associated with the diagram). In addition, the name of the file is preceded with the character `*` if the diagram has been modified and by the characters `%%` if the file's permissions are such that it is not writable by the user.

At the bottom of the working area is the part of the screen in which diagrams are constructed, the *drawing area*, and which is equipped with scroll bars to allow large diagrams to be manipulated. In the examples mentioned so far, a diagram will already have been placed within this window. In many of these cases, the diagram is "complete": you can't add any more information to it. Here's what you will be able to do:

- Click on a subpart of the diagram.

  The area occupied by the subpart of the diagram will be indicated (see also §3.3 below).

- Then select the menu item Kill from the Edit menu.

  The subpart of the diagram will be removed. According to the design of diagrams in use (and whether "continuation points", described shortly, are being shown), the deleted part of the diagram may be replaced by one of three symbols:

  $\boxed{?}$ an *insertion point*, which indicates that a single diagram can be introduced at this place in the diagram or

  $\boxed{\ldots}$ a *continuation point* which indicates that any number of diagrams (of the same or of different types) may be introduced at this place in the diagram (see also §4.3 below).

  "?" a *text insertion point*, which indicates that a sequence of characters can be introduced at the current location.

3.3 Selecting locations

If you click with the mouse in a location marked in one of the above ways, the display of the screen will alter, highlighting buttons naming diagram types, to indicate which types area available at the location you have indicated, or *selected*. In the rest of this document, we'll refer to a part of a diagram picked out in this way as the *active location*. The current active location is indicated by the use of a colour different to the background. If you click again in the same subpart of the diagram, the selection is *extended* to the immediately containing diagram. Incidentally, this behaviour provides a way of seeing how the diagram class combines smaller diagrams into larger ones.

3.4 Choosing a diagram type

Once you have chosen an location, clicking on a button in the control panel results in the insertion of a diagram of the corresponding type. The diagram that this action introduces is then made the current active location. We refer to the process of introducing material into a diagram as *instantiation*.

If the location you add a diagram to is a continuation point, a new continuation point will be introduced to the right of or below that location. If a location is already instantiated, the existing material is copied, and made available for later yank operations.[7]

[7] In fact, the behaviour is rather more complex than this. If the type you select for a location is the same as some previous instantiation of that location, the previous diagram will be restored. This is interesting behaviour, as it shows a very limited kind of type inference, but this behaviour may not be maintained in later release.

3.5 Inserting text

Some diagram types contain subparts consisting just of a sequence of characters. When such a subpart is selected, the area in the control panel labelled Enter text: is activated. If characters have already been supplied, they will appear in the box. Typing return will insert any characters you supply into the diagram.

Note that there are currently limitations on the range of characters that may be used. See problem ¶3b.

4 The Menu Bar

The menu bar will allow a number of operations. In some cases, functionality corresponding to some menu items is not yet in place. These cases are noted below. A message is reported to the user if an unimplemented option is chosen.

4.1 File

- Close

  The current window is closed, with a warning if the current diagram is modified. If the last window is closed, the editor will exit.

- New (not yet implemented)

  A new window is created using the current diagram class.

- Save

  A representation of the current diagram is saved to the current file. See also §4.2.2.

- Save as

  Prompts for a file in which to save the current diagram. See also §4.2.2.

- Create DTD

  Prompts for a file in which to save an SGML document type definition for the current diagram class. (See problem ¶7a.)

- Create Script

  Prompts for a file in which to save the current diagram as a script. See §5.3.

- Back end program

  Configure the automatic generation of files. See §4.2.1 below.

- Create PostScript

  Prompts for a file in which to save a printable version of the diagram. (See problems ¶2i and ¶2i before attempting to include generated PostScript within a docuement.)

4.2 Edit

- Copy

  Material at the active location is copied for later use in a Yank command.

- Kill

  Whenever part of a diagram is selected, it may be deleted by pressing the Kill button. The most recently deleted material is stored and can be inserted in another diagram using the Yank operation.

- Yank

  This operation is available whenever the most recently deleted material is of the right type to insert at the current active location.

- Insert

  This operation is available when the current active location is part of a diagram where any number of diagrams (of the same type or of different types) may appear. (In other words, there is a continuation point at the same level as the active location.) The operation introduces a new diagram to the left of the active location. See problem ¶1b for a discussion of situations when inserting a diagram can be confusing.

4.2.1 Back end programs

Thistle may be configured so as to produce alternative versions of data automatically, or to invoke programs over data stored in files. The option Back end program displays a dialog box with the options Activate (or OK if the back end program is already activated), Deactivate (or OK if the back end program is nor already activated), or cancel to leave the state of the program unaffected.

The first of these options request that a program is invoked each time data is saved, via a Save or a Save as command. The second revokes any such request.

A program is specified as a Bourne shell command line. The following interpolations are made:

`%f` the name of the file under which the data was saved

`%F` the path of the file under which the data was saved

`%b` the name of the file under which the data was saved, less any extension immediately preceded by '.'

`%B` the path of the file under which the data was saved, less any extension immediately preceded by '.'

`%c` the name of the diagram class file for the document

`%C` the path to the diagram class file for the document

`%%` a single `%` is interpolated.

Here are a selection of useful program specifications:[8]

- `nsgmls -s -c ${THISTLE_HOME}/lib/catalog %f > /dev/null`

  The data is validated as SGML using James Clark's SGML parser [1].

- ```
  mknsg -x -c ${THISTLE_HOME}/lib/catalog %f | \
       sgrpg -f rewrite.nsg > %b.out.sgml
  ```

  The data is normalized (as in normalized SGML, [2, 3].), transformed according to the rules in `rewrite.nsg` and stored in a file which replaces any final extension in the original file name with `.out.sgml`.

- (Not yet implemented) `thistle2script -c %c %f > %b.script` An up-to-date version of the script for this diagram is maintained.

Note that, as the back end program is only invoked on saving, you may need to perform an explicit save (either to the original file, or to a temporary file) in order to produce a new representation of the data. Note further that back end programs may not interact with standard error or standard output. (See also problem ¶9.) The precise form of the program invoked is reported. You may need to kill the program if Thistle hangs.

[8] These programs take advantage of the fact that Thistle sets the environment variable `THISTLE_HOME` on startup.
  ```

4.2.2 Backing up and checkpointing

If a diagram is saved under the name of an existing file, a back up copy of the file with a name suffixed with '~' will be made. Note that back up copies are *not* made of DTD, script or Postscript files. Note also that back up copies are made by moving, rather than copying, the original file.

Files are checkpointed every five moves, where a *move* is an editing step that affects the content of a diagram. (In other words, the act of changing selections or hiding ellipses doesn't count as a move.) If the current diagram is associated with a file, the diagram is saved under the name of the current file prefixed with '#'. Otherwise, the diagram is saved to a file called '`#thistle.sgml`' in the user's home directory.

4.3 Options

The options menu currently has to do with the presentation of diagrams. There are two kinds of option. The first allows one to *fold elements* with the consequent suppression of substructure in a diagram. A marker is left to indicate this fact. Second, in *hiding* diagrams, some part of the diagram is completely removed. Currently, only continuation points may be hidden. There are four specific choices:

- Hide continuation points (Hide ...)

  The locations at which further subdiagrams could be added may be hidden by selecting the option Hide ... in the control panel. (See also problems ¶6e and ¶2d below.)

- Fold the selected element (Fold selection)

  The selected element is removed from the display and replaced by a marker (currently a blue dot), to indicate the suppression of structure. Folded structure may be revealed (or *unfolded*) by clicking on the dot.

- Unfold all currently folded elements (Unfold all)

- Fold all elements in the diagram (Fold all)

- Fold elements in the diagram according to their type (Fold by type)

  In the last case, a list of types defined in the current diagram class is presented, together with the options Apply and Done. Selecting one or more types and pressing Apply will result in all diagrams of that type or those types being folded.

- Suppress (or allow) highlighted elements within PostScript files (Suppress highlighting in printing).

  This option determines whether generated PostScript files retain the highlighting denoting the selected location.

5 File formats

5.1 Preliminary definitions

We use the term *diagram class* to refer to the set of all diagrams which the editor can be be used to generate. A diagram class *specification* is, in effect, a grammar for diagrams.

The following characters may appear in *symbols*: `a-z0-9_`. A *symbol* is a sequence of characters beginning with an alphabetic. A *string* is a sequence of characters surrounded by double quotes. A variable name is a symbol, and so is a type name. Note that upper case characters are not allowed to appear within symbols.

A term consists of a symbol and zero or more *arguments*. Arguments are surrounded by parentheses, and separated by commas. List are surrounded by square brackets, and their elements separated by commas.

5.2 Diagram class specifications

A diagram class specification consists of a collection of *diagram specifications* terms, and of *diagram unions* terms, terminated by the symbol `start`, and possibly followed by a *script* (see §5.3).[9] We recommend the use of the extension `.spec` to indentify files containing specifications.

5.2.1 Specification identification

The following terms may be used to supply information about the class of diagrams in use:

- `diagram_class(`*`description`*`, `*`version`*`)`

  where *`description`* and *`version`* are strings, the first giving a brief description of the class of diagrams captured by the specification. The second is the print form of a floating point number (for example `"1.0"`), indicating the current version number of this specification (see also problem ¶7b).

- `dtd_name(`*`string`*`)`

  `string` will be used as the name of the external entity referred to in the "doctype" specification of Thistle files. (See also §5.4 below.)

If you create a diagram class, please check the relevant Web page[10] for potential name clashes. If you modify a diagram class specification in a way which results in previously created diagrams no longer fall within the class (and this will typically happen through removing or introducing intermediate structure in diagram content), you should give the new specification a distinct name.

5.2.2 Diagram specifications

Diagram specifications indicate the subparts of a particular type of diagram and their layout. They have the concrete syntax:

`diagram_spec(`*`TypeName`*`, `*`Contents`*`)`

where *`TypeName`* is a symbol and *`Contents`* is a term built from the graphical primitives detailed in §A. For a given *`TypeName`*, there must be exactly one specification. There must be a specification for the diagram type with the name `top`.

5.2.3 Diagram unions

A diagram union is given using the term

`diagram_union(`*`UnionName`*`, `*`Contents`*`)`

where *`UnionName`* is a symbol and *`Contents`* is a list of type or union names. For a given *`UnionName`*, there must be only one union. A diagram union must have a unique set of contents.[11]

---

[9] The inclusion of scripts within specifications is not encouraged. You should only do this if you are certain that you will not want to use the specification for displaying more than one diagram.

[10] http://www.ltg.ed.ac.uk/software/thistle/home/classes.html

[11] From the above discussion, it is clear that, from the point of view of *the arrangement of types within diagrams*, the grammar consists of statements in one of two forms:

- $T \rightarrow C_1 \ldots C_n$
- $U \rightarrow C_1 | \ldots | C_m$

where T and U are diagram types (and have a unique rewrite). U is also a union. C_i a type or a union, or a type or union annotated with '*' (Kleene star) to indicate that any number of diagrams of that type (or union) may appear. A type will be a terminal symbol if its specification contains no other types. It is also clear that any context free grammar may be represented in these terms: if there is a nonterminal T which occurs in the LHS of more than one rule,

- replace each occurrence with new unique symbol $T_1, \ldots T_j$
- add the rule $U \rightarrow T_1 | \ldots | T_j$ where U is a unique new symbol
- replace all occurrences of T on the RHS of rules with U

The result of this transformation conforms to the constraints stated above. So, any context free grammar has an equivalent formulation respecting our constraints.

5.2.4 Variable subparts of diagrams

Subparts of diagrams which may vary are described using `var` elements, whose syntax and interpretation are given fully in §A.1.

5.3 Scripts

A *script* is a sequence of *instantiation commands* or *directives*. Instantiation commands make use of *paths* through a diagram, where a path is a list of symbols, possibly ending in a string, and where the content of the list can be grouped into pairs or triples according to the following rules:

variable name followed by type name
variable name followed by integer followed by type name
variable name followed by string

In the first case, the subpart of the diagram referred to by the variable is stated to have the given type. In the second case, assuming i to represent the integer in question, the ith diagram in the subpart of the overall diagram to which the variable refers is stated to have the given type. In the last case, the part of the diagram referred to by the variable is assigned the content of the string.

We recommend that scripts are stored in files with the extension `.script`.

5.3.1 Instantiation commands

There are two instantiation commands:

`set(`*path*`)`	the diagram is assigned the types given in path.
`clear(`*path*`)`	the part of the diagram referred to by path is replaced by the most general diagram possible at that location. As a special case, `clear()` deletes the entire diagram.

As an concrete example, the following is a path taken using the diagram class `hpsg94.spec`:

```
set([tops, 0, tree,
        mother, alias_and_feature_list,
            feature_list, 0, plain_sort,
                sort, "inf"])
```

says, starting from diagram type `top` (as all paths do), the first position in the repeating location `tops` has type `tree`; within that type, the `mother` is of type `alias_and_feature_list`; ...; within that type, the string value of the label `sort` is the sequence of characters `inf`.

5.3.2 Directives

There are three directives:

`show()`	continuation points are shown
`hide()`	continuation points are suppressed
`sleep(`*integer*`)`	the system waits for the given number of milliseconds. (See problem ¶2g.)
`pause()`	(`ldplayer` only) wait for next user input.

On creating a script, the state of hiding is checked,[12] and a directive inserted at the end of the script file to reflect that state.

[12] or rather the state of hiding of the largest diagram is checked, see problem ¶2d.

5.4 SGML

SGML is used for storing the content of diagrams. The files generated by Thistle conform to a document type definition that is derived from the diagram class specification. The SGML files so created rely on an "SGML declaration" which, as far as the length of element names and the characters available within such names go, is considerably more liberal than typical SGML declarations. Accordingly, SGML processors will in general have to be informed of properties of the SGML files created by Thistle. This can be done by means of the Thistle SGML declaration, available via the World-wide web[13]. Consult the documentation for your SGML processor to determine how to use this declaration in your application.

The reason for this is that the SGML created by the editor involves the creation of (by SGML standards) long tags, so that we get maximum leverage from SGML parsers. For each variable v in each type t, the corresponding SGML tag is the string v_x_t. So the total string length of the names of variable and type plus 3 gives the length of the corresponding tag for that variable.

We attempt to be liberal in reading previously created files. In particular, if a file can be read without error, we assume that the diagram in question indeed and intentionally falls within the class captured by the specification. This provides a modicum of flexibility in transferring diagrams from one class to another, and in coping with changes to specification. The system is, arguably, too liberal in other ways in interpreting SGML files. See problem ¶4 for details.

6 The Thistle Applet

The basic Thistle applet is called `LDApplet`. This is essentially just a drawing area with no facilities for scrolling. There is an applet for the playback of multiple diagrams called `Player` described in §7.2.

6.1 Installation

You can install the code to run the applet version of Thistle by downloading an appropriate set of class files (e.g. those contained in the standard code directory[14]), or by downloading the complete Thistle package from the Thistle front page[15]. (Completion of a license agreement is required in the latter case.)

6.2 Usage

In addition, to the obligatory applet parameters of `width` and `height`, a diagram class specification file is provided via the parameter `class`, and a script file via the parameter `script`. The parameter `enableEditor` with an arbitrary value indicates that an editor window may be created for the displayed diagram.

7 Playing diagrams

There are two methods for displaying diagrams with changing content. the first an applet (§7.1) allowing a diagram with changing content to appear on a Web page, the second a standalone program `ldplayer` (§7.2) available with the Thistle release. Both instances of this program allow the user to control which diagram is displayed and to move forward and backward through a collection of diagrams.

7.1 The Player Applet

The Player Applet presents a scrollable window containing one or more diagrams, and a set of control buttons for moving forward or backward through a sequence of diagrams or to move to one or other of the extremes. These controls are labelled first, next, previous and last.

[13] http://www.ltg.ed.ac.uk/software/thistle/home/lib/sgmldecl.this
[14] http://www.ltg.ed.ac.uk/software/thistle/code
[15] http://www.ltg.ed.ac.uk/software/thistle

When the applet is started, the script file, which is most likely to be a document served over the Internet, is read at the same time as the drawing area is displayed. Details of the format of scripts for players are given in §7.3 below.

Each time a diagram is received, this fact is reported through the status bar of a browser such as Netscape. Movement through the sequence of diagrams is possible even while not all diagrams have been served to the applet. Options available via control buttons are updated dynamically to reflect the number of diagrams currently available. This means that the interpretation of the last button is dependent on the diagrams received so far. Note that keyboard control is also possible, by using the tab or arrow keys to move from one control button to another, and the space bar to "press" a selected button.

Each time a diagram is displayed, its location in the sequence of diagrams is reported in the format

showing diagram *i* of *j*

to indicate that this is the ith diagram of a sequence of length j or greater: if the words or more are appended, this indicates that the connection to the diagram source is still open and that potentially more diagrams may be received and presented.

On displaying a diagram, the scroll bars on the window are adjusted so that the scrolling area is limited to the size of the diagram.

7.2 `ldplayer`

The overall appearance of the standalone player is similar to that of the applet. Two additional control buttons appear:

save save the currently displayed diagram to a file (in SGML format) and

quit close down the player. Confirmation is sought before exiting.

7.3 Scripts for players

Scripts for players must commence with at least two empty lines. In other respects, they are simply instances of scripts as described in §5.3.

A player's interpretation of a script is determined by the location of *pause points*, which correspond to places where it makes sense to assume a diagram is ready for viewing. A pause point is any instance of a sleep, pause, hide or clear command. The next diagram is deemed to start from any following command. An implicit clear command is performed at the start of a script.

In moving backward through a sequence of diagrams, the beginning of a target diagram may be deemed to be earlier than the last pause point, if that pause point is not the result of a clear command. In that case, the beginning of the diagram is assumed to be the most recent diagram preceded immediately by a clear command, and the content of each intervening diagram is added in turn before that of the requested diagram. (This is necessary in order that diagram content is not lost in a reverse-order viewing of a sequence of diagrams. Note that equivalent tracking of the state of hiding of continuation points is *not* performed.)

7.3.1 Usage

In general, the same options are available as for the command `thistle` (see §3.1), and the interpretation of the environment variable `SPECSPATHS` is also the same. No editing file may be supplied, and a script file must be supplied. As a special case, a script file argument of `-` requests that diagrams are read from standard input. Note that it is not possible to use a script file with the name `-`.

8 Credits and other acknowledgements

The editor was designed by Jo Calder and implemented by Richard Tobin and Jo Calder. Generation of PostScript makes use of code written by E.J. Friedman-Hill[16] of Sandia National Labs, with minor modifications.

[16] http://herzberg.ca.sandia.gov/jlpr/

Regular expression matching facilities are made available using The *OROMatcher* System from ORO Inc.[17].

The author would like to thank the following people for comments, criticisms, suggestions or assistance: Chris Brew, Richard Cox, Riccardo Grazioli, Claire Grover, Mark Johnson, Frank Keller, David MacKelvie, Colin Matheson, Andrei Mikheev, Henry Thompson, Richard Tobin, Roberto Zamparelli. None of them should be held liable for any shortcomings in the system.

A Graphical primitives

Table 1 lists all of the graphical primitives. The designation '*' indicates that the primitive is not yet implemented. Any term constructed from these primitives can appear as the contents of a diagram specification. Integer units refer to dimensions in pixels. With the exception of `var`, if the arity of a term may be greater than 1, the arguments are supplied in a list.

Options appear within a list as the first argument to a graphical primitive term. The different kinds of options are listed below. The result of supplying more than one option of a particular kind is undefined.

font* The option `smaller()` (or `larger()`) decrease (or increase respectively) the size of characters.

align This option specifies the relative placement for items within a larger unit. `align(top)` in an hbox will set the elements in the box with their tops aligned, similarly for `centre` (or `center`), and `bottom`. For elements occupying no more than one "row" of vertical space, `baseline` will align elements at bottom, except that account is taken of any descent specified by the `baseline` option.

For vboxes, the choices available are `left`, `right` and `centre` (or `center`) with the obvious interpretation.

base* The option `baseline(`*`integer`*`)` specifies that allowance should be made for the descent of the current element below the baseline.

delimiter The option `delimiter(curly)` will use curly brackets to surround the enclosed element. Other choices are `square`, `angle` and `round`. (See problem ¶3e.)

dimension (including "height") The terms `width(`*`integer`*`)` and `height(`*`integer`*`)` may indicate the absolute (perhaps minimum) size of the graphic in question.

expand The option `expand()` causes the element in question to be increased in size to occupy as much space, either horizontal or vertical, as possible.

pad The option `padding(`*`space`*`)` specifies the amount of space to be left at either side of the elements contained in a hbox, at top and bottom of a vbox or around the enclosed element in a box.

sep The option `separator(`*`term`*`)` indicates an element to appear between each adjacent pair of elements.

`term` may be an arbitrary term constructed from the available primitives.

tree The options `align_leaves` and `invert` may be used. The first requests that all leaf elements (i.e. those furthest from the root node of the tree) are set on the same horizontal line. The second requests that mother nodes appear below any daughter nodes.

dendro The option `invert` reflects the geometry of the tree on the x-axis. The align options of `top`, `centre` (or `center`) and `bottom` indicates where the mother node should appear relative to daughters.

[17] http://www.oroinc.com/

type	*options*	*arity*	*description*
`array_element`	align	2	the arguments are aligned horizontally, with the second left-aligned with any other second arguments to `array_element` within an immediately containing `vbox`.
`box`	shape, pad	1	the argument is set within an enclosing shape
`bracket`	delimiter	1	the argument is set within an enclosing pair of delimiting brackets, parens, *et cetera*. (See also problem ¶3e.)
`cbox`	pad	2	the first argument is set inside a box enclosed by a second box; the second argument appears in the larger box.
`hbox`	align, expand, sep, base, pad	$\geq$0	the arguments are disposed horizontally.
`italic`	font	1	the string argument is set in italic font.
`line`	expand	0	a line, horizontal or vertical depending on context, is set.
`nothing`	none	0	nothing is set.
`plain`	font	1	the string argument is set upright.
`bold`	font	1	the string argument is set upright and bold.
`smallcaps`	font	1	the string argument is set in small capitals.
`space`	expand, dimensions	0	a space is set.
`stree`	height	1	a triangle is set above the argument.
`symbol*`	font	1	See problem 3e.
`tree`	tree	$\geq$2	the first element is set as the mother, and the remainder as the daughters, of a tree.
`dendro`	dendro	≥ 2	a tree with increasing depth shown as left-right traversal is shown, with the first element as the mother and the remainder as daughters.
`fraction`	none	3	The arguments are elements that appear wrt the fraction line i) above, ii) below and iii) to the right. The third element may occupy no space (i.e. is `nothing()`).
`var`	none	2	see below
`vbox`	align, expand, sep, pad	$\geq$0	the arguments are disposed vertically.

Table 1: Graphical primitives

A.1 `var` elements

Variable subparts of a diagram are denoted by occurrences of the term `var(`*`name`*`, `*`type`*`)`, where *`name`* is a symbol and *`type`* is either a symbol, or a symbol appearing as the single element in a list. In the first case, a single diagram of the stated type or, if *`name`* refers to a diagram union, of a type in that union, is allowed to appear in the relevant position of the diagram. The second, which can only occur in the context of a graphical primitive which allows an unlimited number of elements, indicates that any number of diagrams of the stated type (or in the stated union) may appear in the relevant part of the diagram.

A.1.1 Text elements

In the case of `var` terms appearing as the argument to a font primitive, there are two options for *`type`*. A string containing a Perl5-style regular expression may be given (See the Perl documentation, available in many forms on the web[18]. In this case any string supplied by the user has to match the given regular expression. If the supplied string doesn't match the given regular expression a message is printed in the message area, indicating failure to match and the pattern is shown (although often this pattern will be inscrutable). The symbol `Text` may also be given and this is equivalent to the regular expression `".*"`, i.e. any sequence of characters excluding newline. (See problem ¶3c.)

B Known problems, limitations, shortcomings, etc.

This list also functions as a list of planned enhancements. The symbol "**" indicates what we view as urgent or highly desirable.

1. Diagram specifications and processing of diagram content

   (a) Only one element within each diagram type may contain text. In practice, this is not a major limitation, as the grammar for diagram classes can simply be rewritten so as to conform to this restriction. It does mean that server programs may have to remove additional layers of structure.

   (b) On inserting a diagram (via Insert), possible instantiations of the new insertion point are not checked, and so no instantiation is performed even if there is only a single type of diagram permitted at that location. Consequently, *two* continuation points appear (one is the inserted new diagram, the other at the 'end' of the diagram). If the user then asks for instantiation, an additional continuation point is shown at the 'end' of the containing vbox, hbox or tree. These superfluous boxes may be removed by toggling the state of hiding of continuation points. (See also problem ¶11).

2. Typography, layout and other aspects of display

   (a) If the first subpart (reading from top to bottom or left to right) of a diagram type only requires text to complete it, insertion of a diagram of that type will result in the whole new subpart being selected and the text box activated at the same time. One example of this is the relation symbol in an atomic condition in DRSs[19]. This behaviour is at worst a little confusing, but should be corrected once we remove limitation 1a.

   (b) Subtrees drawn with the option `align_leaves` may occupy less space than stated in their bounding boxes. Consequently, highlighting a subtree in an unbalanced tree may result in the highlighting straying on to branches associated with other subtrees. Trees without daughters are incorrectly drawn.

[18] http://reference.perl.com/query.cgi?regexp
[19] http://www.ltg.ed.ac.uk/software/thistle/demos/example4.html

(c) The size of the drawing area (i.e. the area which can be scrolled) is initially set to have dimensions six times larger than the visible area. This size is computed on start up, and *prior* to computing the size of the diagram. In the case of very large diagrams, it may not be possible to scroll completely through the diagram. On resizing the window, dimensions are recomputed, and if the above formula doesn't allow enough room for the whole diagram to be viewed, the dimensions of the scrollable area are increased so as to be able to accommodate the whole diagram. In summary, then, resizing the window will guarantee that the entire diagram may be viewed. (This behaviour is likely to change so that the user doesn't have to force resizing.)

(d) Hiding of ellipses is in fact a property of (sub)diagrams. Consequently, if one hides ellipses and kills a part of a diagram, that part is still deemed to have ellipses hidden. If you now show ellipses again, and yank the killed diagram back again, the yanked diagram will not show the ellipses. (The converse situation can also arise.) We have not yet determined the most appropriate behaviour for Thistle in this instance. See also the comments in footnote 7.

(e) RHS elements within an array are sometimes not selectable. (In example diagram hpsg.5, do Fold all and then attempt to unfold by clicking working top-to-bottom and then left-to-right). This seems to happen when some LHS element in an array is less wide than some preceding LHS. .

(f) In playing a sequence of diagrams, the status of hiding is not tracked. We recommend in consequence that scripts set the hiding state explicitly at the end of each collection of instantiation commands.

(g) Real-time animated diagrams may not update correctly when running under a Web browser. (It looks like double buffering will be necessary here, to prevent the browser getting hold of incomplete images.) . Under Netscape, at least, waggling the mouse in and out of the space reserved for the Applet will force updates of the screen. The new programs `ldplayer` and corresponding Applet (see §7) are a useful workaround where realtime animation is not required.

(h) The system may not work satisfactorily on monochrome systems.

(i) Calculation of PostScript bounding boxes is an estimate and not always correct. PostScript diagrams contain a fair amount of whitespace surrounding the graphic. If including a graphic within a LaTeX file (using, say, the graphics or graphicx package), be sure to crop the graphic to its bounding box (e.g. use the starred form of the command.) All of the PostScript files used in this document have been edited *post hoc* to produce a tighter bounding box. Incorrect PostScript files will be created if the backslash character appears within a text field.

(j) The graphic primitive `smallcaps` doesn't use a proper small caps font.

3. File formats, character sets, etc.

(a) General
Variable names may not contain the string `_x_`.

(b) The character encoding is not specified.

(c) The character `"` cannot be specified within a regular expression. The only way it can be required to appear within a diagram is by creating a diagram type containing the entity corresponding to that character. For example:

```
diagram_spec(quote,
     plain("The following is the double quote: &quot;"))
```

(d) Constraints on text elements given by regular expression are only checked when the element is supplied by the user, and not when reading a diagram file or a script. The latter facility ought to be available.

(e) Symbol font (or some equivalent array of appropriate characters) is not implemented in the versions of Java we are currently working with. A consequence of this is that curly brackets and parentheses are not properly rendered if their contents extend over the equivalent of more than one one vertical line.

4. SGML

   The SGML reader used by the system has restrictions which means that the SGML files that can be read by the system are in fact more general than those specified by the generated DTD: any number of diagrams of appropriate types may appear. Only the last of these diagrams will be displayed by the system.

5. Platform-specific and browser-specific problems

6. The standalone version of the program has been tested only under Solaris and there is no fully functional non-UNIX version available.

   (a) Internet explorer

      i. Under Internet Explorer on Macintoshes, with all of the Java machines available to me, the menu bar is not shown on editor windows. So, most importantly, kill, copy and yank operations are not available.

   (b) Netscape under Linux

      Under earler versions of RedHat, closing editor windows may crash Netscape.

   (c) Netscape 3 under Sun's CDE.

      The height of characters is wrongly reported by the X server and diagrams are consequently incorrectly laid out.

   (d) Problems have been encountered with this software running on Macintoshes. The program has been observed to work on

      - 8500, System 7.5.5, Open Transport 1.1.2; Navigator 3.0.1
      - 6100/66, System 7.5.5, Open Transport 1.1.2; Navigator 3.0.1
      - 630, System 7.5.3, Apple Talk 58.1.5; Navigator 3.0

      Problems *have been encountered* on

      - 6100/66, System 7.5.3, Open Transport 1.1; Navigator 3.0 and 3.0.1

   (e) Under Macintoshes, some versions of Java do not allow the status of a check box to be changed.

   (f) Under Netscape 3 on some Sun SPARCstations, attempting to invoke an editor from an applet may result in the apparent death either of the applet on the Web page or of the invoked editor. This seems to be caused by a difference in thread handling in that configuration. Forcing a reload of the Web page (i.e. SHIFT+RELOAD) seems to clear the problem.

7. Other system issues.

   (a) Sensible file names are not supplied in dialog boxes. This is a particular problem in the case of DTD names as they have to be synchronized with the DTD names used in files.

   (b) Version numbers of specifications are not checked.

8. Interrupts received from the controlling terminal will cause the program to exit even if there are unsaved changes.

9. Any attempt by a back end program to use standard error or standard out will result in Thistle hanging. Back end programs may not run asynchronously. Under most UNIX systems, the standard error may be redirected to the "controlling terminal" by means of a construct such as `2> /dev/tty`.

10. There is no way to associate a back end program specification with a diagram, or to provide such a specification via the command line.

11. Java 'out of memory' errors have been observed when manipulating large diagrams, , seemingly connected with problem ¶1b.

12. Editors spawned from an applet contain back references to the applet which are invalidated by closes (including implicit ones) of the containing web page.

13. There is no on-line help.

14. This list is incomplete.

C Plans

The following section lists enhancements we would like to add to the system.

1. System issues
   (a) Port to JDK 1.1.* which will enable us to do more sensible things with printing and Unicode.
   (b) We need a general scheme for interaction with client and server-side processes. We also need to think about the kinds of logic appropriate for constraining sets of paths.
   (c) Migrate persistence format from SGML to XML. Provide compilers for XML DTDs and XSL!
   (d) There should be a mode for Applets in which the content of nodes can be interpreted as a URL, used to direct the containing browser.

2. User interaction
   (a) Allow keyboard control over diagram instantiation; implement undo.
   (b) (by general agreement) On selecting or moving the pointer to a location, there should be the option of seeing, on a local popup menu, the diagram types available at that point.
   (c) There should be a kill ring.

3. Graphics and typography
   (a) There should be a "disappear" option to most graphics primitives, which reduces the size of the graphic to nothing when no variable subpart is instantiated. This will be most useful in controlling amounts of whitespace, but also offers a means of encoding optional elements. (The downside of this trick is that optionality is not explicitly encoded within the grammar.)
   (b) There are many aspects of typography which could quite reasonably be changed per subdiagram. One obvious case is that of font size.
   (c) A scheme for colour should be devised (although this will need to interact quite intimately with the content of diagrams).
   (d) There should be graphical primitives `table` and `table row` (and `table header`?)
   (e) LDApplets should be be smart enough to add scroll bars when diagrams exceed the width or height supplied by the Applet tag.

4. Diagram classes
   (a) Write a comprehensive grammar for LFG diagrams.
   (b) Write a Gentzen sequent system.
   (c) Upgrade existing classes to implement appropriate regexp restrictions on text content.

5. Miscellaneous
   (a) There should be some support for generating scripts including control information for the players.

C.1 Known instances of forwards incompatibility

1. Diagram classes uses regular expressions will not work under versions prior to 1.5.

C.2 Known problems with releases

C.3 1.4

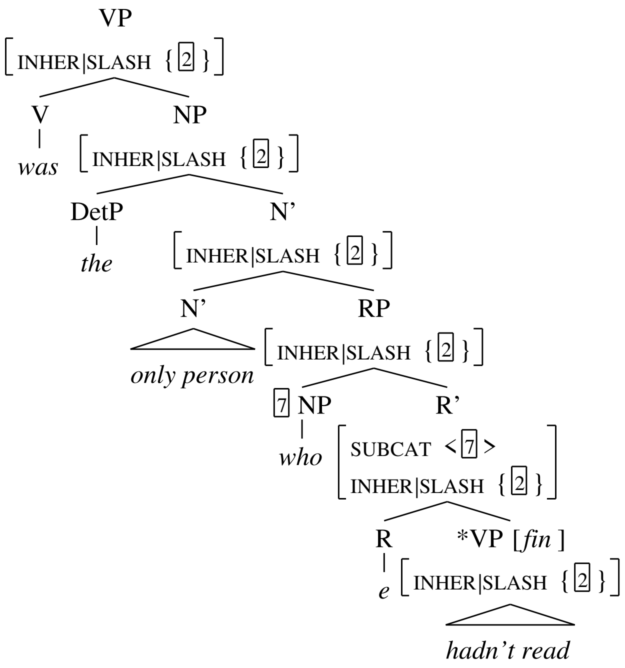
1. The file `bin/ldplayer` should be replaced with a symbolic link to the file `bin/thistle`.

D Example diagrams

All of the diagrams below use PostScript output by Thistle (but have had their bounding boxes edited *post hoc*, see problem ¶2i).

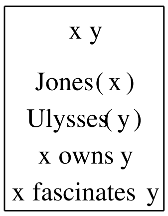
D.1 HPSG94

The diagram below is also available on-line[20] with other HPSG diagrams.



D.2 DRS (Kamp and Reyle)

The diagram below is also available on-line[21] with other DRT diagrams.

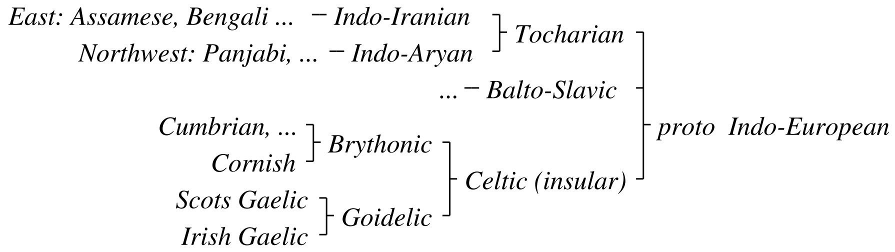


[20] http://www.ltg.ed.ac.uk/software/thistle/demos/example_hpsg.html
[21] http://www.ltg.ed.ac.uk/software/thistle/demos/example_drs.html

D.3 Dendrograms

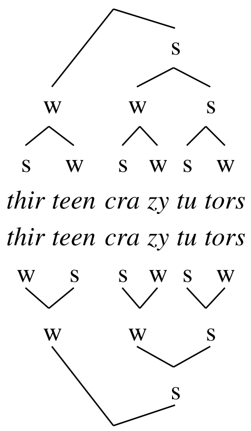
The diagram below is also available on-line[22].

The second uses the same data, but different layout.



D.4 Metrical trees

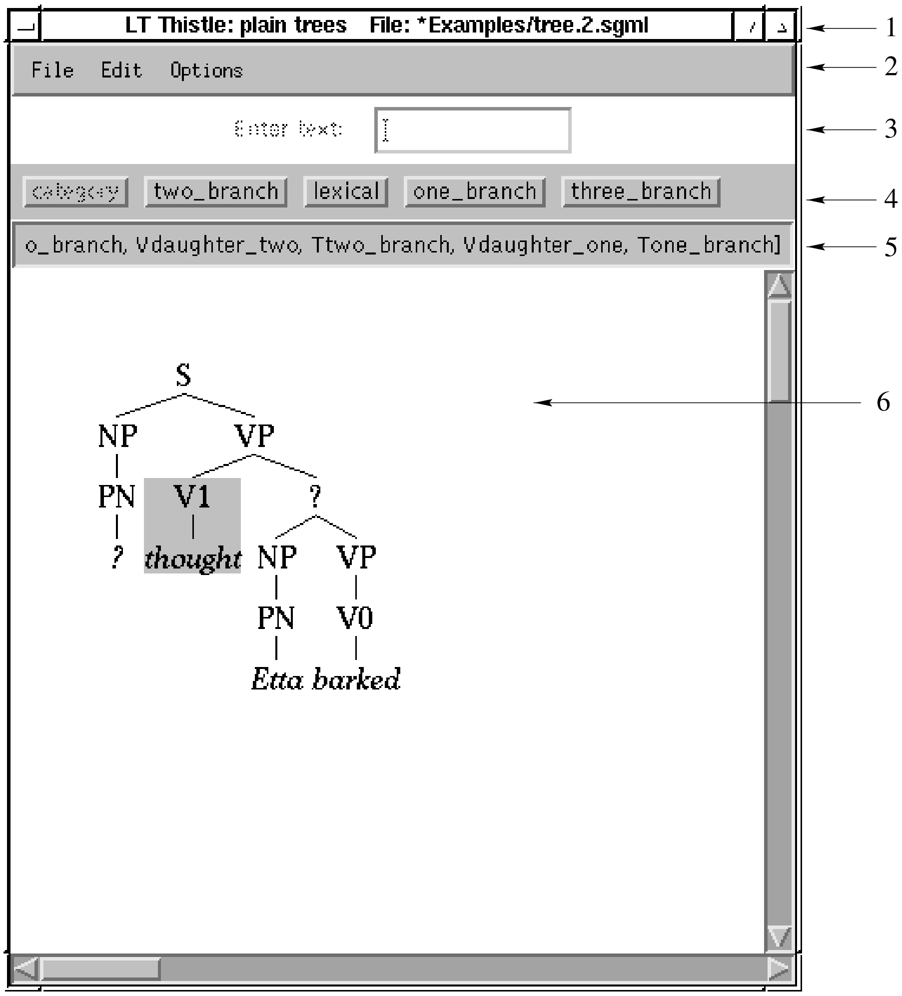
The diagram below is also available on-line[23] with others.



E Annotated screen layout

The figure directly below shows the arrangement of the elements constituting the editor:

[22] http://www.ltg.ed.ac.uk/software/thistle/demos/example_dendro.html
[23] http://www.ltg.ed.ac.uk/software/thistle/demos/example_metrical.html



The function of each of the elements is as follows:

1 The title bar indicating the diagram class in use and the name and modification status of any file associated with the diagram.

2 The menu bar (§4)

3 The text input box (§3.5)

4 The type buttons (§3.4)

5 The message area. Various messages are displayed here. In this picture, the *path* to the selected location is shown.

6 The drawing area (§3.2). Note how the selected location is displayed.

References

1. James Clark's SGML parser[24]
2. The LT NSL system[25]

[24] http://www.jclark.com/sp/index.htm
[25] http://www.ltg.ed.ac.uk/software/nsl/index.html

3. The LT XML system[26]

[26] http://www.ltg.ed.ac.uk/software/xml/index.html