TR-IT-0306

– Boardedit: A Multilingual Board Editor – From Interactive Tree Editing To Analysis by Analogy

Nicolas Auclerc Yves Lepage

September 1999

Abstract

This report describes a multilingual application called `Boardedit` designed for Natural Language Processing, and more precisely for the data preparation of some linguistic applications. `Boardedit` is a board editor. A board is the pair constituted by a text and its linguistic structure, generally a tree. Correspondences, which are an original feature of boards, are links between the linguistic tree and the sentence. In order to edit a board and build treebanks, `Boardedit` proposes simple and fast editing facilities, on texts, as well as on trees. It also integrates some searching methods, as well as analysis by analogy.

Keywords

Board, interactive tree edition, parsing aids, approximate matching, analysis by analogy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This report describes a multi-platform application called `Boardedit` designed for Natural Language Processing, and more precisely for the data preparation of some linguistic applications (*e.g.*, the analysis task for a machine translation system). `Boardedit` proposes an easy-to-use editor for constituting a treebank which will aid linguistic investigations.

Thanks to this tool, linguists will be able to type in a sentence in the same way as they do in any usual text-editor. Then, they will be able to draw a tree above it. As a final touch, they will give the correspondences between the tree they have designed and the sentence.

We give a general description of the project and explain the background of the `Boardedit` project. Then we explain the Editing Facilities implementation. And finaly we explain the different methods of the Search functions.

1.1 Generalities

1.1.1 Natural language processing

The `Boardedit` project is designed for natural language processing, *i.e.*, the application of computers to the study of language, or their application for enhancing of the technology of the computer itself. Natural language processing encompasses voice recognition, text retrieval and machine-aided translation.

We will concentrate on the part which is concerned with the study of language also known as linguistics.

1.1.2 Linguistics

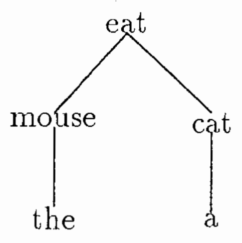
The study of language, or linguistics, was born quite recently, at the beginning of the century. The founder of modern linguistics, Ferdinand de Saussure, defined the task of linguistics as the description of what he called *signe*: the association of two things, the *signifiant* or acoustical image, and the *signifié* or concept [Saussure 16]. He insisted that both the *signifiant* and the *signifié* have their own *structures* [1].

If we draw a parallel between the *signe* and the objects NLP researchers work with, we can consider any text as a *signifiant* and any linguistic representation as a *signifié*.

1.1.3 Tree structures in linguistics

In a century of history, linguistics attained many achievements and was marked by many different streams. We will mention two of the most widespread linguistic structure representations: Dependency and Constituency structures.

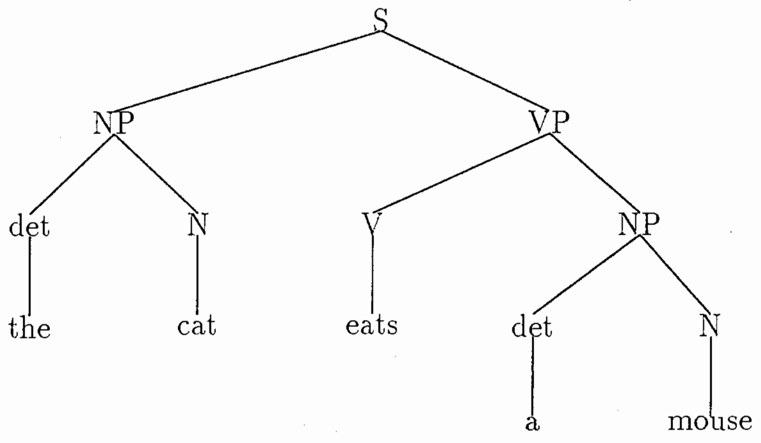
Dependency structures The structuralist approach (the school of Prague, Tesnières). We only grossly recall that linguists belonging to this trend proposed representations of sentences based on a logical interpretation, similar to Prolog terms in computer science:
the cat eats a mouse $\longrightarrow$ `eat(mouse(the),cat(a))`



[1]In a structure, in the structuralist meaning of the word, elements have no absolute existence. They exist thanks to their relationships or *oppositions* to other elements. The concept of structure gives rise to the distinction between phonology (the study of functional or opposition features of the sounds of a given language) and phonetics (the study of the articulatory or perceptive aspects of the sounds of one or many languages).

Constituency structures The Chomskyan approach. Again, we will only consider the fact that linguists from this approach interpret a sentence in the way words can be grouped together:

the cat eats a mouse → `((the),cat),(eats,((a),mouse))`



1.2 Machine translation

Historically, machine translation was the first application of computers to a task other than computing with numbers. The first generation of machine translation in the 50's (mainly in the USA and Russia) relied on the idea that translating was just a matter of word-to-word translation followed by rearrangement of the words (exchange, insertion, deletion).

The second generation of machine translation systems (mainly in France, Germany, and the USA) were designed with the belief that, before translating words, a step of understanding the structure of the sentence was necessary. This step is called analysis. Moreover, in the second generation, the linguistic models are clearly separated from the computational models.

In the ATR-MATRIX systems of ATR-ITL [http://www.itl.atr.co.jp/matrix], as in many other current machine translation systems, translation is performed in three steps: analysis, transfer and generation.

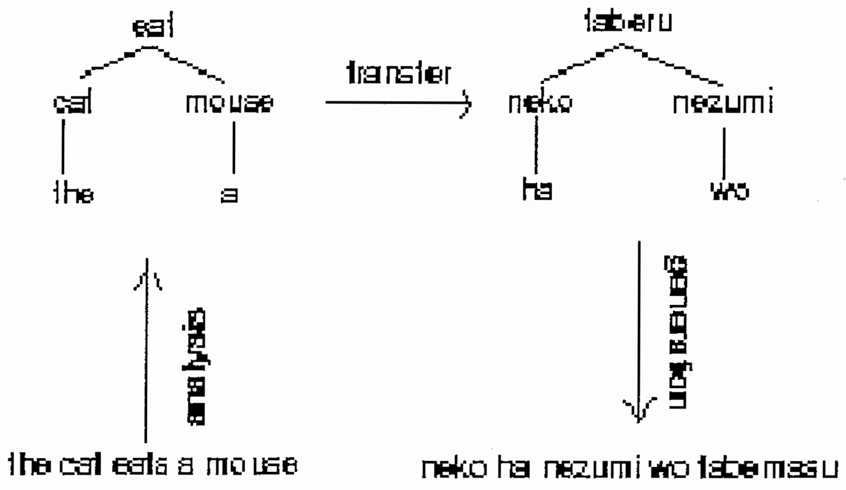


Figure 1.1: Analysis, transfer and generation process

1.2.1 Analysis

Analysis is the step of building a structure to understand the text. In many systems, this structure is of one of the forms presented above (dependency or constituency). In some systems, many structures adopting various linguistic points of view are built. In all of these systems, the phase of building a structure, which is always equivalent to a tree, is called analysis.

1.2.2 Transfer

In the second generation system, the rearrangement does not take place any longer on the level of words, but on the level of structures. This phase is called transfer.

1.2.3 Generation

Generation is somehow the reciprocal to the task of analysis (see 1.3.1). This operation starts with a linguistic tree and builds a sentence.

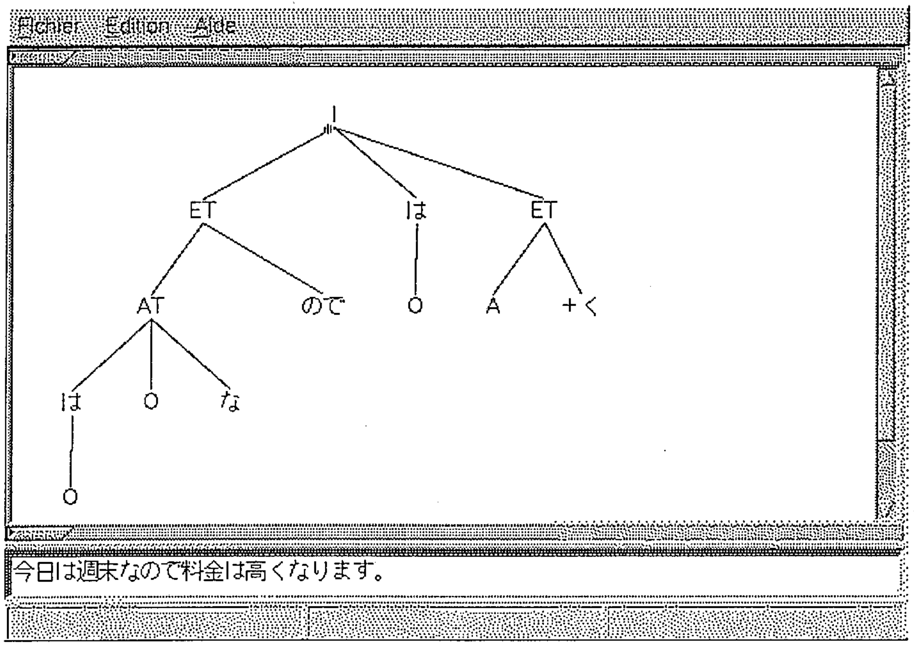
1.3 Boards

A board is the pair constituted by a text and its linguistic structure. The following two objects are commonly used in this pair:

- a text is a string, and
- a structure is a tree.

In current grammars, a grammar "element" is either a chunk of structure ($S \rightarrow NP\ VP$ or $det \rightarrow the$ are the same as *S(NP,VP)* and *det(the)* respectively) or a string pattern (in grammars inspired by Harris).

In a board, both aspects are present. In this way, the structural aspect of linguistic works can be mixed with examples. Figure 1.2 shows a sentence and its associated representation in a board.



今日は週末なので料金は高くなります。

Figure 1.2: A board: text and associated tree

Because a board can also be read from the tree to the text, it can also be used for generation (or synthesis). So a board is bi-directional. Moreover, we will see that it is also non-directional.

1.3.1 Bi-directionality

The term *non-directionality* was used by [Winograd 83] to characterize context-free grammars, because rules can be applied in both directions: analysis and generation. Nowadays, *non-directionality* has been replaced by *bi-directionality*. In fact, as [Zaharin 90] observed, the reverse operation of analysis would imply starting from the start symbol only. But, in actual NLP systems, it starts with a complete structure. So generation is not the exact reverse operation of analysis.

Many formalisms are said to be bi-directional, but often, some extensions or tricks in programming make grammars suitable only for analysis (or only for generation). In other systems, only the formalism (or the source code to adopt a programmer's point of view) is bi-directional, and a compilation delivers two different executable codes, one for analysis, and one for generation. This means that the engine (or the executable code, to follow our comparison) is not bi-directional.

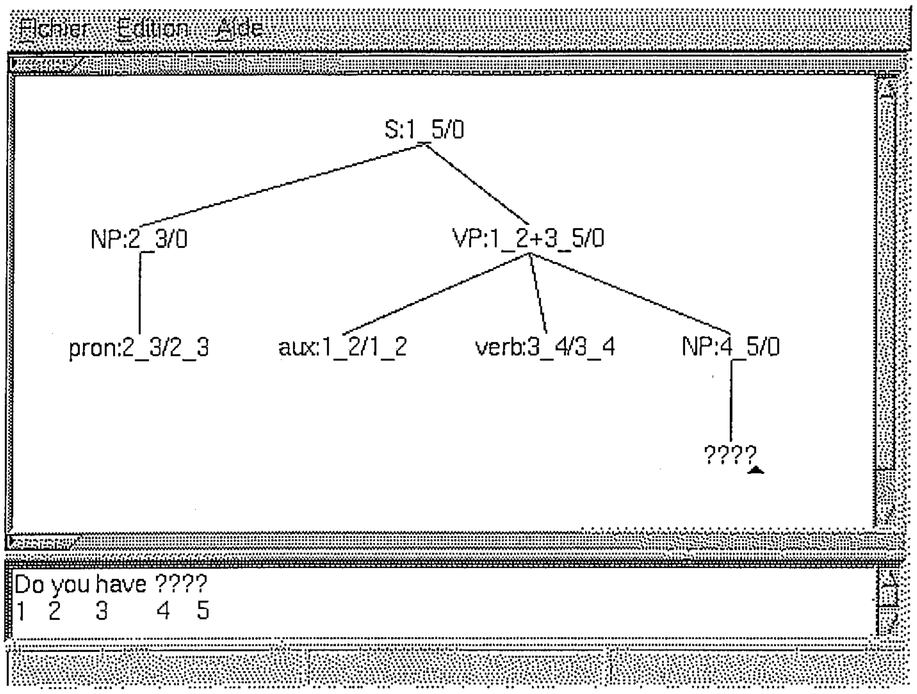
1.3.2 Non-directionality

Boards are not only basic objects in grammar, they can also be input and output objects for analysis and generation. This view of analysis and generation gives birth to a more general operation which we call non-directional completion. Bi-directionality appears to be a particular case of what we call *non-directionality*, to revive the term [Lepage 91].

Non-directional completion consists of proposing an incomplete string and an associated incomplete tree to the system. Figure 1.3 shows a non-directional completion.

The system has to deliver complete strings associated with complete trees which match the uncomplete input and which come from the grammar.

This is impossible with a compilation of grammar delivering two different specialised modules, one for analysis and one for generation. It is possible for a proposal such as the one in [Lepage 94] because the basic object is not only bi-directional but also non-directional. As a matter of fact, the general function of the system proposed in the abovementioned report is to deliver complete correspondences for partially specified correspondences.



Do you have ????
1 2 3 4 5

Figure 1.3: A board with incomplete tree for non-directional completion

1.3.3 Correspondences

There is a link between the linguistic tree and the sentence because the linguistic tree is another representation of the sentence. The relationship between the sentence and its structural representation is called *correspondences*. Correspondences are an original feature of boards.

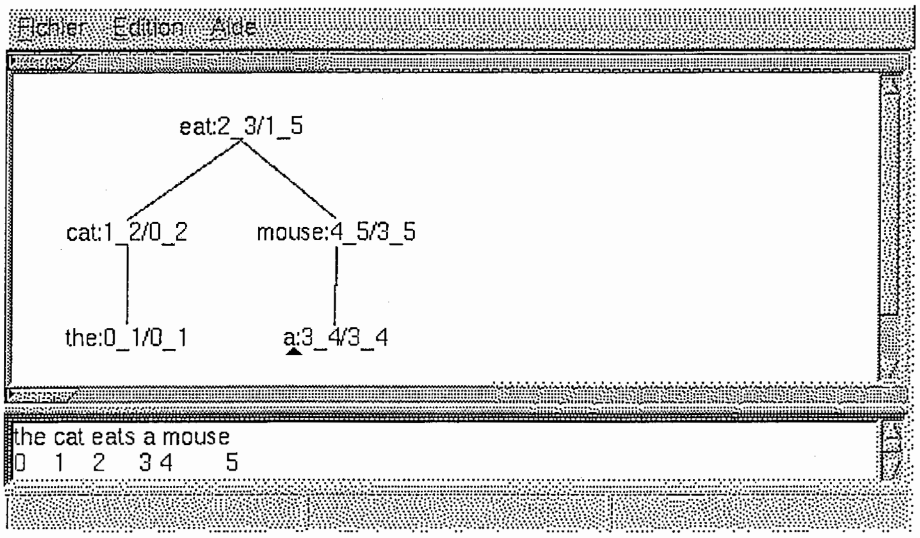
[Zaharin 87] proposed that there would be two kinds of correspondences:

- one from node to text;
- one from subtrees to texts.

Also, because in linguistic structures only some kinds of subtrees are used, he proposed that only complete subtrees (the entire tree under a given node) be considered. He called these two kinds of correspondences STREE and SNODE [Zaharin and Lepage 92]. In the figure below, the entire tree starting at the root *eat* corresponds to the entire text, hence the first interval under *eat* is 0_5, because the entire text extends from 0 to 5. But, as a node, *eat* corresponds only to the chunk of text *eats*, which spans from 2 to 3. So, the second interval mentioned under *eat* is 2_3.

The two kinds of links are thus :

- between (possibly a list of) words and nodes;
- between (possibly non-connex) substrings and complete subtrees (designated by their root).



the cat eats a mouse
0 1 2 3 4 5

[Lepage 89] more clearly defined the idea that correspondences be constrained. He claimed that three constraints (global, inclusion and membership) are enough to represent the usual kinds of linguistic structures (mainly constituency and dependency). These constraints are explained as follows:

- Global correspondences: the whole tree corresponds to the whole text. It is an error if there exists a word in the text which does not correspond to any node in the tree and vice versa. In the example above, global correspondences imply that the first interval under *eat* is 0_5 (the entire text).
- Inclusion: if a tree corresponding to text T1 is a subtree of another tree corresponding to text T2, then T1 is included in T2. This constraint implies that, for example, the text which corresponds to the

tree `cat(the)`, which is *the cat*, is included in the text corresponding to the entire tree: *the cat eats the mouse.*

- Membership: if a node corresponding to words T1 is member of a tree corresponding to text T2, then T1 are in T2. This implies, that, for example 3_4 (the second interval under *a*) which shows the word in correspondence with the node *a*, is in the first interval under *mouse*, which shows the text corresponding to the tree `mouse(a)`.

Chapter 2

Background

2.1 Objectives

A treebank is a corpus in which each sentence carries a linguistic description (in fact, a tree) input by hand by an indexer. The interest of such banks is undeniable as a linguistic resource, and some statistical approaches in analysis already use treebanks. A famous treebank for English is the Upenn treebank, which was designed at the University of Pennsylvania (Web page: `http://cis.upenn.edu`). Department 3 of ATR-ITL has been building a treebank for English known as the ATR–Lancaster treebank [Black et al. 96]. Similar efforts for Japanese are in process for constituent structures [Kawata *et al.* 98] and dependency structures [Lepage 96].

The construction of a treebank is a very cumbersome and time-consuming process. To speed up this process, we propose a tool, an editor with extra functionality. Also, the consistency of the data in treebanks is often problematic: similar portions of texts are sometimes assigned different structures. This is a particularly sensitive point if these data are to be used by statistic models. But checking the consistency of data requires an enormous amount of time.

We propose to view the construction of a treebank as a sequence of editing and searching steps, as follows [Lepage & Ando 96]:

- edit a new sentence (input it);
- find similar sentences, to retrieve their associated structures, so as to propose possible candidate linguistic structures;
- edit the new linguistic structure (build it);

- find similar structures in order to check or ensure consistency with previous data in the treebank.

This procedure has a beneficial effect: the larger the treebank, the faster and the more realiable its extension.

2.2 Aim

The aim of the project is to implement a tool, `Boardedit`, that will allow a user to build a treebank in a faster and more consistent way. This tool must be user-friendly and must allow the user to edit sentences, in various languages, as well as trees.

2.2.1 Editing

In order to speed up the creation of new data in the treebank, our tool will propose simple and fast editing facilities, on texts, as well as on trees.

Text-editing With our tool, text-editing will be carried out as it is with modern text editors. Cut, copy and paste commands will be available, as well as selection by a double or triple click of the mouse (for words and lines respectively).

Tree-editing Some viewers for trees are available for free, but they have to be ruled out for our purpose as they do not allow tree-editing. There are certainly some tools which allow tree-editing, like that of University Sains Malaysia, or xoobr (of Stern Mark), but, with these tools, tree-editing is particularly cumbersome (requiring a dialogue box to input a label, adding of new nodes only by menu, *etc.*)

With our tool, tree-editing will be as simple and direct as text-editing is. The interaction will be done directly in the tree, without the use of any dialogue boxes.

That is possible thanks to a rigorous parallel between nodes and subtrees on the one hand, and words and lines on the other hand. This parallel underlies all functions of editing on trees: any function (click, select, insert, cut, copy, paste, *etc.*) for the edition of trees will have exactly the same behaviour as in text-editing.

Not only direct selection by mouse will be possible, but also the insertion of a node will be possible by pointing the mouse directly where the node should be inserted. New branches will be automatically created.

Text-and-tree editing Correspondences will be an interesting function in our tool. They will establish links between the text and the tree so as to make explicit which part of the text corresponds to which part of the tree [Boitet and Zaharin 88]. That is of much significance for non-projective representations, like dependency structures.

2.2.2 Searching methods and parsing aids

In order to increase the consistency of new data created, our tool will contain some searching methods and parsing aids.

Pattern-matching Approximate matching refers to searching for those lines in a file that contain a substring at a distance less than or equal to a certain threshold k from a given pattern p (of length m). This distance is the minimum number of character insertions, deletions or substitutions necessary to transform the substring into the pattern.

For example, when looking for `analogy` with a threshold of 2, only the first and third lines of the following file are output.

`analogous`	dist(`analog`, `analogy`) = 1
`explanation`	(insert `y`)
`neuroanatomy`	dist(`anatomy`, `analogy`) = 2
	(substitute `l` for `t` and `g` for `m`)

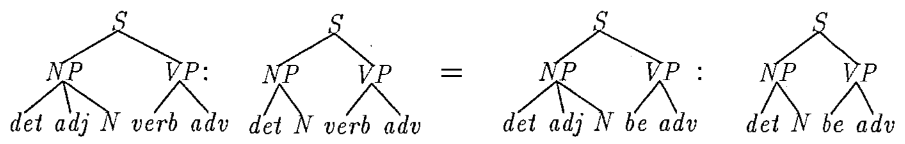
Wu and Manber [Wu & Manber 92] proposed a practical implementation of the Baeza-Yates and Gonnet method, which exhibits the behaviour of $O(nk\lceil \frac{m}{w} \rceil)$, where w is in fact some constant. *agrep* is considered the fastest practical algorithm of this kind.

However, agrep is limited in the length of the pattern (24 characters) and in the threshold (8 edit operations). Also, agrep handles only ascii characters, and we shall need 2-byte characters for Japanese. [Lepage 97] proposes a new algorithm, called Agrep, which is faster than Wu & Manber's algorithm in average when the ratio threshold/pattern length becomes greater than 0.4. Also, for multilangual pruposes, Agrep handles 2-byte (and even 4-byte) character searches.

Analysis by analogy The technique of *analysis by analogy* relies on the idea that, if there is analogy on some low representational level, for instance, that of syntactic classes:

The signal is off : *The green signal is on* = *The lamp turns on* : *The green lamp turns off*

then, there should be analogy on a higher representational level (see [Itkonen 94]), for instance, that of structural representations. For example, if we picked up the last three sentences above by searching with the first sentence, for which we want a structure, x, the tool may compute the following proposal:



Here, note the difference between the first and the third trees due to the absence of an adjectival phrase. x is different from the second tree in that *verb* has been replaced by *be*. This tree is an exact description of the sentence *the signal is off*.

2.3 Previous realisations

There are already preliminary versions of `Boardedit` but as they reflect an early stage of research, they are all either uncomplete or not finished. There are three such versions: a Mac version and two Unix versions.

Mac version The Mac version implements all the editing functions designed (see section 2.2.1), including correspondences. Nevertheless, in this version no search functionality was implemented. This tool has been implemented by Goh Chooi Ling [Goh 96] of the Universiti Sains Malaysia, as a final year project, using a collection of C functions [Lepage 92a] [Lepage 92b]. This tool is in use at Projek Terjemahan melalui Komputer (PTMK) [Tang 96].

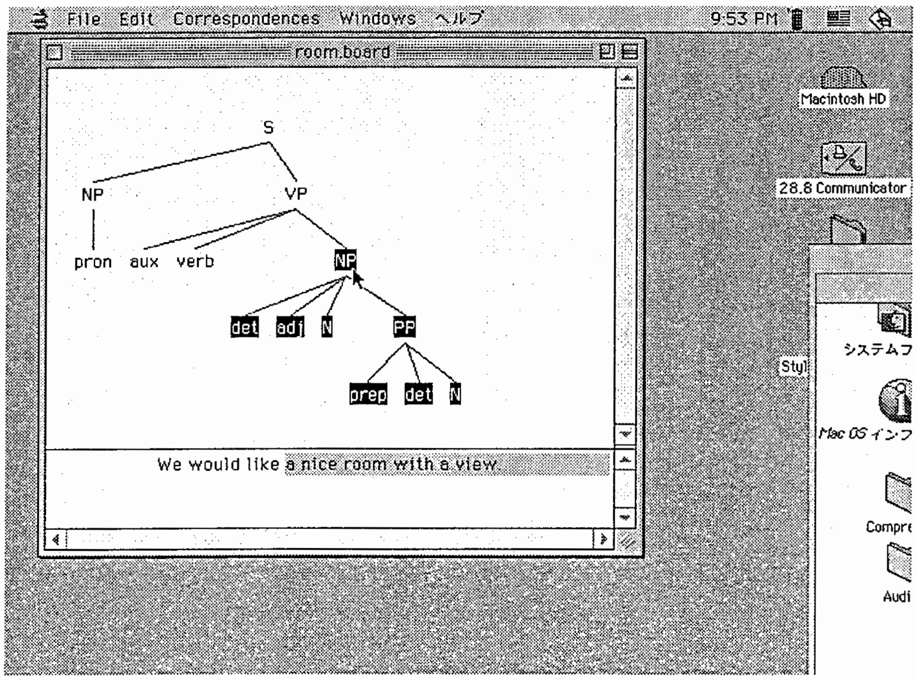


Figure 2.1: Mac version of `Boardedit`

Unix version For visualisation only, a by-product of the Unix version was `treecanvas`, an online command which automatically opens a window in which a parenthesised tree given on the standard input is displayed. For example, the tree `would(like(we,room(a,nice,with(view(a)))))` is displayed in Figure 2.2. As the basic data structure is in fact that of a forest, this tool is able to display any parenthesised forest. Under Openwindow, this tool was easily integrated under the text editor Textedit to visualise any parenthesised tree selected.

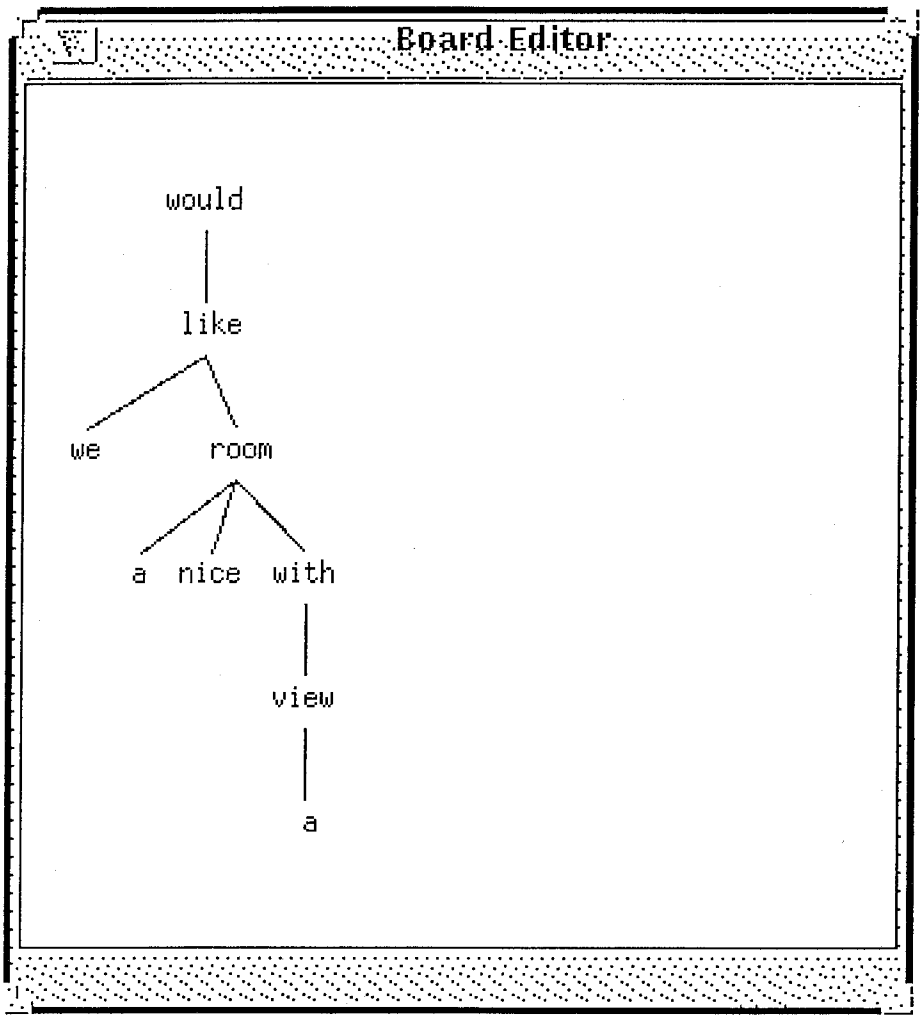
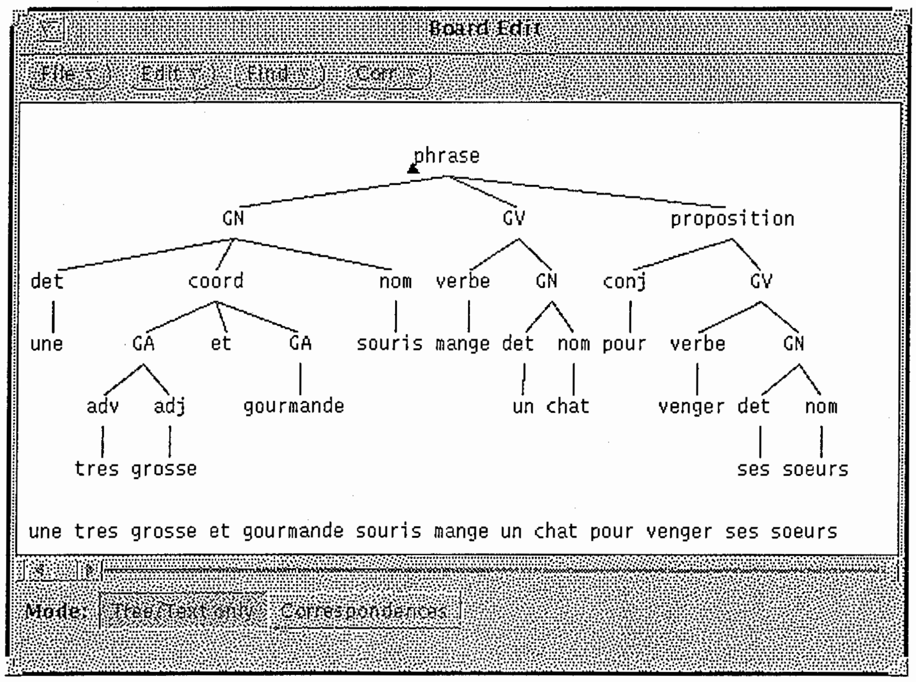


Figure 2.2: Old version of `treecanvas` (notice the wrong window title...)

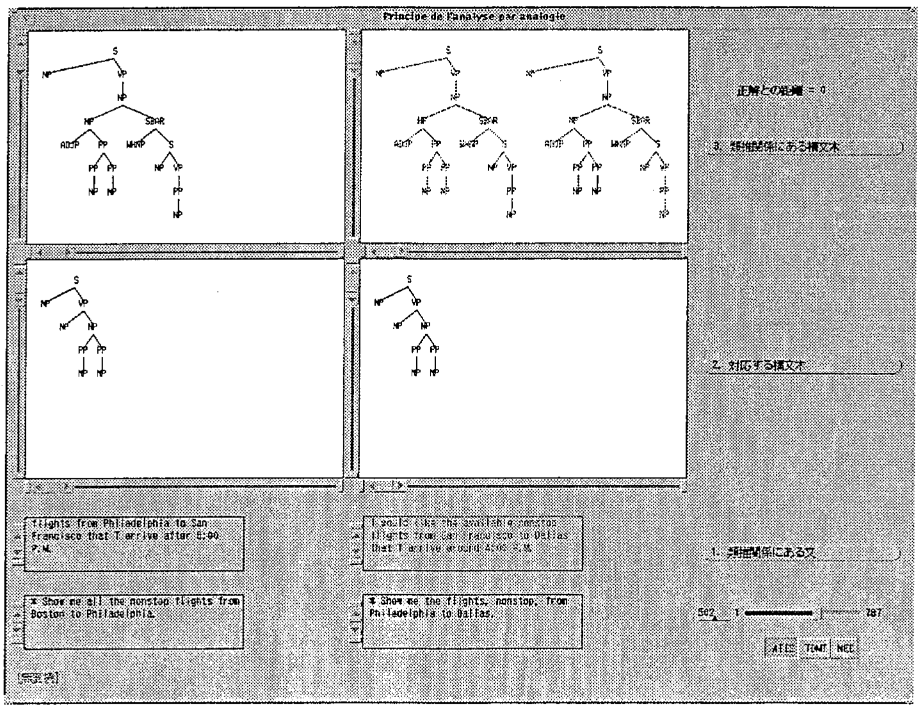
A complete version of `Boardedit` was never finished under Unix. Attempts have been implemented with either Openlook (see Figure 2.3) or with wxWindows 1.6. This resulted in multiple versions which almost always integrate the entire editing facilities. However, selection does not work in some versions; an old, slow version of approximate matching works in other versions, while correspondences partly work in still other versions. Our project aims to have a look at all these earlier versions and, most of the time, restart programming from scratch to finalise Boardedit.



une tres grosse et gourmande souris mange un chat pour venger ses soeurs

Figure 2.3: Old Unix version of Boardedit

As for the searching facilities, a demonstration of analysis by analogy was made in November 1997 during the ATR open house. For this purpose, a program was developed under Openlook (see Figure 2.4), and uses the display part for trees of Boardedit.



flights from Philadelphia to San Francisco that I arrive after 5:00 P.M.

I would like the available nonstop flights from San Francisco to Dallas that I arrive around 4:00 P.M.

* Show me all the nonstop flights from Boston to Philadelphia.

* Show me the flights, nonstop, from Philadelphia to Dallas.

Figure 2.4: A demonstration interface for analysis by analogy

2.4 Project environment

Because the GUI (Graphical User Interface) of the tool to be built in this project is required to run on several platforms, we had to find a multi platform GUIDE (Graphical User Interface development environment) which would support Unix X11, Mac Os and Microsoft Windows.

There are not many such GUIDEs and most of them are not free. As an exception, wxWindows is free. Moreover, as we saw in Section 2.3 the first attempt at implementing `Boardedit` was done with wxWindows 1.68. These considerations justified our choice of wxWindows 2.0.

Under Unix, wxWindows 2.0 relies on a low-level graphic layer, which may be either Motif 1.2 or GTK+ 1.x. A requirement for `Boardedit` is to allow the input of Japanese data. As internationalisation (i18n) is better supported under GTK+ 1.x than under Motif 1.2, we chose GTK+. Also GTK+ is free. Moreover, the resource editor for wxWindows, wxDialog, is not yet available for Motif.

YLlib is the collection of C functions already mentioned for the Mac version of `Boardedit`.

The overall structure of our application is shown in Figure 2.5.

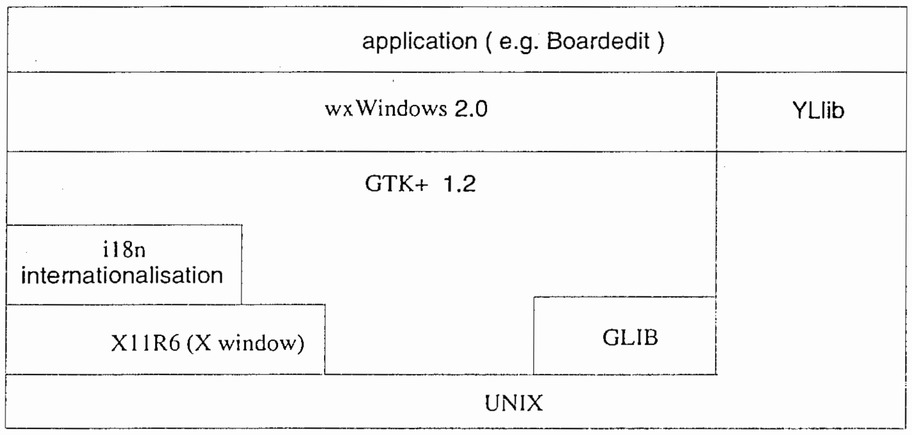


Figure 2.5: The layered structure of an application like `Boardedit`

2.4.1 wxWindows

wxWindows [Smart 92] (Web page: `http://web.ukonline.co.uk/julian.smart/ wxwin`) is a C++ framework providing GUI and other facilities on more than one platform. Version 2.0 currently supports MS Windows (16-bit, Windows 95 and Windows NT) and GTK+, with Motif and Mac ports in an advanced state.

wxWindows is implemented as a set of libraries that allows C++ graphical applications to run on several different types of computer, with minimal source code changes. There is one library per supported GUI (such as Motif, or Windows). However a common API (Application Programming Interface) is provided for GUI functionality, which makes it possible to access commonly used operating system facilities, such as copying or deleting files. wxWindows is a 'framework' in the sense that it provides a lot of built-in functions, which an application can use or overload as needed, thus saving the programmer a great deal of coding effort: *e.g.*, basic data structures such as strings, linked lists and hash tables are supported.

History wxWindows was started in 1992 at the Artificial Intelligence Applications Institute, University of Edinburgh, by Julian Smart for MFC 1.0 and XView (hence w for Windows and x for X). As it became clear that XView would no longer be supported, a Motif port was written.

During 1995, a port of wxWindows to Xt, the X toolkit, was released. From wxWindows 2.0, the need was felt for APIs. in August 1997, wxWindows 2.0 was ported to GTK, a graphical toolkit built on top of X11. GTK's major problem was that it is C-based, and only a thin (and unportable) C++ wrapper existed for it. In May 1998, the Windows and GTK ports were merged into a CVS repository. In September 1998, a new version of the wxMac 2 port was started and a beta version was released in February 1999. A BeOS port was started shortly after.

An official release of the wxGTK and wxMSW was distributed in early 1999.

Requirements To make use of wxWindows, one of the following setups is necessary.

- Unix:
  Almost any C++ compiler, including GNU C++.

Almost any Unix workstation, and one of: GTK+ 1.x, Motif 1.2 or higher, Lesstif.
At least 60 Mb of disk space.

- MAC OS:
  A MAC OS compiler: supported compilers include Metrowerks Code-Warrior.
  At least 60 Mb of disk space.

- MS Windows:
  A 486 or higher processor.
  A Windows compiler: most are supported. Supported compilers include Microsoft Visual C++ 4.0 or higher, Borland C++, Cygwin, Metrowerks CodeWarrior.
  At least 60 Mb of disk space.

2.4.2 GTK+

GTK+ (Web page: `http://www.gtk.org`) is a multi-platform open source GUI toolkit. It is a set of libraries to create graphical user interfaces. It works on many Unix-like platforms, and a Windows version is under development. GTK+ is released under the GNU Library General Public License (GNU LGPL), which allows for flexible licensing of client applications.

GTK+ is also called the GIMP toolkit because it was originally written for developing the General Image Manipulation Program (GIMP), but GTK has now been used in a large number of software projects, including the GNU Network Object Model Environment (GNOME) project. GTK is built on top of GDK (GIMP Drawing Kit) which is basically a wrapper around the low-level functions for accessing the underlying windowing functions (Xlib in the case of the X windows system).

GTK+ has a C-based object-oriented architecture. Bindings for other languages have been written, including C++, Objective-C, Guile/Scheme, Perl, Python, TOM, Ada95, Free Pascal, and Eiffel.
GTK+ consists of the following component libraries:

1. GLib: A lower-level library that provides many useful data types, macros, type conversions, string utilities, memory allocation, warnings and assertions and a lexical scanner.

2. GDK: A wrapper for low-level windowing functions that lies on top of Xlib. GDK also provides routines for determining the best available color depth and the best available visual, which is not always the default visual for a screen.

3. GTK+: The set of advanced widgets.

2.4.3 YLlib

This is a collection of C functions which offers a set of basic algorithmic objects: atoms, lists, AVL trees, forests, *etc.* [Lepage 92b]. The most relevant features for our project are the data types for correspondences, and the handling of various tree formats [Lepage 92a].

Chapter 3

Editing Facilities

3.1 A model of tree edition

The board editor must incorporate a tree editor. We have looked at several applications and solutions to the problem of editing a tree.

As we have already seen with already available tools, tree-editing is particularly cumbersome as they use the root window editing method (see section 5.3). For instance, a dialogue box opens when inputting a label, one can only add new nodes by means of a menu, *etc.*

This way of operating would be highly criticised if it were adopted for a text editor. For example, suppose the user had to click on a zone marked "new word" each time he wanted to insert a new word. Then, he would have to double-click on the newly created word place. A box would appear which would say: "type in the new word," *etc.* This is much too complicated. In the same way text editing has been made natural, we want our tree-editing method to be natural (on-the-spot method, see section 5.3.1)

3.1.1 Parallel between text and tree editing

The board editor incorporates a very special case of trees: nodes bear only labels, and no further information. Moreover, in the design of boards, there has always been a tendency to unify the data structures of the tree and the text part. This was realised by the work on the wood data structure [Lepage 94], which allowed the text and the tree part to be considered as being the same data structure, and hence allowed them to share exactly the same functions and operations.

Tree	Text
label of node	word
node	–
complete subtree	lines

Table 3.1: The parallel between text-editing and tree-editing

This leads to a parallel between nodes and subtrees on the one hand, and words and lines on the other hand (shown in table 3.1) [Lepage & Ando 96]. This parallel underlies all functions of editing on trees: any editing function (click, select, insert, cut, copy, paste, *etc.*) for trees will have exactly the same behaviour as in text-editing.

Although the parallel clearly shows that a node is different from a label for a naive user, the distinction between a node and a label is usually not intuitive. On the contrary, people usually think that "a label is a node" To make our tool intuitive to users, we shall not contradict this way of thinking as much as possible. Only in certain cases of edition where the difference between a label and a node is unavoidable shall the editor make the difference (see section 3.2.1, notion of an editing transaction).

3.1.2 Edition

In our model, there are two types of edition:

- Selection.
- Manipulation.

Selection: structural object & editing mode

Thanks to the parallel shown in table 3.1, two selectable structural objects appear:

- Nodes.
- Complete subtrees.

Hence, we proprose the following interpretations of clicks which correspond to two different kinds of manipulation: label editing mode and object or structural editing mode.

- One click on a node: label editing mode.
- Double and triple click on a node: structural editing mode for selection of node and selection of complete subtree, respectively.

Manipulation

There are three ways to manipulate a tree:

- Editing using the keyboard.
- Editing using the mouse.
- Editing using the clipboard.

Manipulation corresponds to the two different kinds of editing mode:

- Label editing mode: manipulation of a node label using the keyboard, and
- Structural editing mode: manipulation of a node or complete subtree using the keyboard, mouse and clipboard.

3.2 Label editing mode: editing using the keyboard

Like with a text editor, trees can be edited with the keyboard. Many parallels exist between editing text and editing a node label using the keyboard.

Editing a node label using the keyboard is only possible when the user is in label editing mode. Label editing mode is signalled by a cursor appearing under the current node. If the tree editing window loses the focus, the cursor automatically changes.

3.2.1 Edit transaction

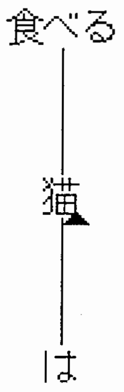
In our editor, a new node is created with an empty label, except for nodes created using the clipboard. If a user changes the edit mode, a node with an empty label will be automatically deleted. This implies the notion of an edit transaction, as there are two states:

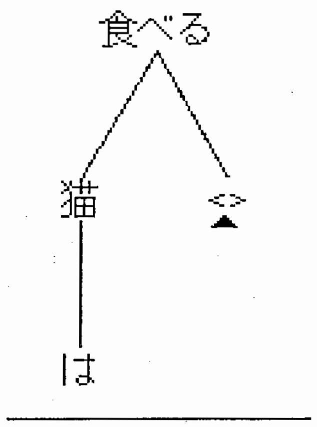
- An insecure state: it starts when a label is empty and remains until the user enters the node label.
- A secure state: the end of the edit transaction, when the user confirms the node label.

When a tree is displayed, there is no difference between editing a node or editing a label of the node, but internally during an edit transaction, a node can exist with an empty label. This is an insecure state, because until the user enters the node label, he may change his mind and do something else. An empty node label is represented by the symbol character <>, which disappears as soon as the node label contains one character.

Semantics of the space bar

With a text editor, words are separated by a space. In a parallel way, when editing a label, hitting the space key automatically creates a node sister for the previous node. This method implies two things: a label of a node cannot contain a space, and typing a space adds a right sister to a node (the direction of the text input is left to right).

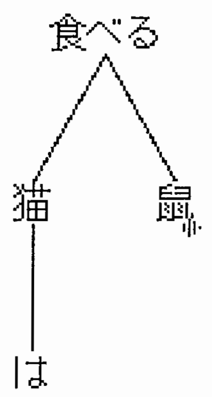


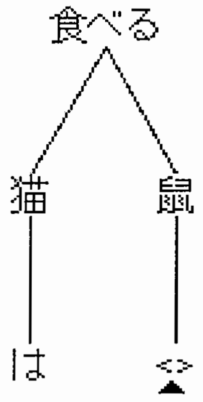


Of course, if a space marks the separation between two words, the deletion of the space between two words implies the concatenation of the two words. The parallel is kept for label editing: the deletion of the space between two sister nodes, either by hitting the del key at the end of the left sister, or by hitting the backspace key at the beginning of the right sister, merges the two nodes into one, and concatenates their labels.

Semantics of the return key

With a text editor, the user adds or inserts a new line by hitting the return key. Because of our parallel, in the label editing mode, hitting the return key will create a daughter to a node. This implies that a label is necessarily on a single line (no two-line labels).





Of course, the deletion of the first character of the first left sister node implies, as with any text editor, that all the daughters of this node become her right sisters. In the same way, in the middle of a label, if the enter key is hit, the label will be cut at the cursor position and a daughter node will automatically be created, with the last part of the label as its label.

Semantics of the arrow keys

In all text editor, the arrow keys enable the user to browse the text. In the same way, in our tree-editor, arrow keys allow the user to traverse the tree node by node.

Although there is no equivalent in usual text editors, the arrow keys associated with the shift key will have a special meaning: they create a new empty node in the direction of the arrow key. For example, the shift key together with the up key creates a new mother mode to the current node. The new node is created with an empty label (noted <>), and an edit transaction is entered. Unless the user types in a label, the new node will disappear with any other action (see Section 3.2.1).

3.3 Structural editing mode

As the name implies structural editing mode is used to modify of the structure of the tree. In the structural editing mode, the user is able to select a structural object. The user can select a structural object in the same way as he selects words and lines by double or triple clicking the mouse respectively, in a text editor.

3.3.1 Editing using the keyboard

In the structural editing mode, when a structural object is selected, any editing key will delete the entire selection, as is also true in a text editor, and put the last node or the root node of the subtree, into the label editing mode.

3.3.2 Editing using the mouse

In a text editor, a new word is inserted by simply clicking where the new word is to be added, and then typing in the word. Similarly, in the tree editor, a new node is added by simply clicking in the corresponding position in the tree where the node is to be added. This implies that clicking on some special areas on the screen will automatically create a new node, hence creating new branches where necessary.

There is another way of editing called drag and drop. The effect of a drag and drop operation is similar to the use of the clipboard to cut, copy and paste data. A description of editing using the clipboard is given in the following section.

3.3.3 Editing using the clipboard

Cut/copy/paste

When editing text, it is common to move objects around, for example, to exchange two words. This has been abstracted thanks to the intuitive paper-and-scissors metaphor into three basic actions: cut, copy and paste. These items are common in any editing tool.

The Copy command copies a selected structural object into a special container, the clipboard. The Cut command is the same as copy, but it deletes the selected object from the screen. The Paste command inserts the content of the clipboard into a selected place which can be a node or a special area (see section 3.3.2). It doesn't change the content of the clipboard, so that it is possible to paste a node or a complete subtree several times.

Drag and Drop

Drag and Drop (DnD) is a data transfer mechanism and allows an application to transfer a piece of data of any type to the same process or to another one.

To start a DnD operation, the user presses mouse button 1 (left) on the selected structural object and drags the selected object to the same window or to another one which must be at least partially visible on the screen. To end the operation, the user releases the button. The default DnD operation is "copy," but pressing `<Ctrl>` simultaneously performs a "cut."

3.4 wxForestedit implementation

3.4.1 The MVC model

The MVC model (Model, View, Controller) is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model of the application, the view, and the controller. MVC was originally developed to map the traditional input, processing and output roles into a GUI.

$$\begin{tikzcd}
\text{Input} \arrow[r] & \text{Processing} \arrow[r] & \text{Output} \\
\text{Controller} \arrow[r] & \text{Model} \arrow[r] & \text{View}
\end{tikzcd}$$

The user input, the modeling of the external world, and the visual feedback to the user are separated and handled by the model, viewport and controller objects. The controller interprets mouse and keyboard inputs from the user and maps these user actions into commands that are sent to the model and/or viewport to effect the appropriate change. The model manages one or more data elements, responds to queries about its state, and responds to instructions to change states. The viewport manages a rectangular area of the display and is responsible for presenting data to the user through a combination of graphics and text.

The model is used to manage information and notify observers when that information changes. It contains only data and functionality that are related by a common purpose.

The view or viewport is responsible for mapping graphics onto a device. A viewport typically has a one-to-one correspondence with a display surface and knows how to render it. A viewport attaches to a model and renders its contents to the display surface. In addition, when the model changes, the viewport automatically redraws the affected part of the image to reflect those changes. There can be multiple viewports on the same model and each of these viewports can render the contents of the model to a different display surface.

A controller is the means by which the user interacts with the application. The controller accepts input from the user and instructs the model and viewport to perform actions based on that input. In effect, the controller is responsible for mapping end-user actions to application responses. For example, if the user clicks the mouse button or chooses a menu item, the controller is responsible for determining how the application should respond.

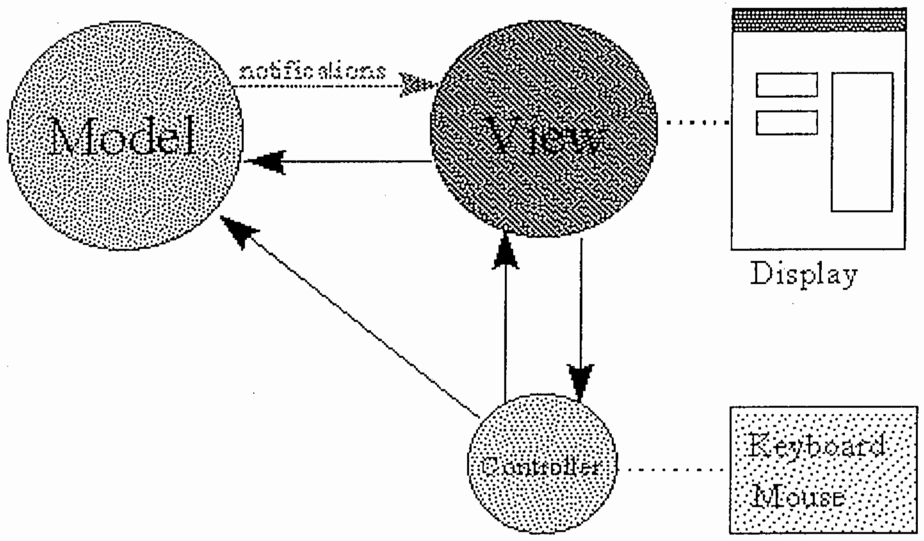


Figure 3.1: The MVC model

The model, viewport and controller are intimately related and in constant contact. Figure 3.1 illustrates the basic Model-View-Controller relationship. It shows the basic lines of communication among the model, viewport and controller. In this figure, the model points to the viewport, which allows it to send the viewport weakly-typed notifications of change. Of course, the model's viewport pointer is only a base class pointer; the model should know nothing about the kind of viewports that observe it. By contrast, the viewport knows exactly what kind of model it observes. The viewport also has a strongly-typed pointer to the model, allowing it to call any of the model's functions. In addition, the viewport also has a pointer to the controller, but it should not call functions in the controller aside from those defined in the base class. The controller has pointers to both the model and the viewport and knows the type of both. Since the controller defines the behavior of the triad, it must know the type of both the model and the viewport in order to translate user input into application response.

3.4.2 The wxForestedit component

We implemented a non runnable component for forest edition with a general tree specification called wxForestedit. wxForestedit is not a widget (graphical object). wxForestedit has been designed according to the MVC model. This MVC model is show in Figure 3.2.

wxForestedit comprises three C++ classes:

- wxForestEditCtrl (controller): reads from a file or a string. It act on the set of nodes by adding or deleting a mother, a daughter or a sister by functions which take as argument the current node number. This object verifies the general tree specification.

- wxForestEditView (viewport): manages scrollbars when the forest is bigger than the viewport. wxForestEditView is only able to display a forest from the top to the bottom. However, the MVC model makes it easy to envision any other way.

- wxForestEdit (Model): is just a hashtable for the set of nodes and the Set/Get methods associated. A node has a unique identifier in time, which easies the implementation of the Undo functionality

3.4.3 The wxForestedit component in `Boardedit`

In `Boardedit`, wxForestedit has to handle forests with special data like correspondences.

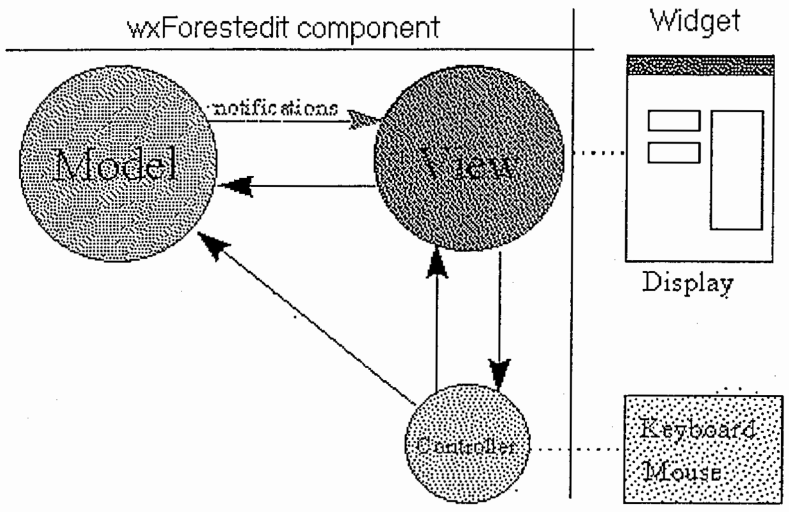


Figure 3.2: The wxForestedit component

Because wxForestedit is not a widget, we implemented a widget and a new controller to manage the forest in `Boardedit` show in Figure 3.3.

bdForesteditWidget We created a widget for forest edition according to the specifications described in the previous sections 3.1, 3.2 and 3.3 and based on the wxForestedit component with a more specific controller, bdForesteditCtrl.

bdForesteditCtrl We created an inherited class of wxForestEditCtrl (which is the controller of wxForestedit) called bdForesteditCtrl. This new controller verifies all the editing choices explained in section 3.1.2. bdForesteditCtrl is able to read and manage the C FOREST type, which is a basic C type in our C library ([Lepage 92b]). For Instance, the search functionality makes use of the C FOREST type.

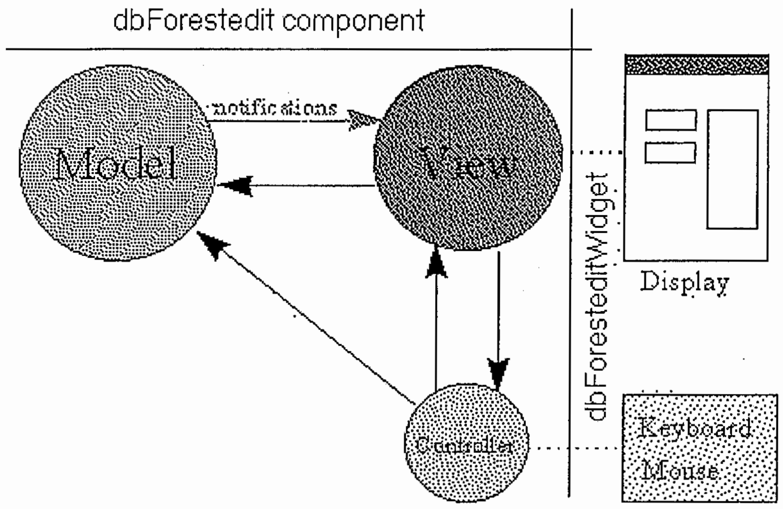


Figure 3.3: The bdForestedit component

3.4.4 Exporting wxForestedit

wxForestedit as a MVC model can be exported to other systems or GUIDEs. based on this component, we have proposed the construction of an ActiveX control (OCX) which could become a product for the PC MS-Windows environment. Also we reimplemented `TreeCanvas`, a tool mentioned in section 2.3.

ATRForestedit OCX enables developers to display and manipulate a forest in a program. ATRForestedit OCX has the same features as wxForestedit under wxWindow. It was implemented and successfuly demonstrated under some MS-Windows applications (MS Word, Internet Explorer). See appendix C.

`TreeCanvas` allows to diplay a tree or a forest. Interestingly, it acts as a filter between different formats of tree. See appendix D.

Chapter 4

Search Functionality

The aim of `Boardedit` is to allow a treebanker to build a treebank in a faster and more consistent way. We propose to view the construction of a treebank as a sequence of editing (see section 2.1) and searching steps, as follows:

- edit a new sentence (input it);
- find similar sentences, to retrieve their associated structures, so as to propose possible candidate linguistic structures;
- edit the new linguistic structure (build it);
- find similar structures in order to check or ensure consistency with previous data in the treebank.

4.1 Ideal case

The ideal way of obtaining a linguistic structure for a new sentence is to have a complete parser for the language in which the sentence is written, and to feed the sentence to this parser. Hence the steps would be:

- edit a new sentence (input it);
- get the structure from the parser.

4.2 Reality

Unfortunately, as parsing is still an object of research, and as `Boardedit` is designed to be an aid in this kind of research, `Boardedit` proposes to gradually fill the gap between editing by hand and complete automatic parsing. Hence we propose some parsing aids:

- Exact match
- Closest match
- Analogical completion

These three searching methods work as well on single-byte characters (such as French) as on multi-byte characters (such as Japanese). But the treebanker has to be precise concerning the type of characters for which the base file is encoded.

Exact match is performed to search in the base file for the exact sentence selected by the treebanker.

Closest match is performed to search for the closest sentence in the base file selected by the treebanker. This method uses approximate matching by setting the threshold, and stops when it gets a result.

Analogical completion is more than a searching method, it is a parsing aid. It does not only propose candidate linguistic structures, but builds an adapted candidate for the sentence selected from the base file. It is based on analysis by analogy.

4.3 Integration under `Boardedit`

4.3.1 Find dialog box

So as to easily select a search method, we implemented a Find dialog box. This dialog box appears when a treebanker pushes the Option button or chooses Find in the Edit menu.

To perform a search in `Boardedit`, the search method and the treebank (base file name) must be set. These can be set at the `Boardedit` launch (see appendix B). But the treebanker can change them with the Find dialog box.

Figure 4.1 shows the Find Dialog box. On the left, the treebanker can choose the search method by clicking on the search method name. On the right, there is a push button for setting the name of the base file and an option button to set the granularity. In the Figure 4.1, there is also another search method called approximate matching, which will be explained in a later section.

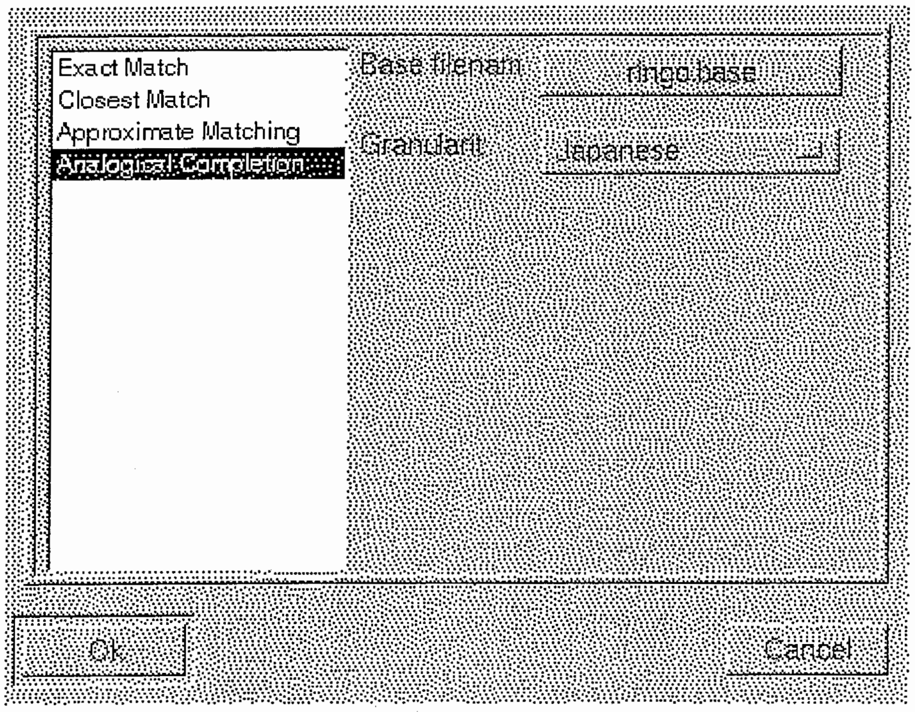


Figure 4.1: Find dialog box

Base file or treebank can be set by pushing the button. This action will open a File selection dialog box (see Figure 4.2) to browse all directories and files. When a file name is selected, its name will appear as the caption of the button. The base file name can be changed at any time. When the base file is not set, the caption of the push button is "no file name."

Granularity is a discrete set. So the treebanker is able to change the granularity by simply clicking on the option button and choosing the correct granularity. As explained in appendix B, the granularity is often set with the base file name. Granularity can also be set at the `Boardedit` launch.

There are three types of granularity:

- Latin char: for French, English, *etc.*
- Japanese: for double-byte characters
- Words: to perform a search by words, this is an option for the future.

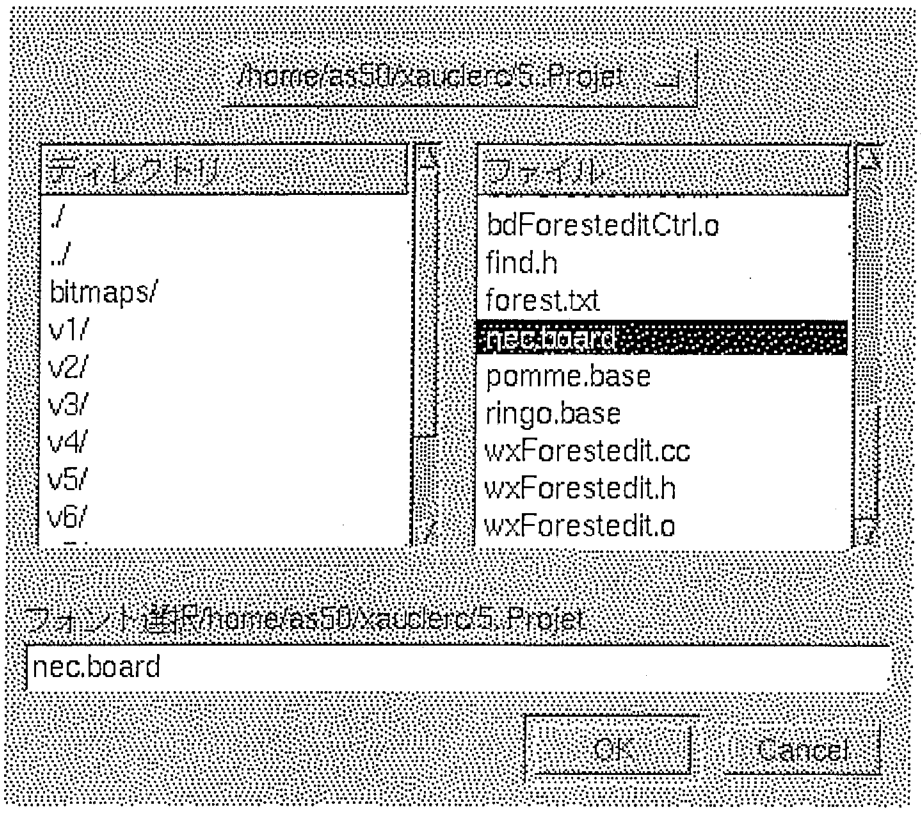


Figure 4.2: File selection dialog box

4.3.2 Use of the editing facilities

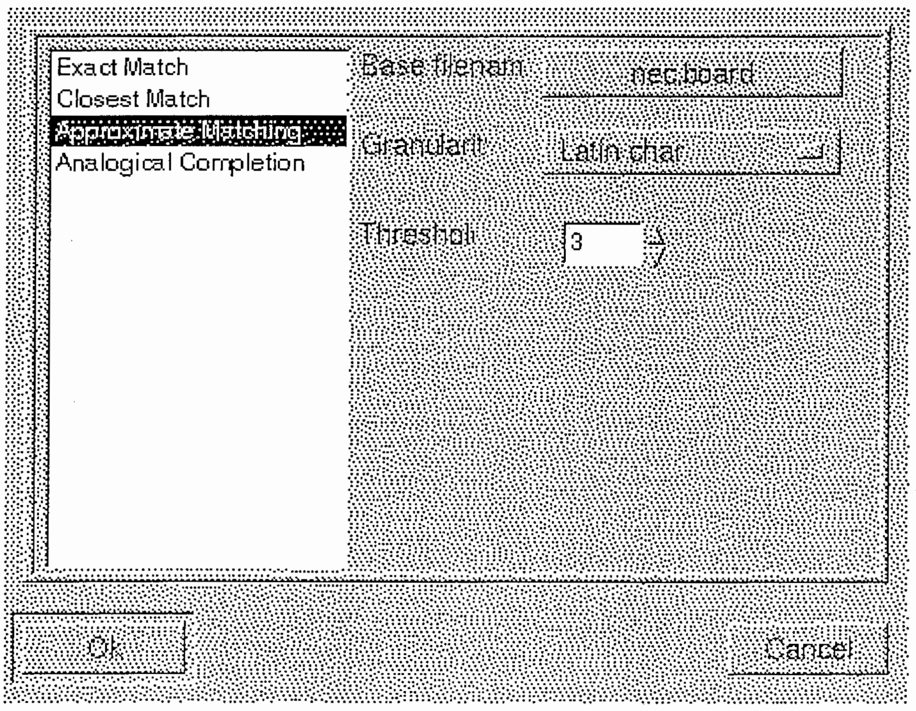
`Boardedit` is an editor which implements searching functions. The result given by those search methods has to be edited to adapt it to the sentence. So there is a link between the search method and the editing facilities.

- Exact match: copy/paste or DnD the result. It means that the sentence was already in the treebank.
- Closest match: copy/paste or DnD the best proposed linguistic structure and edit it to fit with the sentence.
- Analogical completion: copy/paste or DnD the best built linguistic structure.

4.3.3 Plug-in facility

Thanks to the plug-in system we implemented to manage the different search methods, `Boardedit` is able to support many other parsing aids, like an analyser. `Boardedit` also proposes another seach method, approximate matching [Lepage 97], which is already used by the closest match method. Approximate matching needs a new parameter: a threshold. All new options needed for the new parsing aid will be added after the two default parameters: base file name and granularity.

The approximate matching option in the Find dialog box is shown in Figure 4.3.



Base filenam: nec.board
Granularit: Latin char
Threshol: 3

Figure 4.3: Approximate matching option

4.4 Parsing using `Boardedit`

In this section, an example of parsing will show how it is done using `Boardedit` with the Find dialog box. This example uses the ATR NEC treebank, which is in Japanese. The following are some boards taken from this treebank:

```
...
か (IT(I, +えます (は (O))))              "食べ物は持ち込めますか。"
IT(I, +えません (は (O)))                "食べ物は持ち込めません。"
か (IT(ET(I(は (O)), +て), います))      "寝袋は持っていますか。"
...
```

The new sentence we have is 寝袋は持っていません。 and we want its linguistic structure. So we are going to parse this sentence using `Boardedit` help to get it.

4.4.1 Using exact match

Figure 4.4 shows that there is no result with the exact matching method because the sentence does not exist in the treebank. At this point, the treebanker has to continue to search with `Boardedit` help or just make the linguistic structure himself with the editing facilities.

4.4.2 Using closest match

The closest match method is more helpful than the exact match because with this same sentence, it is able to find two answers in the treebank.

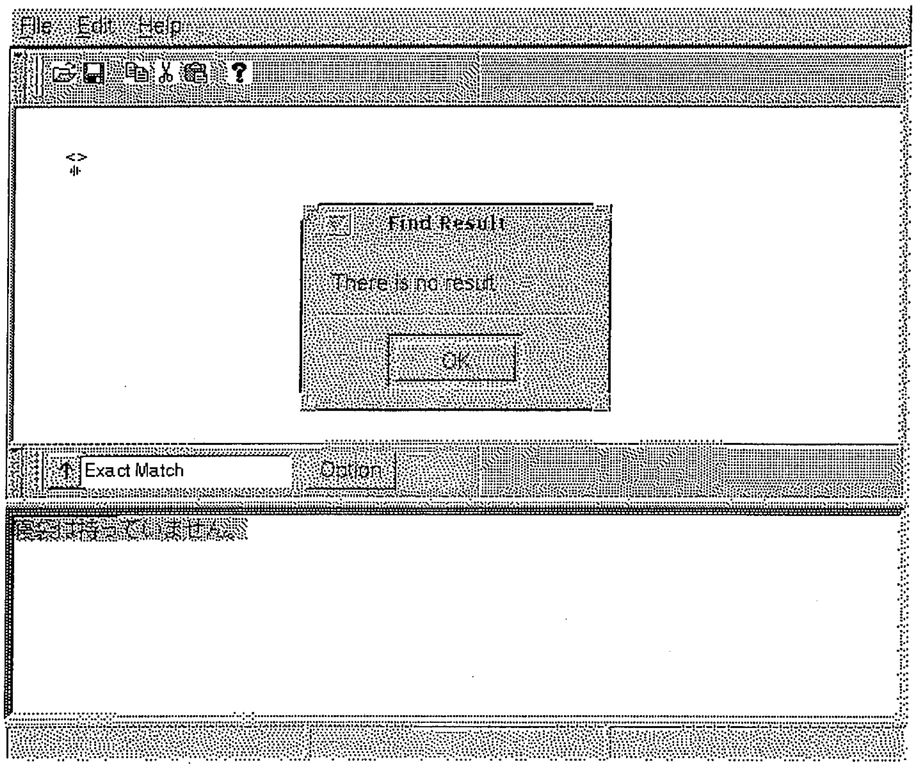
For the sentence 寝袋は持っていません。, we get the following two results:

1. `IT(ET(I(は (O)), +て), いません)`   "カードは持っていません。"

2. `か (IT(ET(I(は (O)), +て), います))`   "寝袋は持っていますか。"

These two results are shown in Figures 4.5 and 4.6. They are not perfect because they are only close to the given sentence. The treebanker has thus to edit the proposed linguistic structure.

In the first answer (Figure 4.5), カードは持っていません。, only the beginning of the sentence has changed: カード replace 寝袋 . By chance, the linguistic structure of the first answer, is adapted to the given sentence so



Find Result

There is no result.

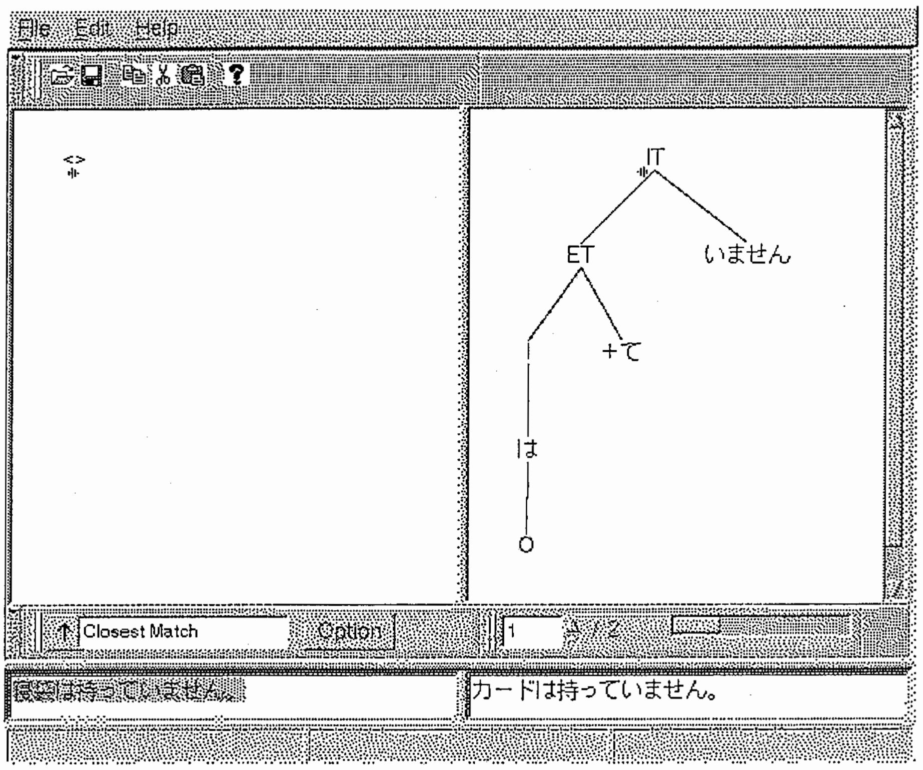
寝袋は持っていません

Figure 4.4: Exact match result

the treebanker does not have to edit the linguistic structure. In the second answer (Figure 4.6), 寝袋は持っていますか, only the end of the sentence (the form of the verb) has changed. But in this case the linguistic structure has to be edited to adapt to the given sentence. The treebanker has thus to edit the answer か (IT(ET(I(は (O)), +て), います)): delete the node か because the given sentence is not a question and replace the node います by いません because the verb of the given sentence is in the negative form.

4.4.3 Using analogical completion

The analogical completion method is more than a search method bcause the technique creates an answer from the treebank. Figure 4.7 shows that there is one answer when the treebanker uses analogical completion on the given sentence. The linguistic structure we get is already adapted to the given



[illegible]は持っていません。

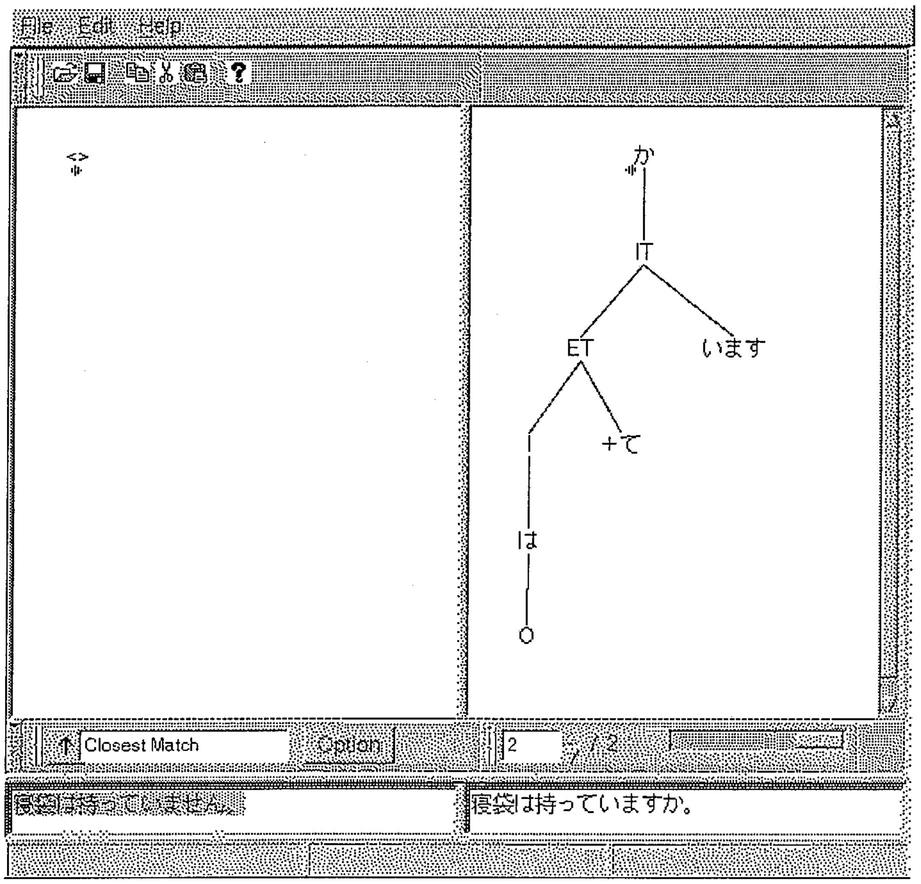
カードは持っていません。

Figure 4.5: Closest match First result

sentence because the given sentence and the sentence of the answer are the same since the answer has been built from sentences of the treebank.

4.5 Conclusion

In this chapter, we have seen three search methods and parsing aids: exact match, closest match and analogical completion. These three methods were explained from the most simple (exact match) to the one offering the most help (analogical completion), but when a treebanker wants to have the linguistic structure of a sentence, he is free to choose his own way to use these search methods. It seems to us that the most reasonable order in which to use them is as follows:

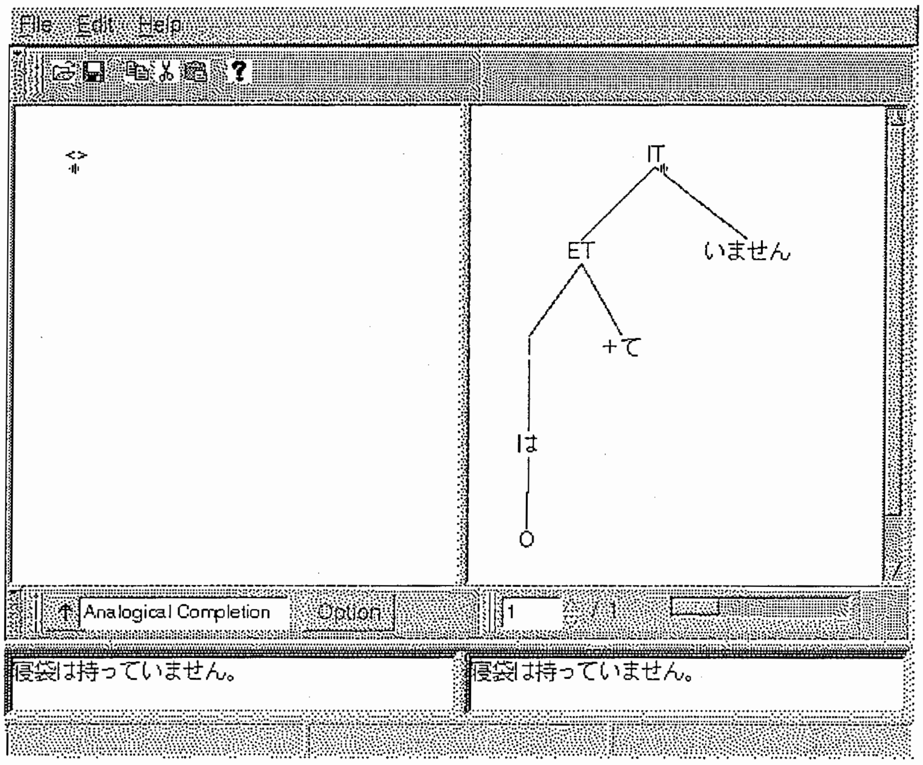


寝袋は持っていません。

寝袋は持っていますか。

Figure 4.6: Closest match second result

- Exact match: the given sentence already exists in the treebank; we get the linguistic structure as the final answer.
- Analogical completion: the sentence does not already exist. So we may look in the treebank to see if we can find sentences which are in an analogical relationship (see paragraph 2.2.2) and then get the linguistic structure by the analogical relationship with the linguistic structures of the sentences found.
- Closest match: if an analogical relationship is found, then we may look in the treebank to see if we can find a similar sentence to the given sentence and edit the linguistic structure of the sentence found to get a valid linguistic structure



寝袋は持っていません。

寝袋は持っていません。

Figure 4.7: Analogical completion result

Chapter 5

Internationalisation (I18n)

Boardedit is designed to edit linguistic structures in order to build tree banks for machine translation. This implies that Boardedit should support different languages, like Japanese or French, in displaying and editing.

Nowadays, there are general methods for handling different languages in the same application. All of the description about a particular language is found in a system called locale. Boardedit will use the codeset defined by the locale to display and draw the characters of the language. Boardedit also needs to display menus and messages in many languages as the tree bankers may have different mother tongues. For this, GNU proposed a tool named gettext.

Boardedit must also allow the tree banker to edit languages like Japanese which have more characters than can fit on a standard keyboard. So the Japanese characters, like other Asian languages, are input with special methods called input methods.

The locale implies some special problems when mixing languages. For example, Boardedit in the Japanese locale can edit and display French characters like c-cedille, but the French locale cannot display and edit Japanese characters.

wxWindows supports locales, but does not support input methods. Nevertheless, this is supported by GTK+. As explained in section 2.4 and shown in figure 2.5, Boardedit is built with wxWindows but also relies on GTK+.

5.1 Locales

A locale describes the user's environment: the local conventions, culture, and language of the user's geographical region. A locale is made up of a unique combination of a language and a country. Two examples of locales are: French/Canadian and English/U.S.

A language might be spoken in more than one country; for instance, French is spoken in France, Belgium, Switzerland, Italy (val d'Aoste), Canada, and many African countries. While these countries share a common language, some national conventions (such as currency) vary among the countries. Therefore, each country represents a unique locale. Similarly, one country might have more than one official language. Canada has two: French and English. Therefore, Canada has two distinct locales.

5.1.1 The concept of locale

A locale is a language environment determined by the application at run time. It includes the specification of a language, the territory, and the codeset.

Locale precedence rules are the rules determined by the definitions of LANG, LC_ALL, and the other LC_ environment variables for setting the locale associated with the various categories.

LANG is an environment variable which determines the locale for any category not specifically selected via a variable starting with LC_. Additional semantics of this variable, if any, are implementation-defined.

LC_ALL is an environment variable which shall override the value of the LANG variable and the value of any of the other variables starting with LC_.

5.1.2 Locale categories

A category is a set of internationalisation (i18n) features all presented to the user in the same locale. It is one of the following: Characters and Codesets, Dates, Numbers, Currency, and Messages.

Characters and Codesets The 8-bit ISO 8859-1 codeset has special characters needed to handle the major European languages. However, in many cases, the ISO 8859-1 font is not adequate. The 16-bit JIS 0208-0 (1983) codeset is used for Japanese. Hence each locale will need to specify which codeset they need to use and will need to have the appropriate character handling routines to cope with the codeset. This part of the locale constitutes the main use for `Boardedit`, because one of the main function of `Boardedit` is to draw characters.

- The ANSI standard uses only a single byte to represent each character, so it is limited to a maximum of 256 character and punctuation codes. Although this is adequate for French or Canadian, it doesn't fully support other languages.

- The Double Byte Character Set (DBCS) is used in most parts of Asia. It provides support for many different East Asian language alphabets, such as Chinese, Japanese, and Korean. DBCS uses the numbers 0 to 128 to represent the ASCII character set. Some numbers greater than 128 function as lead-byte characters, which are not really characters but simply indicators that the next value is a character from a non-Latin character set. In DBCS, ASCII characters are only 1 byte in length, whereas Japanese, Korean, and other East Asian characters are 2 bytes in length.

- Unicode is a character-encoding scheme that uses 2 bytes for every character. The International Standards Organisation (ISO) defines a number in the range of 0 to 65,535 for every character and symbol in every language. Although both Unicode and DBCS have double-byte characters, the encoding schemes are completely different.

Currency The symbols used vary from country to country as does the position used by the symbol. Software needs to be able to transparently display currency figures in the native mode for each locale. This is not important for `Boardedit`.

Dates The date format varies between locales. For example:

- In French: lundi, 16 août 1999, 09:38:36

- In Japanese: 1999 年 08 月 16 日 (月) 09 時 43 分 17 秒

Dates are not used by `Boardedit`.

Numbers Numbers can be represented differently in different locales. For example, the following numbers (same value) are both written correctly for their respective locales:

French:	12 345,67
Japanese:	12,345.67

Messages The most obvious area is the language support within a locale. This is where GNU gettext (see section 5.2) provides an easy way for developers and users to change the language that the software uses to communicate to the user.

The concept of categories allows better organisation of the locale information and the implementation of mixed language environments.

- Single language environment: an environment in which all I18N features of a program are presented in the same locale.
- Mixed language environment: an environment in which some I18N features of a program are presented in one locale and other I18N features of the same program are presented in another locale.

As explained in the next section, `Boardedit` uses GNU gettext to display menus and messages in the tree banker's mother tongue.

5.2 Localisation

Localisation is the process by which an application is adapted to a locale. It involves more than just literal, word-for-word translation of the resources. It is the meaning that must be communicated to the user.

5.2.1 GNU gettext

The GNU Translation Project is a formalisation of the internationalisation problem into a workable structure, in order to achieve a truly multi-lingual set of programs.

The GNU gettext utilities are tools that provide a framework to help other GNU packages produce multi-lingual messages. These tools include a set of conventions about how programs should be written to support message catalogs, a directory and file naming organisation for the message catalogs themselves, a runtime library supporting the retrieval of translated messages, and a few stand-alone programs to display in various ways the sets of translatable strings, or already translated strings.

5.2.2 Files conveying translations

The letters PO in '.po' files mean Portable Object, to distinguish them from '.mo' files, where MO stands for Machine Object. This paradigm, as well as the PO file format, was inspired by the NLS standard developed by Uniforum, and implemented by Sun in their Solaris system.

PO files are meant to be read and edited by humans, and associate each original, translatable string of a given package with its translation in a particular target language. A single PO file is dedicated to a single target language. If a package supports many languages, there is one such PO file per language supported, and each package has its own set of PO files. These PO files are created by the xgettext program, and later updated or refreshed through the tupdate program. The xgettext program extracts all marked messages from a set of C files and initializes a PO file with empty translations. The tupdate program takes care of adjusting PO files between releases of the corresponding sources, commenting on obsolete entries, initializing new ones, and updating all source line references.

MO files are meant to be read by programs, and are binary in nature. A few systems already offer tools for creating and handling MO files as part of the Native Language Support coming with the system, but the format of these MO files is often different from system to system, and non-portable.

5.3 Input method editing styles

Each platform (Unix/X, Macintosh, Windows) supports the input of several Asian languages (*e.g.*, Japanese, Chinese, Korean) through a special system service called an Input Method. An input method is a software component that converts keystrokes into text input which cannot be typed directly. Input methods are normally used to input text for languages which have more characters than can fit on a standard keyboard. Input methods are com-

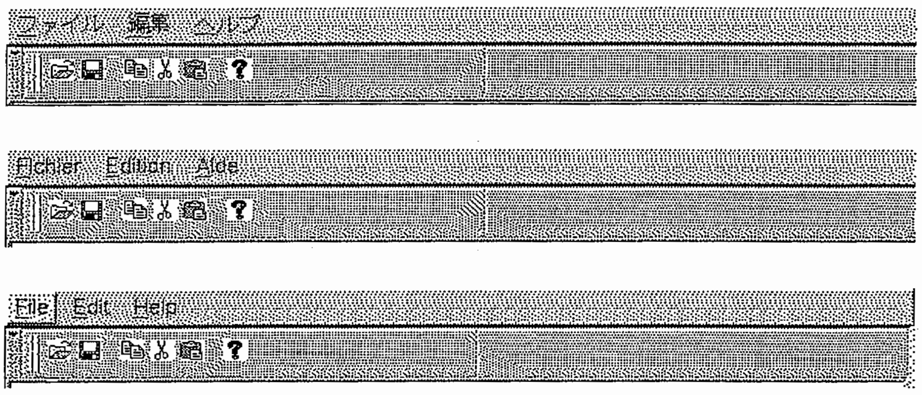


Figure 5.1: Same version of `Boardedit` using different locales

monly used for Japanese, Chinese and Korean, but also show up in other languages, like Thai and Hindi.

There are four basic styles of input method editing: on-the-spot, over-the-spot, off-the-spot, and root-window. Unlike Macintosh and MS Windows, the XIM (X Input Method) standard defines all four styles. The style used is negotiated from the set of common styles supported by the application and the input method.

In `Boardedit`, the On-The-Spot style is the default input method.

5.3.1 On-The-Spot composition style

The composed text is rendered inside a text window by the application, by maintaining a special editing area "between" the text that exists before the insertion point and the text that exists after the insertion point. The composed text looks like the text part of the document, however, different stylistic attributes are applied to the text to indicate that it is part of the input method composing string. Different parts of the input method composing string will have different styles applied to them, which indicates that they are in different stages of editing. Once the composition text is finalised by the user, it merges into the original document and is indistinguishable from the surrounding text. The on-the-spot style is also known as inline input on some platforms.

5.3.2 Over-The-Spot composition style

The composed text is rendered over the insertion point in a "layer" above the document window. The document text doesn't change until after the user has committed the text, so the composed text ends up obscuring part of the document during editing.

5.3.3 Root Window composition style

The composed text is rendered in an entirely separate window which has no relationship to the application window. Once the input is committed, it is then inserted into the document at the insertion point. The root window style is also known as bottom line or floating window on some platforms.

5.3.4 Off-the-Spot composition style

The off-the-spot composition style is very similar to the root window style. These two styles are distinguished only by the position of the editing region. The off-the-spot style draws the editing region in a status bar attached to the bottom of the active window. Each application window has a status and editing bar, instead of having a single independent window.

5.4 GTK+ I18n

5.4.1 Manipulating text

The most basic task that GTK+ (and applications using GTK+) have to handle when dealing with international text is manipulating strings. The strings in the GTK+ interfaces are handled in the multi-byte encoding (compatible with double-byte encoding (DBCS)) for the locale. This allows good compatibility with existing applications that aren't explicitly design for multi-byte support. Internally, GTK+ converts these strings to wide-character strings for easier manipulation, and the conversion routines are also available for applications that need these facilities.

5.4.2 Input

Internationalised input in GTK+ is done using the X Input Method extension (XIM). The X libraries include a simple built-in input method that does compose-key handling for European languages. The more complicated input method handling for Asian languages, such as Japanese, is typically done by an external program.

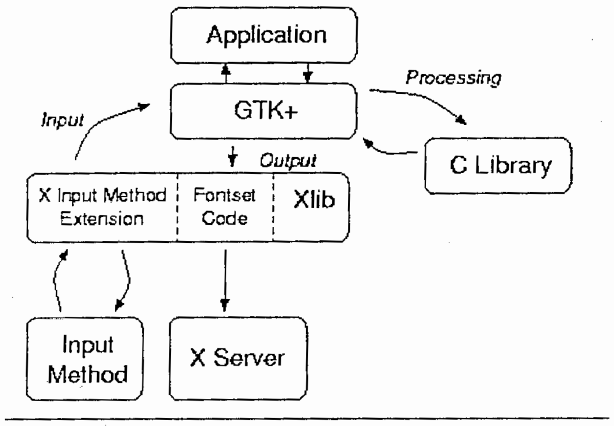


Figure 5.2: Architecture of XIM

Figure 5.2 shows the basic architecture of XIM. GTK+ forwards the keystrokes it receives to the input method via Xlib, and when a complete input string is received, it is displayed to the user. From the point of view of `Boardedit`, such as one using only the standard GTK+ Text and Entry widgets, this is all done transparently behind the scene, and the application only sees the final strings.

5.4.3 Output

GTK+ handles the output of strings in different scripts using font sets. A font set is a list of X fonts for different character sets. When drawing, for example, mixed Roman and Japanese, Chinese or Korean text, then the two different fonts needed are extracted with multiple character sets. For the Japanese locale, they are two different fonts: one single-byte font for katakana and Latin text and one double-byte font for kanji, hiragana, *etc.*

The font to use for a particular widget is generally determined in GTK+ using resource configuration (RC) files. There is a system-wide file that the system admininstrator can set up, and additionally, each user can override the settings by creating a file in his home directory. This mechanism has been extended to deal with getting the correct fonts for each locale, even for users that switch between different locales. When GTK+ reads in an RC file it also looks for the same file with an extension corresponding to the current locale. If the locale is ja for Japanese, then GTK+ will check for the file gtkrc.ja. GTK+ ships with gtkrc files for Japanese, Korean, and Russian, and a system administrator can easily create them for additional languages as needed.

5.5 Internationalisation of `Boardedit`

5.5.1 Input

`Boardedit` is able to input Japanese thanks to GTK+. Here is an example of the input of a Japanese sentence into `Boardedit`.

The Japanese sentence to input is: 今日は週末なので料金は高くなります。 (transcribed as *konnitiha syuumatu nanode ryoukin ha takaku narimasu*[1]). To input this sentence into `Boardedit`, the user begins to type phonetically into the edit area (text-editor or our tree-editor). The user types the series [*k,o,N,n,i,t,i*] which the input method automatically translates into the two Japanese syllables [こんにち] displayed on the figure. The romanised "k" remains because its conversion is still ambiguous. As the user types, the editor automatically expands the composition area. The Japanese text is displayed in highlight to indicate that it is unconverted text and still has additional input steps to go through.

After entering the pronunciation of the word he wants to add into the document (*koNniti*, the Japanese word for 'today'), the user selects the appropriate Kanji conversion for the syllables. Since a single syllable has multiple possible conversions, the input method will bring up a list of possible candidates. The user selects the appropriate conversion, which is then displayed in the document window. Once the user selects the appropriate conversion, the highlight style of the text changes.

[1]'Today is not a weekday, so it is more expensive.'

Finally, the user performs some action (usually pressing the return key) which commits the final text. The Japanese text is merged into the document and is indistinguishable from the surrounding text. The rest of the sentence can be edited in the same way.

5.5.2 Output

Of course `Boardedit` is able to display Japanese thanks to GTK+. The previous example show how Japanese is display under text an tree canvases.

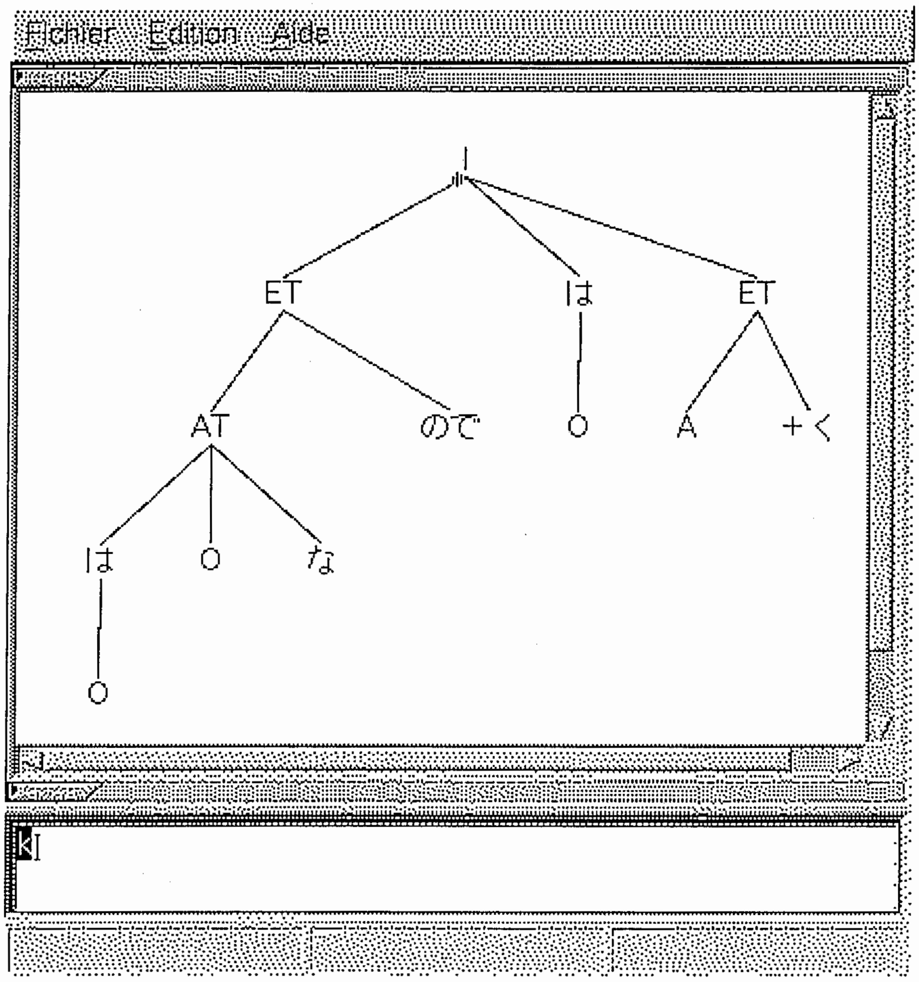
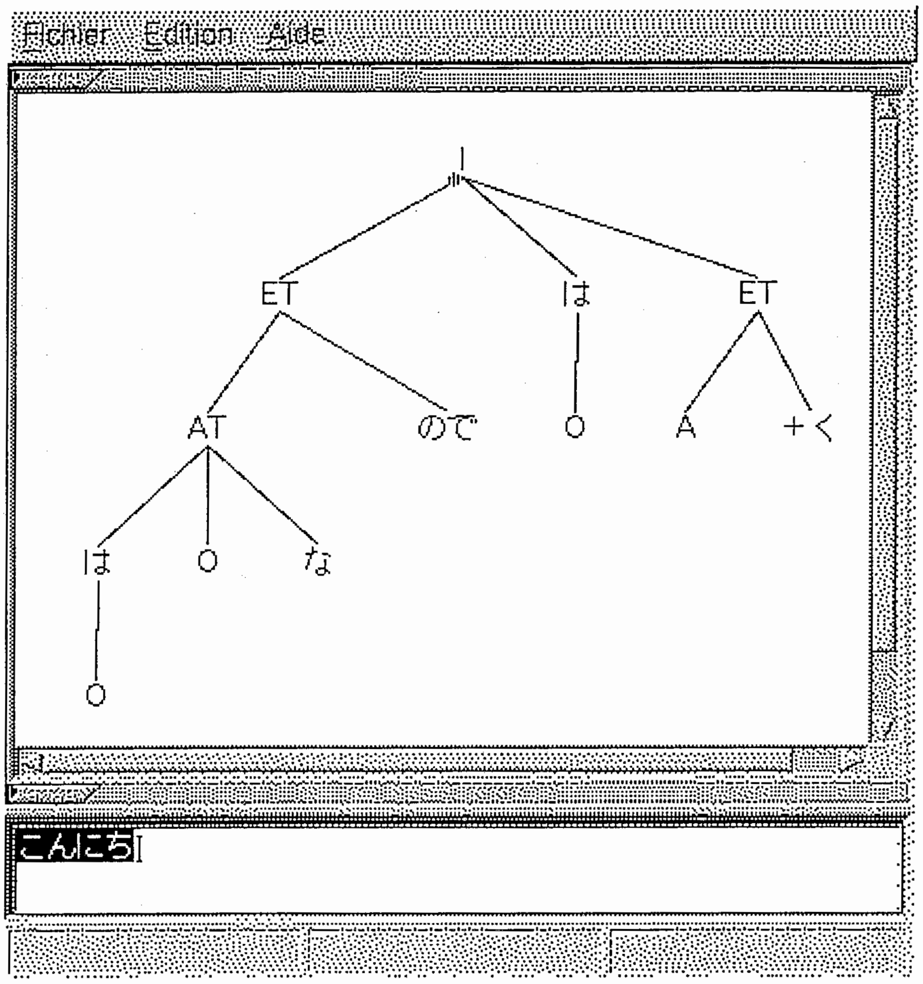


Figure 5.3: Typing a *k*.



こんにち

Figure 5.4: The series $[k,o,N,n,i,t,i]$ is translated in Japanese hiragana.

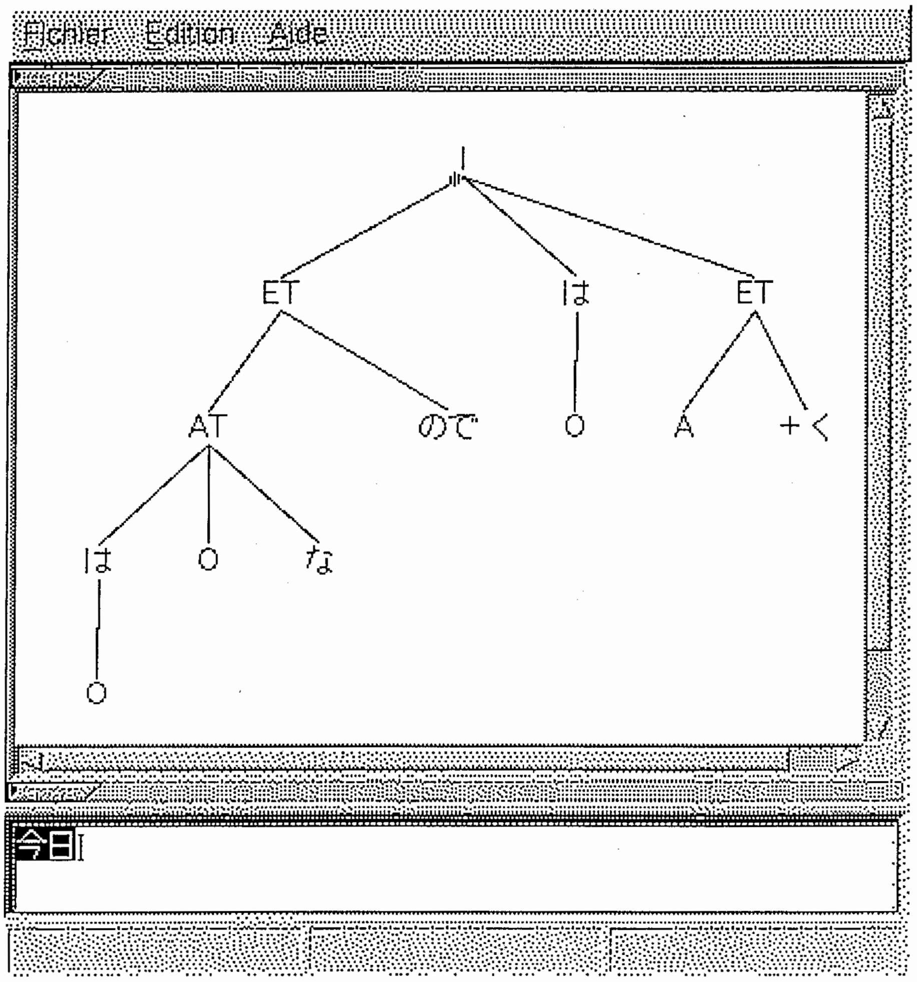
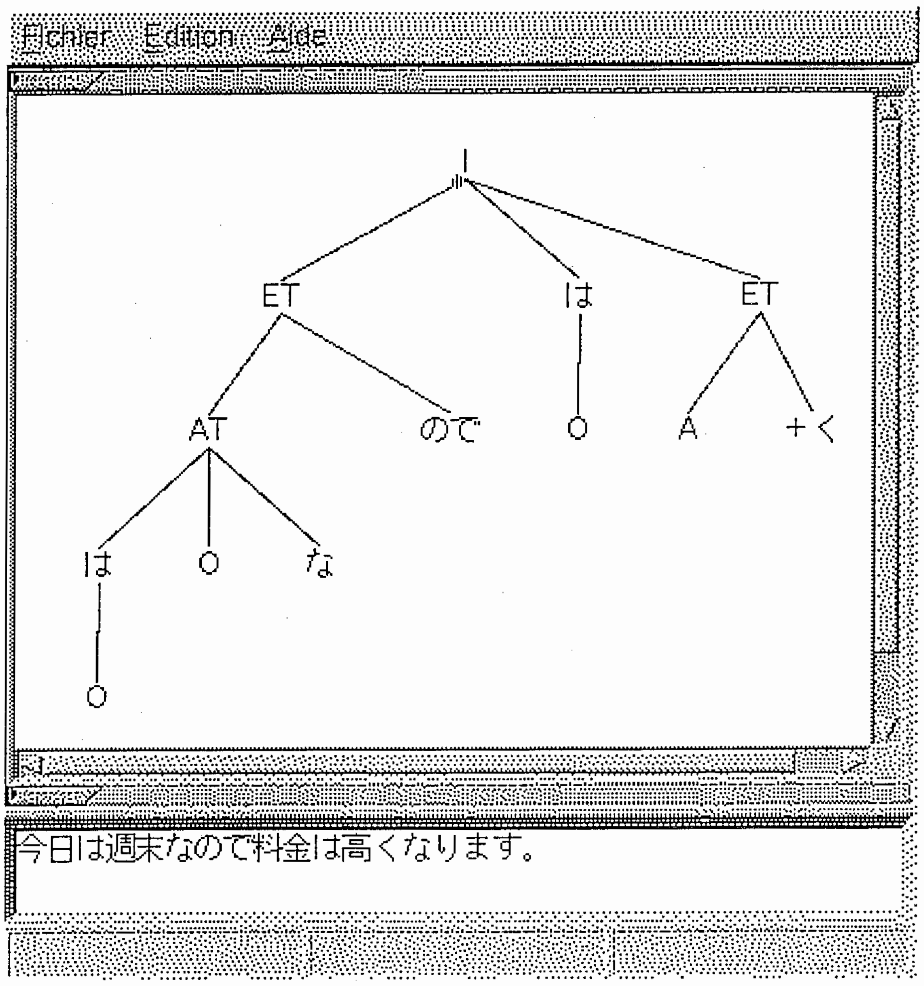


Figure 5.5: After selecting the kanji corresponding to *koNniti*.



今日は週末なので料金は高くなります。

Figure 5.6: After typing the entire sentence.

Chapter 6

Conclusion

A treebank is a corpus in which each sentence carries a linguistic description (in fact, a tree) input by hand by an indexer. The interest of such banks is undeniable as linguistic resources, and some statistical approaches in analysis already use treebanks, but the consistency of the data in treebanks is often problematic: similar portions of texts are sometimes assigned different structures. This is a particularly sensitive point if these data are to be used by statistic models.

The construction of a treebank and the verification of its consistency are also very cumbersome and time-consuming processes. To speed them up, we have proposed a tool, an editor with extra functionality. In order to speed up the creation of new data in the treebank, our tool proposed simple and fast editing facilities, on texts, as well as on trees. Also, in order to increase the consistency of the new data that is created, our tool contains some searching methods and parsing aids.

The tool we proposed is `Boardedit`, a multilingual board editor. A board is the pair constituted by a text and its linguistic structure, generally a tree. We have made a complete implementation of `Boardedit` from specifications. We have implemented all the edit facilities and proposed a plug-in system to easily integrate new parsing aids in the future. `Boardedit` is an internationalised software: it handles many language character code sets by using locale specifications.

We have implemented an independent complete MVC model for the tree editing facilities before porting into `Boardedit` as well into other applications like ATRForestedit or `TreeCanvas`.

- ATRForestedit is designed to work under MS-Windows. It enables a user to display and manipulate a forest from inside an application. It is composed of two different modules: an ActiveX control for developpers and an ActiveX document for end-users. These two modules are runnable and insertable under any MS-Windows application. The base of ATRForestedit was implemented and demonstrated to the ATR Intellectual Property Support section. It could become a commercial software with a patent. However, everything is yet to be decided.

- `TreeCanvas` is designed for UNIX and is already available for internal use in ATR–ITL. It is a filter between the standard input and the standard output. It recognizes several input formats and outputs different formats. Particularly, it offers a nice filter to output trees in LaTeX format which may be of interest for the research community which uses mainly LaTeX.

As for the future, a complete implementation of ATRForestedit is still to be done. Also, during our final talk, we received some feedback suggesting that `TreeCanvas` (our MVC model for the tree editing facilities) could be ported to some other UNIX environments (SunOs, Linux, ...) In this respect, we could envision finding another graphical user interface development environment to make the port of `Boardedit` easier for other UNIX workstations.

Our final hope is that `Boardedit` will integrate more parsing aids in the future, and that it will be of use for expanding the ATR-NEC treebank.

Bibliography

[Black et al. 96] Ezra Black, Stephen Eubank, Kashioka Hideki, David Magerman, Roger Garside and Geoffrey Leech
Beyond Skeleton Parsing: Producing a Comprehensive Large–Scale General–English Treebank with Full Grammatical Analysis
Proceedings of COLING-96, Copenhagen, August 1996, pp. 107-112.

[Boitet and Zaharin 88] Christian Boitet and Zaharin Yusoff
Representation trees and string-tree correspondences
Proceedings of COLING-88, Budapest, 1988, pp 59-64.

[Goh 96] Goh Chooi Ling
Penyunting Papan (Board Editor)
Projek tahun akhir, Pusat Pengajian Sains Komputer, Universiti Sains Malaysia, 1996.

[Itkonen 94] Esa Itkonen
Iconicity, analogy, and universal grammar
Journal of Pragmatics, 1994, vol. 22, pp. 37-53.

[Kawata *et al.* 98] 河田康裕、金城由美子、柏岡秀紀
日本語会話文の構文木付コーパス作成
言語処理学会第 4 回年次大会, 九州大学, 1998 年 3 月, pp. 622-625.

[Lepage 89] Yves Lepage
Un système de grammaires correspondancielles d'identification
Thèse, Université de Grenoble, juin 1989.

[Lepage 94] Yves Lepage
Texts and Structures – Pattern-matching and Distances
ATR report TR-IT-0049, Kyoto, March 1994.

[Lepage 97] Yves Lepage
String Approximate Pattern-Matching
55th Meeting of the Information Processing Society of Japan, Fukuoka, August 1997, vol. 3, pp. 139-140.

[Lepage 92a] Yves Lepage
Easier C programming
Input/output facilities
ATR report TR-I-0293, Kyoto, November 1992.

[Lepage 92b] Yves Lepage
Easier C programming
Some useful objects
ATR report TR-I-0294, Kyoto, November 1992.

[Lepage 96] Yves Lepage
Tesnière structural syntax: notations for tree-banking using BoardEdit
ATR report TR-IT-0176, Kyoto, July 1996.

[Lepage & Ando 96] Yves Lepage & Ando Shin-Ichi
Un éditeur pour la construction de banques d'arbres
Actes de TALN-96, Marseille, mai 1996, pp. 104-111.

[Lepage 91] Yves Lepage
Parsing and Generating Context-Sensitive Languages with Correspondence Identification Grammars
Proceedings of the Pacific Rim Symposium on Natural Language Processing, Singapore, November 1991, pp. 256-263.

[Saussure 16] Ferdinand de Saussure
Cours de linguistique générale
publié par Charles Bally et Albert Sechehaye, Payot, Lausanne et Paris, 1916.

[Smart 92] Julian Smart
wxWindows, a multi platform GUIDE
Artificial Intelligence Applications Institute, University of Edinburgh, 1992.

[Tang 96] Tang Eny Kong
Interactive Disambiguation in Multilevel Parallel Texts Alignment – towards the Construction of a Bilingual Knowledge Bank
Proceedings of MIDDIM-96, post–COLING seminar on interactive desambiguation, Christian Boitet ed., August 1996, pp. 101–106.

[Winograd 83] Terry Winograd
Language as a cognitive process
vol.1 Syntax
Addison Wesley, 1983.

[Wu & Manber 92] Sun Wu & Udi Manber
Fast Text Searching Allowing Errors
Communications of the ACM, Vol. 35, No. 10, October 1992, pp. 83-91.

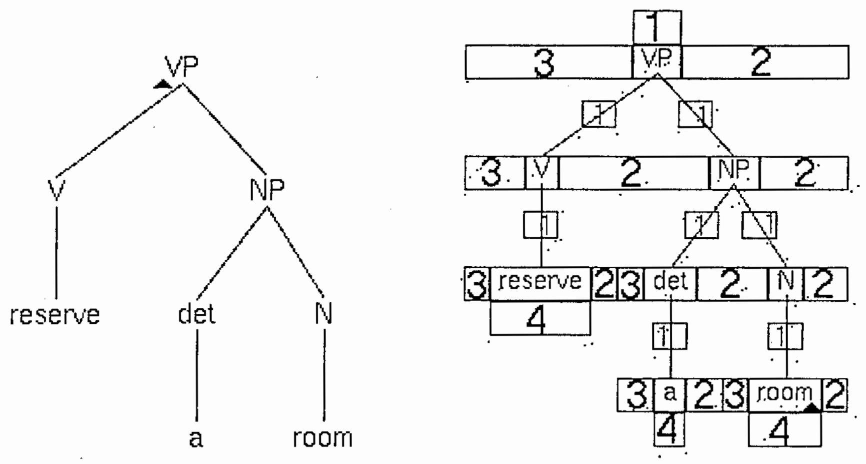
[Zaharin 90] Zaharin Yusoff
Generation of synthesis programs in ROBRA (ARIANE) from String-Tree Correspondence Grammars (or a strategy for synthesis in machine translation)
Proceedings of COLING-90, Helsinki, 1990, vol 2, pp 425-430.

[Zaharin and Lepage 92] Zaharin Yusoff and Yves Lepage
On the specification of abstract linguistic structures in formalisms for Machine Translation
Proceedings of the International Symposium on Natural Language Understanding and AI, pp 145-153, Iizuka, July 1992.

[Zaharin 87] Zaharin Yusoff
String-Tree Correspondence Grammar: a declarative formalism for defining the correspondence between strings of terms and tree structures
Proceedings of the 3rd Conference of the European Chapter of ACL, Copenhagen, 1987, pp 160-166.

Appendix A

Sensitive areas in tree editing



Those sensistive areas allow the user to insert nodes by pointing with the mouse. They are attached to a node and separated into four categories, designated by their respective number:

1. mother node area,

2. right node area,

3. left node area,

4. daughter node area, and

When clicking on any of these areas, a new node is created with an empty label (edit transaction see Section 3.2.1). The user knows that the pointer enters one of these sensitive areas because the cursor changes its form from an arrow to a '+' sign.

Of course, nodes are also sensitive areas for the label editing mode (see section 3.2).

Appendix B

Boardedit as a command line

This appendix presents the command line options of `Boardedit`. With the command line, the user can set all the parameters of the Find dialog box and open a file of boards. The command line options were very useful during our experimentation of `Boardedit`.

```
use:  Boardedit [-<n>] [-use=[exact|closest|match|analogy]]
                [-base=<filename>[,B<n>]]  [<filename>]
design: Yves Lepage
implementation: Nicolas Auclerc  (1999)

default: Boardedit  means Boardedit -use=exact
         (granularity is Latin characters)

job: apply the find method for the board given on standard input
     using the file of boards <filename> (one board on each line)
     as a base file.

options:
        -base=<filename>:  set <filename> as the base file
             ,B<n>:        <filename> contains <n>-byte texts
                           (granularity)

        -use=exact: select exact pattern matching
        -use=closest: select closest pattern matching
        -use=match: select approximate pattern matching
        -use=analogy: select analysis by analogy
        -<n>: threshold for approximate matching
```

Examples Here are some examples of the use of `Boardedit` with command line options.

`Boardedit`

This example is the default. Launch `Boardedit` without setting the Find dialog box and does not open any file of boards.

`Boardedit nec.board`

This opens the file of boards called `nec.board`.

`Boardedit -base=nec.base,B2`

This sets the treebank as being `nec.base` and sets the granularity to Japanese characters.

`Boardedit -3 -use=match -base=nec.base,B2 nec.board`

This sets the treebank as being `nec.base` and sets the granularity to Japanese characters. `Boardedit` will open the file of boards `nec.board`. The search method selected will be approximate matching and the threshold is set to 3.

Appendix C

ATRForestedit: a screen shot

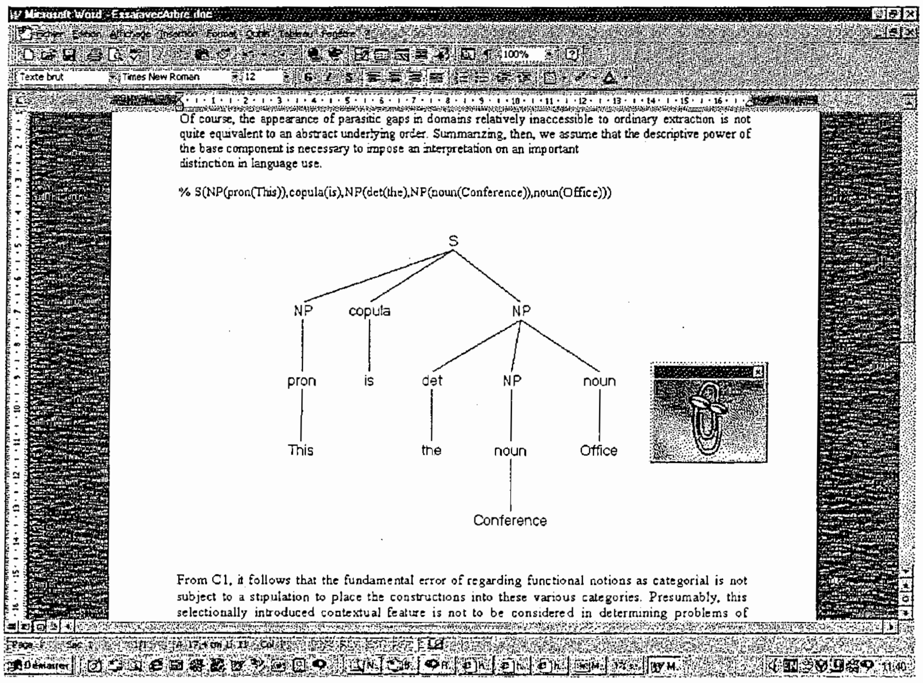
Our wxForestedit component could be useful to other researchers. To make it available, we have proposed to port it to the PC MS-Windows environment, under the form of an ActiveX component. ActiveX is a set of technologies introduced by Microsoft for MS-Windows.

Because we distinguish between two different types of potential users, we proposed, and started to implement, two modules:

- An ActiveX control (OCX) for developers. ActiveX controls are interactive objects which can be inserted in any programming development kit like MS Studio or Borland Delphi. An ActiveX control proposes to developers a list of methods, properties and events to manage a component. It is an interface to an object, hence it is not directly a runnable software application.

- An ActiveX document for end-users. An ActiveX document is a standalone application and a document insertable under any MS-Windows application which supports ActiveX, making it possible to work on a forest directly.

Figure C.1 shows a screen shot of ATRForestedit under a MS-Windows Application: MS Word 97.


Of course, the appearance of parasitic gaps in domains relatively inaccessible to ordinary extraction is not quite equivalent to an abstract underlying order. Summarizing, then, we assume that the descriptive power of the base component is necessary to impose an interpretation on an important
distinction in language use.

% S(NP(pron(This)),copula(is),NP(det(the),NP(noun(Conference)),noun(Office)))

From C1, it follows that the fundamental error of regarding functional notions as categorial is not subject to a stipulation to place the constructions into these various categories. Presumably, this selectionally introduced contextual feature is not to be considered in determining problems of

Figure C.1: ATRForestedit screen shot

Appendix D

TreeCanvas: a simplified user's manual

TreeCanvas is a by-product of the Boardedit tree editing facilities (see section 2.2.1). It was implemented to develop and test all the tree editing facilities. It is already available for internal use in ATR–ITL. There was also a previous version of TreeCanvas (see section 2.3). It is designed to help create or modify trees. It can handle, input, and output, many different formats. TreeCanvas is a filter between the standard input and the standard output. When the TreeCanvas application is closed, it will output the edited trees in the standard output.

In this appendix, we will show the prompt options of TreeCanvas, the supported trees formats, and particularly an example of a LaTeX output.

D.1 Prompt option

```
use: TreeCanvas [-] [ -T | -D | -F | -bra | -TeX | -\\ ]
design: Yves Lepage
implementation: Nicolas Auclerc  (1999)

job: open a tree canvas with an empty forest (unless option - is given),
     and write the edited forest in the standard output
        on exiting the canvas
```

```
options:
   -:          read a forest from standard input
                (feature structure,
               parenthesised form (with or without intervals),
               drawn form,
               bracketed form)
   -T:         output is parenthesised form on one line (default)
   -D:         output is drawn form
   -F:         output is feature structure
   -bra:       output is constituent bracketing convention
   -\\, -TeX: output is LaTeX (antree.sty package of LaTeX2e)
```

D.2 Supported tree formats

For example, we are going to ask `TreeCanvas` to input a tree in parenthesised format and to output it in another format. Here is an example of a tree in paranthesised form: `VP(V(reserve),NP(det(a),N(room)))` . Figure D.1 shows its drawn form.

D.2.1 –F: feature structure

```
[[VP [[V reserve][NP [[det a][N room]]]]]]
```

D.2.2 –D: drawn form

```
       VP
  _____|___
  V       NP
  |      ___|_
reserve det   N
         |    |
         a  room
```

D.2.3 –bra: consituent bracketing convention

```
[VP [V reserve V] [NP [det a det] [N room N] NP] VP]
```

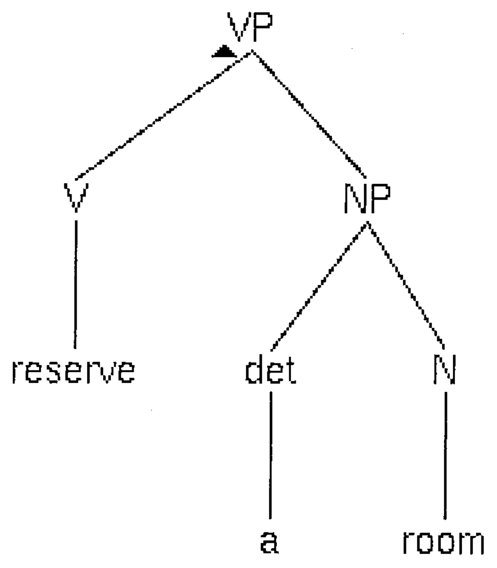


Figure D.1: Graphical display of a tree in `TreeCanvas`

This format is used in the ATR–Lancaster treebank [Black et al. 96] (see section 2.1).

D.2.4 –TeX: LaTeX

```
% Use with: \usepackage{antree}
\begin{antree}{VP}
   \link{\begin{node}{V}
      \link{\leaf{reserve}}
      \end{node}}
   \link{\begin{node}{NP}
      \link{\begin{node}{det}
         \link{\leaf{a}}
         \end{node}}
      \link{\begin{node}{N}
         \link{\leaf{room}}
         \end{node}}
```

```
        \end{node}}
\end{antree}
```

Figure D.2 shows the result of the compilation by LaTeX of the previous text. This is uses the special *antree* style.

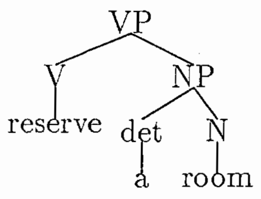


Figure D.2: Result obtained by the LaTeX antree style

Index