TreeForm: Explaining and exploring grammar through syntax trees

Donald Derrick
University of British Columbia, Vancouver, Canada

Daniel Archambault
INRIA Bordeaux Sud-Ouest, France

Correspondence:
Donald Derrick,
University of British Columbia,
UBC Department of Linguistics,
2613 West Mall,
Vancouver, BC,
Canada,
V6T 1Z4.
E-mail:
dderrick@interchange.ubc.ca

Abstract

Linguists studying grammar often describe their models using a syntax tree. Drawing a syntax tree involves the depiction of a rooted tree with additional syntactic features using specific domain conventions. TreeForm assists users in developing syntax trees, complete with movement lines, coreference, and feature association, in order to explore their syntactic theories and explain them to their colleagues. It is a drag-and-drop alternative to LaTeX and labelled bracket notation tools already available, which many linguists find difficult to use. We compare the output of TreeForm to those existing tools and show that it is able to better respect the conventions of the domain. We assess how easily linguists learn to use TreeForm through a series of cognitive walkthroughs. Our reviews find that TreeForm is a viable alternative to existing tools.

1 Introduction

Linguists studying natural language syntax, semantics, and morphology often describe their models using rooted trees, which linguists call parse trees or concrete syntax trees. Syntax trees provide an aesthetically pleasing way of demonstrating the structure of grammar. These trees can exhibit structural ambiguities in sentence and word meaning, and illustrate the composition of words, phrases, and sentences from their constituent components. Syntax trees may also exhibit movement, coreference, and feature association.

A tree can be drawn by hand quickly, but it is difficult to fix mistakes and these trees are unsuitable for publication. As a result, linguists sometimes use general purpose drawing tools, which is an extremely slow process but can produce the best results. Most often, linguists draw trees using word processors with special fonts to draw tree lines and triangles.

LaTeX, an open-source typesetting tool, also has many tree drawing macros that can be used to produce syntax trees. These macros can be effective, but they have steep learning curves because they require a linear encoding of syntax trees and do not allow for direct manipulation. Also, many linguists are not LaTeX users and have little or no experience coding.

Some computer tools allow users to interact directly with their syntax trees. However, they have restrictions that limit the complexity of trees that can be produced. Additionally, most add an extra step by requiring the user to input a set of rules from which visual tree structures can be computed.

In this article, we present the design and evaluation of TreeForm: a drag-and-drop tool for building syntax trees, which has been available as an open source project since February 2006 and downloaded approximately 18,000 times as of February 2009. TreeForm uses established graph drawing algorithms combined with representations for syntactic features. These syntactic features are not handled directly through standard graph drawing algorithms.

We begin by discussing previous work in Section 2, followed by a description of the TreeForm interface in Section 3 and the algorithms of the system in Section 4. We then compare TreeForm to other syntax drawing tools in Section 5 and present two sets of user interviews, or cognitive walkthroughs, in Section 6.

2 Previous Work

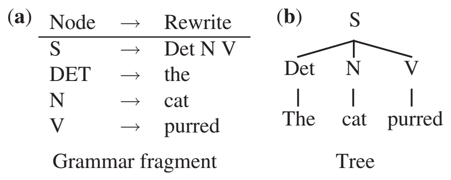
Previous work begins with a summary of the theory behind syntax trees, previous work in graph drawing, and a quick overview of other syntax tree drawing tools.

2.1 Linguistic syntax trees

Generative grammars include several theories that employ syntax trees, beginning with the Chomsky's *Aspects of the Theory of Syntax* (Chomsky, 1965). A 'generative grammar' is the full set of rewrite rules for a language. 'Rewrite rules' are rules for constructing sentences. A 'grammar fragment' is a sufficient enough subset of these rewrite rules to allow construction of a specific sentence. A 'syntax tree' is a rooted tree that illustrates a particular usage of a grammar fragment, as represented by Fig. 1.

Syntax trees have three common structures. 'Non-terminal' nodes are internal nodes of the tree. 'Terminal' nodes are leaves in the tree. Non-terminal nodes appear on both sides of rewrite rules and must be rewritten until all the outputs are terminal nodes. 'Triangles' encode subtree structure between a non-terminal and a set of terminal nodes. Triangles are used to abbreviate the size of trees or indicate that the structure below the triangle is still unknown.

All syntax trees can also be represented using labelled bracket notation. 'Labelled brackets' are



(a)

Node	→	Rewrite
S	→	Det N V
DET	→	the
N	→	cat
V	→	purred

Grammar fragment

Fig. 1 Grammar fragment and corresponding (flat) tree. S is sentence. DET is a determiner (article). N is a noun. V is a verb

[S [Det [the]] [N [cat]] [V [purred]]]

Fig 2 Labelled bracket notation for the sentence 'The cat purred'

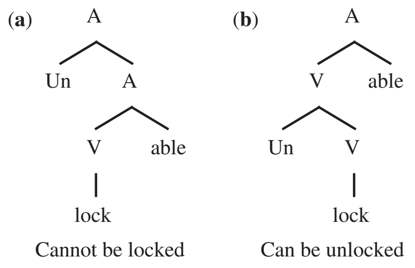


Fig. 3 Two meanings for the word 'unlockable'. A is an adjective

way of representing tree structures and levels using nesting, as seen in Fig. 2. Labelled brackets are compact, but they become increasingly hard to read as syntax trees become more complex.

By convention, syntax trees should be compact, as symmetrical as possible, and all the lines to children should originate under the centre of a parent. Syntax trees are not limited to sentences, but apply to linguistic units of all ranks including word morphology. Different tree structures can encode the evidence of different meanings for the same output, as in the word 'unlockable' in Fig. 3 (Aronoff and Fudeman, 2005). (Unless otherwise stated, all figures of syntax trees in this article have been produced with TreeForm.)

Grammar trees also need to represent the movement of nodes in different levels of linguistic representation. 'Movement' is a theoretical representation of differences between an underlying

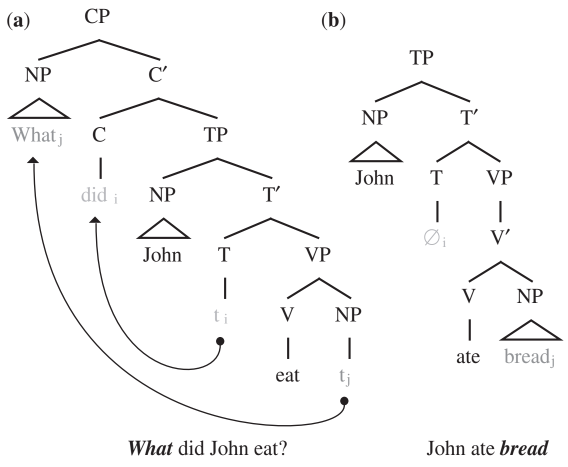


Fig. 4 Syntax tree illustrating movement in a question and its corresponding answer. C is a complementizer, T is (verb) tense. (X)P is a phrase such that NP is a noun-phrase. (X)′ is a bar such that T′ is a tense-bar. Bar is phrase that is a child of a phrase of the same type

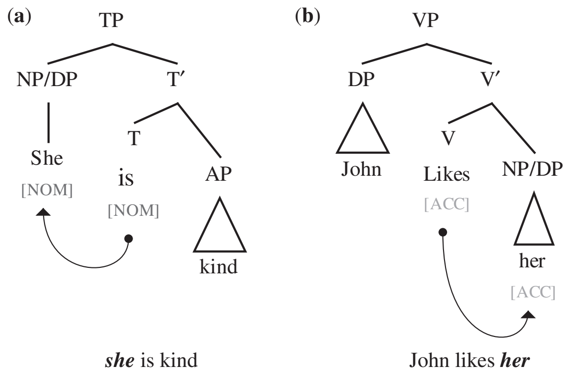


Fig. 5 Syntax tree for nominative case **he** versus accusative case **him**

sentence structure and what you hear. Movement is commonly represented by subscripts and movement lines, as seen in Fig. 4 (Carnie, 2002) [Colour based on Stone (2006)].

Grammar theories also require the association of features, such as case, to explain why the syntactic structure looks the way it does, as shown in Fig. 5 (Carnie, 2002).

Features of this type include 'case', as shown above, 'theta roles' (Θ), which link objects and subjects to their verbs, and what are generically called 'features', which are used for many reasons, including explaining why WH phrases move to the front of sentences in English questions. Features link nodes in a tree through feature 'association', which is indicated by having the same text for features under two nodes or with subscripts. Any of these feature types may be combined in the same syntax tree. We show a syntax tree based on 'government and binding' theory (GB), with colour highlighting related concepts, in Fig. 6.

These syntax trees are used in many generative grammar theories, and include GB, the minimalist program (Webelhuth, 1995), lexical-functional grammar (Bresnan, 2001), and generalized phrase structure grammar (Gazdar *et al.*, 1985), and all of which use syntax trees to illustrate theoretical insights. Some formal syntactic theories, such as head driven phrase structure grammar use attribute value matrices (AVMs) to represent grammar (Pollard and Sag, 1994). AVMs can sometimes be placed into syntax trees, but they produce nodes containing complex formulas whose generation is beyond the scope of TreeForm.

2.2 Computational tree drawing

Tree drawing has been studied extensively in graph drawing and information visualization for years (Di Battista *et al.*, 1999; Herman *et al.*, 2000). Various methods exist for automatically depicting trees, such as H-trees (Shiloach, 1976), radial trees (Eades, 1992), cone trees (Robertson, *et al.*, 1991), balloon views (Carrière and Kazman, 1995; Teoh and Ma, 2002; Grivet *et al.*, 2004), phylogenetic trees (Munzner *et al.*, 2003), and tree maps (Johnson and Shneiderman, 1991). However, as demonstrated in the introduction, linguists from the generative grammar tradition usually want rooted trees that illustrate hierarchical structure top-down, which these algorithms do not produce.

Syntax tree drawings look similar to the rooted tree drawings of the Buchheim–Walker algorithms for tree drawing (Reingold and Tilford, 1981; Walker, 1990; Buchheim *et al.*, 2002). The diagrams have a clear root and a clear depiction of the hierarchy in the generative grammar tradition.

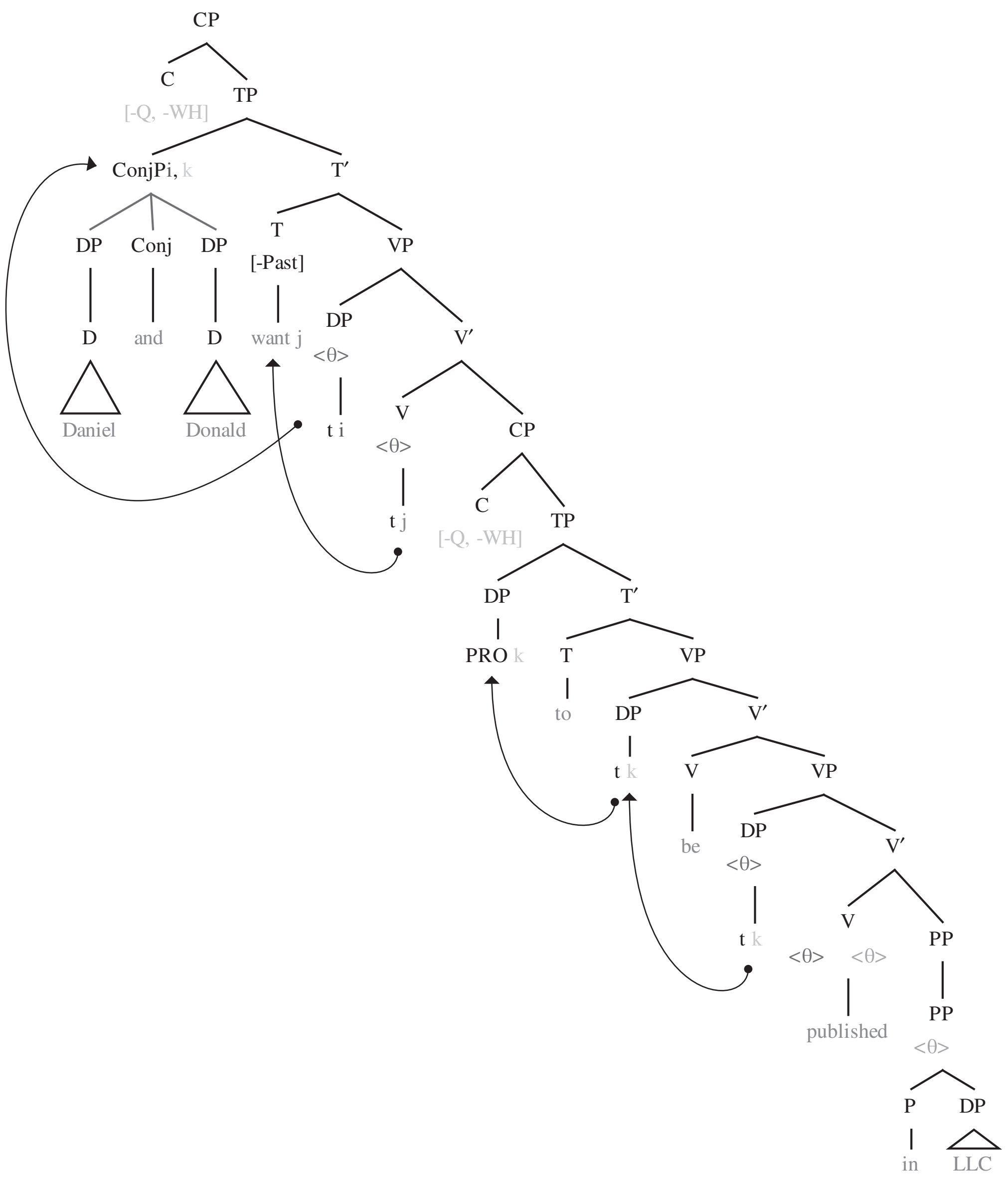


Fig. 6 GB tree of the sentence 'Daniel and Donald want to be published in LLC'

Also, many of the aesthetic criteria described in Buchheim (Buchheim *et al.*, 2002) are similar to those set out in Section 2.1.

While these classic tree drawing algorithms follow the conventions preferred by linguists, it is still unclear how to integrate them into a system useful for drawing syntax trees, which supports features, such as syntactic movement, feature association, unknown syntactic structure, and multi-lingual support.

2.3 Syntax tree drawing tools

Fonts, such as Arboreal (Cascadilla Press, 2008a) and Moraic (Cascadilla Press, 2008b), are the most common way to produce syntax trees. They allow a user to enter tree lines as specialized font characters

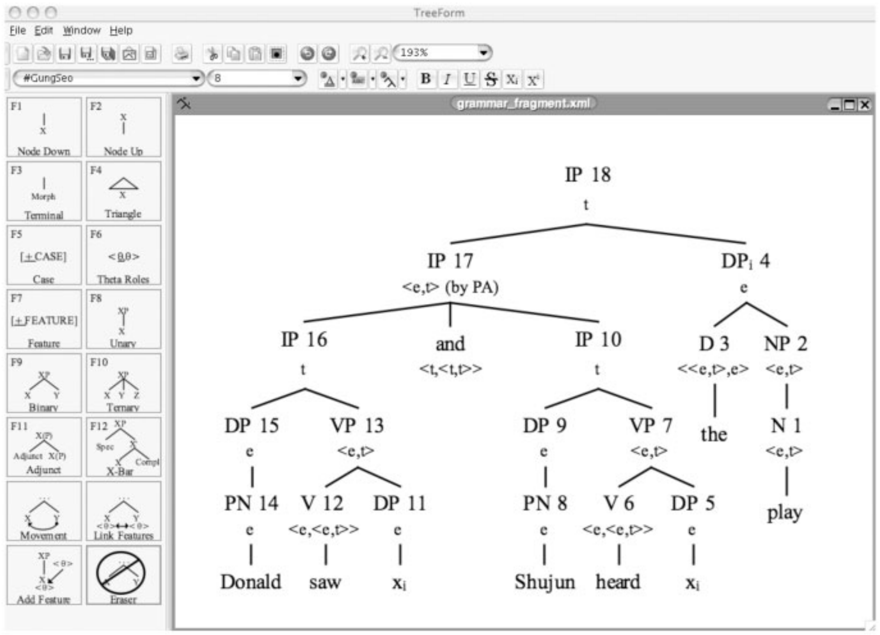


Fig. 7 Screen capture of the final interface for TreeForm

into a word processor document. They are often easier to use than generalized drawing tools and can be entered directly into documents. Linguists like this method because they already use specialized fonts for the International Phonetic Alphabet (IPA) and non-European orthographies. However, this is a time-consuming and error-prone process as users have to position structures in their trees manually.

Users of LaTeX have access to a wide variety of automatic syntax tree drawing macros including qtree (Siskind and Dimitriadis, 2006) and synttree (van Zulijen, 2005). These tools do not have graphical user interfaces, but allow linguists to design trees using a modification of labelled brackets that are translated into a syntax tree. Qtree also allows the addition of movement lines, with control over the position of start and end points around a node, and details of line style beyond what TreeForm can currently accomplish. However, many linguists are not LaTeX users and since qtree and synttree both use labelled brackets, they become difficult to use as trees become more complex. Example code for qtree is shown in Fig. 9.

LaTeX users can also use xyLing (Vogel, 2004) and jTree (Frampton, 2006). Neither tool has an automatic tree layout algorithm; the user has to manually compute the angle and spacing of all node lines. Users can produce incredibly complex trees with movement lines, features, and even tables or formulas in the place of nodes. However, they must manually position everything. Frequently users resort to drawing the entire tree on graph paper in advance so that they can identify the coordinates of tree nodes.

Kosko's tree drawing program (Kostko, 1994) and several web-based tools allow the user to submit simpler labelled brackets, returning syntax trees that can be saved as pictures or pasted into word-processing documents. These tools include phpSyntaxTree (Eisenbach and Eisenbach, 2006), RSyntaxTree (Hasebe, 2007), and Syntax Tree Drawer in SVG (Ruter, 2004). All of them use labelled brackets with roman text only.

Users who want a graphical environment can use a general purpose drawing program or one of several customized tree drawing tools. Some of the best of these tools, like XTAG (Parubek *et al.*, 1992) are specialized for particular languages. Computers running XTAG require UNIX, LISP, and a login to the source site. Like the LaTeX macros, XTAG uses special tags to generate the syntax trees. XTAG is for English only, and the user needs to know the internal grammar rules such as those shown in Fig. 1, but only in more detail.

Other graphical tools also require a set of grammar rewrite rules as input. These include the Syntax Student's Companion (SSC) (Max, 2004) and the commercial product Trees 2/3 (Crist and Kroch, 2005). They both produce elegant trees, but limit the node text and in the case of SSC will not export high-resolution images.

A commercial product, TreeBuilder (Marik and Dever, 2004), is currently undergoing beta testing. TreeBuilder uses some of the principles developed independently for TreeForm, including the ability to draw trees without previously defined rewrite rules. It allows the addition of movement lines, including the movement of subtrees, or 'snowball raising'. The user can toggle all terminal nodes to appear at the bottom the tree, and toggle the removal of lines between terminals and their parents. These last three features are not yet supported by TreeForm.

However, the system does not allow direct addition of subtrees or editing of node labels, and is instead driven by menus, toolbars, and pop-up menus. It displays trees in a similar fashion to synttree as seen in Fig. 8d. At this time, both the code and the output of TreeBuilder is proprietary.

2.4 Formal evaluation

Systems such as XTAG (Doran *et al.*, 1994) have been evaluated for content, but very few syntax drawing tools have been formally evaluated for usability. The SSC was evaluated for COLING 2004 through public feedback resulting from users downloading and testing the software in teaching and presentation contexts. Listings of tree drawing tools also exist on publicly accessible sites such as Linguist (The Linguist List, 2007).

3 Interface

TreeForm was implemented in Java using Swing. The design philosophy was based on the idea of presenting a graphical method for building syntax trees without obligatory rewrite rules, and providing maximum flexibility for changing the shape of the tree as it is being built. Users are constrained to build one rooted tree at a time. TreeForm supports multi-lingual syntax trees and provides a simple drag-and-drop metaphor for construction of syntax trees.

TreeForm has a multiple document interface with icons for syntactic structures in an object browser on the left side, toolbars for all the non-contextual options on top, and a menu bar duplicating toolbar options. The text in tree structures can be edited as one would in a word processor. The final interface is shown in Fig. 7.

The object browser contains well-known syntactic structures. These objects can be dragged into the document pane to add to an existing tree or to begin drawing a new tree. A subtree can also be shift-dragged onto another part of the tree to reshape the tree. Highlights indicate where nodes may be dropped into place when modifying trees.

When adding to or moving subtrees, if the chosen position already has one or more daughter nodes, a series of filled circles appears between the daughter nodes. Moving the mouse over the circles changes the highlighted circle, and clicking the mouse positions the subtree in that place.

Movement lines can be added by dragging a node over another node. The shape of the line may be altered through two control points. User control point positions are overridden if the tree structure changes.

Users may also add any number of case, feature, and theta roles to the tree by dragging and dropping these onto tree nodes. A 'clone' of any feature, called an association, may be added to any other node by dragging any association onto any node. These duplicates always have the same text as the original.

Users can copy a tree to the computer's clipboard and paste it into an editing program or word processor. They can also print their trees to a printer, or directly to a portable document file (PDF). The PDF can then be incorporated into a word processor or LaTeX file. The PDF files are produced as vector graphics, and therefore maintain a smooth, high-quality appearance even when printed on large posters.

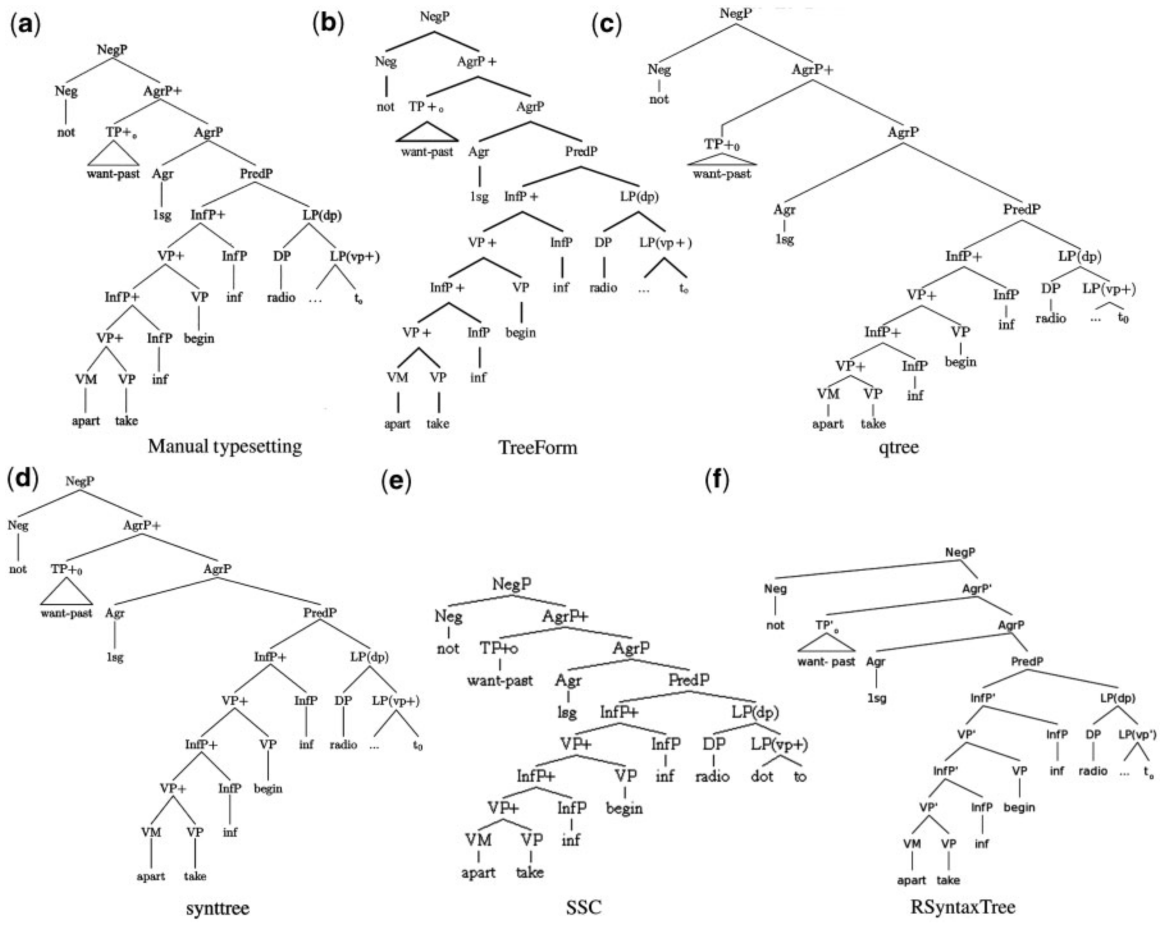


Fig. 8 Manual typesetting and TreeForm versus other tree drawing tools. Subfigure (a) reproduced with permission from 'Verbal Complexes', Tree 5.1, Koopman and Szabolcsi (2000)

4 Algorithms

TreeForm draws its trees using an implementation of the Buchheim–Walker algorithm (2002). This algorithm displays trees in linear time, and require only slight modification to work with nodes of varying size and heights to accommodate the differing text sizes present in node labels.

Buchheim–Walker traverses the tree in post-order, spacing sibling leaf nodes evenly and then centring them under parent nodes. To keep subtrees from overlapping, pointers follow the inside and outside contours of the left and right subtree up to the highest common level. Whenever there is a conflict in position, the left greatest distinct ancestor is used to shift the subtrees and prepare to shift smaller subtrees. After the first traversal is complete, the tree is traversed in pre-order to determine the final position of the nodes.

Line movement is implemented using cubic Bezier curves. These curves connect any two nodes in the syntax tree. Default positions for control points are determined as follows. If both nodes are on far right or left branches of the tree, the line is looped over the top of the tree. If not, the tree is traversed from the highest of the two nodes to the lowest by walking down the right and left side of the tree.

```
\Tree
[.CP C\\$[$-Q,-WH$]$
    [.TP
        [.T′
            [.T\\$[$+past$]$\\$[$NOM$]$ ]
            [.VP
                [.VP \qroof{\node{donald}Donald}.DP$_j$\\$[$NOM$]$
                    [.V′
                        [.V \node{met}Met$_i$\\$[$+past$]$ ]
                        [.vP
                            [.DP\\$<\Theta>$ \node{tj}t~$_j$ ]
                            [.v′
                                [.v\\$[$ACC$]$\\$<\Theta>$$<\Theta>$ \node{ti}t~$_i$ ]
                                \qroof{Dustin}.DP\\$[$ACC$]$\\$<\Theta>$
                            ]
                        ]
                    ]
                ]
                [.AP
                    [.A′ today ]
                ]
            ]
        ]
    ]
]
\anodecurve[bl]{tj}[bl]{donald}{.8in}
\anodecurve[bl]{ti}[bl]{met}{.8in}
```

Fig. 9 LaTeX code for qtree tree shown in Fig. 10b

If the bottom node matches the top node, or there are no tree lines between the nodes, nodes are connected directly. Otherwise the nodes are connected by looping the Bezier curve down to the bottom of the subtree. The user can adjust the shape of the curve by dragging the two control points manually.

5 System Evaluation

We begin our evaluation with a comparison of the output of the best of each type of syntax tree drawing program with the output of TreeForm. These include the LaTeX macros qtree (Siskind and Dimitriadis, 2006) and synttree (van Zulijen, 2005), the online tree drawing utility RSyntaxTree Hasebe (2007) and the SSC (Max, 2004). These four packages were chosen from among the many automatic general syntax tree layout tools because they produce the most elegant and cross-linguistically usable output in their categories.

For our comparison, we selected as ground truth a manually typeset tree from Koopman and Szabolcsi's *Verbal Complexes* (2000). While the tree, shown in Fig. 8, does not have *n*-ary branching, movement lines or colour, it is sufficiently complex for our purposes. The tree was typeset by hand in the Times New Roman font with the 3B2 publishing toolbox by Asco Typesetters of Hong Kong and is of the highest quality (Asco Typesetters, 2007).

The manually typeset tree has an elegant tree display shape, is compact, and is identical to what one would expect from the Buchheim–Walker algorithm. The tree produced by TreeForm has almost identical tree and line spacing as in the manually typeset tree. In contrast, with complex examples like this, qtree, RsyntaxTree, and Syntree produce lopsided and non-compact trees.

Lastly, the SSC-produced trees are well balanced. However, SSC has no utility for exporting high-resolution images of produced trees and does not allow the inclusion of any special characters such as subscripts or ellipses.

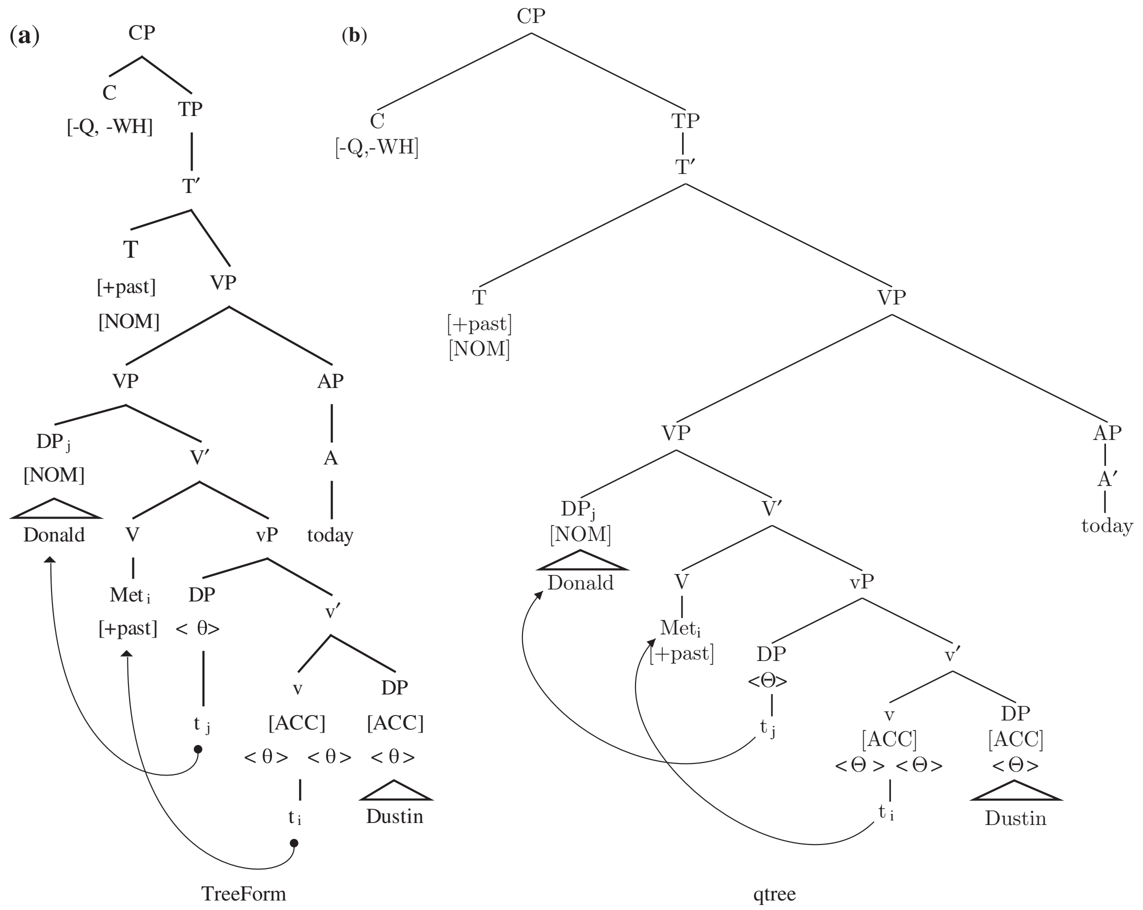


Fig. 10 Syntax tree for ‘Donald met Dustin today’ based on GB

5.1 Movement lines

We also present a comparison of TreeForm with qtree, the only other tool in this comparison that will represent movement lines. Qtree was written to work with tree-dvips, allowing users to write code attaching movement lines to nodes and bend the Bezier curve at one control point on the vertical axis.

The results work most of the time, but close examination shows that the arrow had to be pointed to the side of the word ‘met’, and even then the movement line crossed through the text [+past], as seen in Fig. 10b. Qtree lines do not allow for text below the nodes at the end points of the movement line, whereas TreeForm does. The code for this tree is complex and must be formatted exactly as shown in Fig. 9.

6 Cognitive Walkthroughs

TreeForm is a system that helps expert linguists to better understand and present their syntactic theories. Due to the exploratory nature of this process the goals and tasks of the user are not well defined.

In order to evaluate how linguists, who usually draw their syntax trees by hand and are not familiar with automated tools, use TreeForm, we adapt the

Allendoerfer *et al.*'s (2005) cognitive walkthrough technique for evaluating visualizations, beginning with a description of whom we believe are typical users.

6.1 Typical user

Users of TreeForm are formalist linguists or their students. They will come from a wide variety of grammar theory backgrounds. Most users will not have programming skills or experience with user interface or usability design. They will use computers at least once a week and will be familiar with word processing, e-mail, and Internet browsing tools. Users will most likely be familiar with the online syntax tree drawing tools for checking labelled bracket notation correctness and will be experts in the use of Arboreal fonts in word processors.

6.2 Goals and tasks

The user in our cognitive walkthrough represents a syntactic structure in a theoretical framework of their choice. In order to accomplish this goal, the user needs to undertake three tasks:

Build a syntax tree and modify its structure to better reflect the grammar.
Add syntactic features to the tree as appropriate for their theoretical framework of choice.
Modify tree structure and syntactic features until the resultant tree reflects the intended structure.

The user will be able to identify when the syntax tree represents their theory well, but frequently will not have a good idea of how the final tree will appear until complete. Therefore, they will most likely play with tree structure and features until it looks right.

6.3 User actions

Our users will be choosing both the syntactic structure and the theoretical framework. Therefore, a pre-scripted set of action sequences could not be written beforehand. As in the design of Allendoerfer *et al.* (2005), we asked ourselves, after each user action was taken, whether the users achieved their goals and if the action had the desired effect.

6.4 Participants

Six linguists participated in our first cognitive walkthrough, three graduate students and three professors. Two of the six users had experience with previous versions of TreeForm. Two came prepared to represent their own theories with syntax trees, while the other four produced trees loosely based on GB theory.

6.5 Data collection

The users were seated in front of a computer with TreeForm while we took notes. Users were notified of a list of software features available in TreeForm, but were given no explicit directions on how to use those features. The users were told that TreeForm was intended for both professionals and students, but were asked to focus on the evaluation of TreeForm as a tool for professionals. They were asked to build a syntax tree using features of the program as needed. We did not provide assistance or instruction for any of the actions unless they had completed the main task without using a feature or could not proceed. We noted which features the users found intuitive, and which features they needed help with. At the end of the cognitive walkthrough, we asked questions about the user's overall experiences with TreeForm and how they might use TreeForm in the future. We also asked about bugs, barriers to usability, and feature requests.

6.6 Results

The results showed that while users were all able to learn how to learn TreeForm, they found some features of the software to be unintuitive. Users had the most difficulty with drag-and-drop operations, hitting small targets, and using context menus.

The users examined the interface and noticed the object browser as it contained tree structures from generative grammar theory. They would click on a browser object and thereby build their first tree. After that, the four inexperienced users would click another browser object, and ask why nothing happened. They expected the same action to work a second time. Once we showed them how to drag and drop new nodes, they tended to do well, but two users had serious difficulties with hitting the small targets of nodes in the tree. Once users had seen drag-and-drop operations to build the tree, they tried to add syntactic structures and features in a similar fashion and were successful.

The users did not intuitively discover how to add movement lines or associations because these methods involved option keys and so did not fit the overall metaphor. Once shown how, the users succeeded in adding, changing, and deleting movement lines and associations.

When modifying tree structures, users typically used undo to remove changes they did not like and then dragged new nodes into the places where they were needed instead of reshaping already existing trees. While some users tried flipping the position of nodes with the context menu option, even the most experienced user did not initially try to move parts of the tree around in the editor. Five of the six users immediately succeeded in changing the text of nodes.

6.6.1 *User impressions*

All of the users liked the automatic tree display and movement line drawing. For this reason alone they all said they would use TreeForm in their research and publications. While there were usability issues, all the users liked the ideas behind the graphical user interface. Three of the users said they could imagine developing new syntactic theories using the software, one specifically saying, ‘I could see myself thinking with this’. The remaining three said they would not use computers for this purpose, preferring pen and paper.

About halfway through the cognitive walkthroughs, two users stopped and said, ‘I will use this for articles’. One participant liked the automatic layout enough that she wanted TreeForm to become the new standard for the linguistics journal she edited. The users also believed TreeForm would be useful for formal semantics and morphology in addition to formal syntax.

6.7 Addressing usability issues

Some users did not know how to begin building or reshaping their trees. In response, we added tooltips and button highlights to the object browser buttons. We also adjusted object browser labels and functions to better reflect conventions in linguistics.

All of the users had difficulty when adding new nodes or features to existing trees because they had trouble hitting small targets. We addressed this issue by highlighting the closest node to the cursor when adding or moving nodes or features in the tree thereby increasing ease of addition and tree modification. We termed this behavior ‘nearest neighbour’ functionality.

We also added object browser buttons for movement lines, associations, extra features, and subtree deletion to provide methods consistent with our metaphor for building trees. Features and associations now have linked highlighting between clones.

Colour selection was improved to make it more consistent with word-processing applications. Font, font sizes, and font effects could now be changed by highlighting parts of the tree and selecting options. Shift-highlighting now copies the selected part of the tree to the clipboard, similar to snapshot tools.

6.8 Follow-up review

After implementing the changes requested in Section 6.7, we reran the cognitive walkthroughs with three professional linguists. The users had a higher degree of success with the revised version of TreeForm. Users still had difficulty grasping the drag-and-drop metaphor, and had to be told to drag objects onto the editing pane. However, once they attempted drag-and-drop, the new nearest neighbour functionality eliminated failures in adding and modifying tree structures.

Users suggested implementing automatic panning for manipulating movement line shape, and several interface fixes based on standard keyboard commands from Mac OS X and Windows.

All three users said they would use TreeForm in articles and research. One professor was planning on using TreeForm for solutions to assignments as it would help him provide better feedback to students. Two of the users could imagine using TreeForm for exploring new syntactic theory.

7 Lessons Learned

Careful experimentation revealed the difficulties that the users had with interacting with TreeForm.

Small target sizes for drag and drop were not effective, and careful solutions needed to be implemented to enlarge target sizes as much as possible. Users will not search for options that require shift, option, or alt keys unless directions are supplied in an obvious manner. As a result, they will not succeed in using these options. As users have been trained to draw trees by hand, they need a consistent visual metaphor for all of their operations on the syntax tree.

Adoption of a new syntax tree drawing tool requires design to the highest standards. When linguists are willing to lay out graphs by hand, use graphics editors, or type complex LaTeX code, they expect thoroughly professional output in tree display, particularly for publications. Users need to display their theories correctly and in an visually appealing manner, following the conventions of the linguistics community.

8 Future Work

Since the cognitive walkthroughs were completed, TreeForm has better support for alternative input methods to make it easier for linguists to add special fonts. Linguists could use more types of movement types, such as snowball raising, which would involve the implementation of an algorithm for smallest enclosing ellipses (Gärtner and Schönherr, 1998). Control over the thickness and stippling of lines, colour of movement lines, position of the start and end point of lines between nodes, and other basic typesetting features would add to the aesthetic appeal of TreeForm, and make it more useful in high-grade publications.

While there are many tools that allow linguists to convert from labelled brackets to trees, adding this functionality to TreeForm would give linguists a one-stop place to build and assess linguistic theories.

Finally, we would also like to evaluate how TreeForm performs in an educational setting. TreeForm could use options to automatically generate a minimal set of rewrite rules for that tree in a text file, check for tree consistency with accepted linguistics theories, and default colour scheme for distinguishing terminals and non-terminals. When these changes are complete, we plan to test the usability of TreeForm in introductory and intermediate undergraduate linguistics classes, evaluating its effect on student and teacher performances.

Acknowledgements

The authors also wish to thank Melanie Tory for her help with understanding cognitive walkthroughs in visualization and help with editing this article, Guy Carden for his advice on presenting syntactic theories to general audiences, and Ciarán Llachlan Leavitt for editing. The authors also wish to thank the many users of TreeForm, who have faithfully e-mailed and posted requests for improvements to the interface and features of TreeForm.

References

Allendoerfer, K., Aluker, S., Panjwani G. *et al.* (2005). *Adapting the Cognitive Walkthrough Method to Assess the Usability of a Knowledge Domain Visualization. Proceedings of IEEE Symposium on Information Visualization (InfoVis '05)*, pp. 195–202.

Aronoff, M. and Fudeman, K. (2005). *What is Morphology. Fundamentals of Linguistics*. Malden, MA: Blackwell Publishing.

Asco Typesetters (2007). Asco typesetters. http://www.ascotype.com.hk/index1.html (accessed 19 September 2009).

Bresnan, J. (2001). *Lexical-Functional Syntax*. Malden, MA: Blackwell Publishing.

Buchheim, C., Jünger, M., and Leipert, S. (2002). Improving Walker's algorithm to run in linear time. In *Graph Drawing* '02, pp. 344–53.

Carnie, A. (2002). *Syntax, A Generative Introduction*. Malden, MA: Blackwell Publishing.

Carrière, J. and Kazman, R. (1995). *Research Report: Interacting with Huge Hierarchies Beyond Cone Trees. Proceedings of IEEE Information Visualiztion (InfoVis '95)*, pp. 74–81.

Cascadilla Press (2008a). Arboreal for Macintosh and Windows. http://www.cascadilla.com/arboreal.html (accessed 19 September 2009).

Cascadilla Press (2008b). Moraic for Macintosh and Windows. http://www.cascadilla.com/moraic.html (accessed 19 September 2009).

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.

Crist, S. and Kroch, T. (2005). Trees 2/3. http://lra.netfirms.com/Trees.htm (accessed 19 September 2009).

Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Upper Saddle River, NJ: Prentice Hall PTR.

Doran, C., Egedi, D., Hockey, B. A., Srinivas, B., and Zaidel, M. (1994). *XTAG system: A wide coverage grammar for English. Proceedings of the 15th conference on Computational linguistics*, volume 2, pp. 922–28.

Eades, P. (1992). Drawing free trees. *Bulletin of the Institute for Combinatorics and its applications*, pp. 10–36.

Eisenbach, M. and Eisenbach, A. (2006). phpSyntaxTree. http://ironcreek.net/phpsyntaxtree/ (accessed 19 September 2009).

Frampton, J. (2006). jTree. http://www.math.neu.edu/ling/tex/.

Gärtner, B. and Schönherr, S. (1998). Exact primitives for smallest enclosing ellipses. *Information Processing Letters*, **68**: 33–8.

Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Oxford, UK: Basil Blackwell.

Grivet, S., Auber, D., Domenger, J., and Melancon, G. (2004). Bubble Tree Drawing Algorithm. *International Conference on Computer Vision and Graphics*, 633–641.

Hasebe, Y. (2007). RSyntaxTree. http://www.yohasebe.com/rsyntaxtree/ (accessed 19 September 2009).

Herman, I., Melanon, G., and Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *Transactions of Visualization and Computer Graphics*, **6**(1): 24–43.

Johnson, B., and Shneiderman, B. (1991). *Tree-maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. Proceedings of the IEEE Visualization (Vis '91)*, pp. 275–82.

Koopman, H. and Szabolcsi, A. (2000). *Verbal Complexes*. Cambridge, MA: MIT Press.

Kostko, O. (1994). Kostko's tree drawing program. http://www.nyu.edu/pages/linguistics/workbook/kostko/ (accessed 19 September 2009).

Marik, L. and Dever, D. (2004). TreeBuilder. http://www.sittac.com/download/instructions.html (accessed 19 September 2009).

Max, A. (2004). The Syntax Student's Companion: An elearning tool designed for (computational) linguistics students. In Lemnitzer, L., Meurere, D., and Hinrichs, E. (eds), *COLING 2004 eLearning for Coputational Linguistics and Computational Linguistics for eLearning*, Geneva, Switzerland: COLING, pp. 71–8.

Munzner, T., Guimbretière, F., Tasiran, S., Zhang, L., and Zhou, Y. (2003). *TreeJuxtaposer: Scalable Tree Comparison using Focus+Context with Guaranteed Visibility. Proceedings of SIGGRAPH 2003*, pp. 453–62.

Parubek, P., Schabes, Y. and Joshi, A. K. (1992). *XTAG - A Graphical Workbench for Developing Tree-Adjoining Grammars. Proceedings of the Third Conference on Applied Natural Language Processing*, Trento, Italy: Association for Computational Linguistics, pp. 223–30.

Pollard, C. and Sag, I. A. (1994). *Head-driven Phrase Structure Grammar*. Chicago, IL: University of Chicago Press.

Reingold, E. M. and Tilford, J. S. (1981). Tider drawing of trees. *IEEE Trans. on Software Eng*, **7**(2): 223–8.

Robertson, G. G., Mackinlay, J. D., and Card, S. K. (1991). *Cone Trees: Animated 3D Visualizations of Hierarchical Information. Proceedings of Human Factors in Computing Systems (CHI '91)*, pp. 189–94.

Ruter, W. (2004). Syntax tree drawer in SVG. http://www.linguiste.org/syntax/tree/drawer/ (accessed 19 September 2009).

Shiloach, Y. (1976). *Arrangements of Planar Graphs on the Planar Lattices*. Ph.D. thesis, Weizmann Institute of Science.

Siskind, J. M. and Dimitriadis, A. (2006). The qtree package. http://www.essex.ac.uk/linguistics/clmt/latex4ling/trees/qtree/ (accessed 19 September 2009).

Stone, M. (2006). Color in information display. IEEE Visualization 2006 Course Notes. http://www.stonesc.com/Vis06.

Teoh, S. T. and Ma, K.-L. (2002). *RINGS: A Technique for Visualization of Large Hierarchies. Proceedings of Graph Drawing (GD'02)*.

The Linguist List (2007). The linguist list. http://www.linguistlist.org/ (accessed 19 September 2009).

van Zulijen, M. (2005). synttree. http://www.ling.uni-postsdam.de/~rvogel/xyling/.

Vogel, R. (2004). xyLing - LaTeX macros for linguistics graphics. http://www.ling.uni-potsdam.de/rvogel/xyling/.

Walker, J. Q. (1990). A node positioning algorithm for general trees. *Software – Practice and Experience*, **20**(7): 685–705.

Webelhuth, G. (ed.) (1995). *Government and Binding Theory and the Minimalist Program*. Cambridge, MA: Blackwell Publishing.