The MorphMoulder User's Manual

MoMo v. 2.0

Frank Richter
Seminar für Sprachwissenschaft
Abteilung Computerlinguistik
University of Tübingen
Germany

Ekaterina Ovchinnikova
St. Petersburg State University
Russia

October 25, 2005

Contents

Chapter 1

Introduction

The *Morph Moulder*, or MoMo, is a tool for exploring the relationship between objects, signatures and descriptions in a typed feature logic. More specifically, MoMo is an (almost complete) implementation of the mathematical foundations of the Head-Driven Phrase Structure Grammar of Carl Pollard and Ivan Sag [Pollard and Sag, 1994] based on Relational Speciate Re-entrant Language (RSRL, [Richter et al., 1999, Richter, 2004b]), which is a comprehensive formalization of the formal language and model theory of HPSG. At the same time, MoMo is closely related to the TRALE system, an implementation platform for HPSG grammars. The signatures and descriptions of MoMo are a syntactic extension of the signatures and descriptions of TRALE. MoMo can thus also be used as an introduction to grammar writing in TRALE. It provides a direct link between theoretical HPSG grammars, their model theoretic interpretation, and their implementation in computational systems.

MoMo was first and foremost designed as a tool for teaching the feature logic formalism of HPSG by projecting its highly abstract mathematical concepts on a graphic level where they can be grasped much more intuitively by students and working practitioners in the field of HPSG alike. With MoMo you can study the mathematics of HPSG hands on in a virtual world of objects that are easy to manipulate, without first having to wade through a considerable amount of Greek symbols and disturbing squiggles in abstract mathematical definitions. Once interaction with MoMo has led to a firm understanding, on an intuitive level, of what HPSG grammars are and how they are interpreted, it is much easier to grasp the mathematical definitions of the underlying concepts, because by this point it is only necessary to understand the already familiar concepts in terms of mathematics. The compatibility of MoMo with the TRALE system highlights the relationship between theoretical grammars and their approximations in computationally efficient environments.

Acknowledgments

MoMo was the result of the project *Grammatikformalismen und Parsing (Grammar Formalisms and Parsing)*[1] of the MiLCA consortium (*Medienintensive Lehrmodule in der Computerlinguistik-Ausbildung*), which took place from 2001 through 2003. Within this project, MoMo was created as a software tool to support an HPSG-oriented eLearning course on grammar formalisms and parsing within the MiLCA consortium. Due to its origin, MoMo is closely linked to the textbook *A Web-based Course in Grammar Formalism and Parsing*, which

[1]http://milca.sfs.uni-tuebingen.de/A4/HomePage/top.html

is available on-line at `http://milca.sfs.uni-tuebingen.de/A4/Course/PDF/gramandpars.pdf`. This origin is also visible in that MoMo provides internal links to web resources with exercises and illustrating examples for this textbook. An earlier version of MoMo from the MiLCA period was described in [Richter et al., 2002] and presented to the audience of the *Formal Grammar* conference 2002 in Trento, Italy.

We would like to thank all of the people who were involved in creating and teaching the course on grammar formalisms and parsing within MiLCA. Without their support and suggestions, it would have never been possible to develop MoMo to this extent. Without the numerous students who used and criticized it, MoMo would never have become as reliable, stable and user-friendly as it is now. From among all of the people who contributed to this project we would like to single out a few who had particularly good ideas for the development of MoMo. Many of the initial ideas for a software tool visualizing the logical foundations of HPSG originated in several brainstorming sessions with Gerald Penn in the fall of 2001. Beata Trawiński made many design decisions and provided extensive productive criticism and user support during the first 18 months of the development of MoMo. She was also involved in the initial stages of this user manual. Ashley Brown and Levente Barczy wrote MoMo's note pad during their summer internship in Tübingen in 2002. Manfred Sailer's interest in, and use of, the file import function, which converts TRALE parsing output into models in MoMo, motivated us to develop this feature.

Chapter 2

Running MoMo

MoMo requires Java 1.3.1 (or higher) on Solaris 5.7, Solaris 5.8 or standard installations of Linux. We have tested it on Redhat, Suse and Debian distributions. It can also be adapted to Windows by setting the necessary `.java.policy` rights to grant Java read and write permissions in your directories. Windows is not directly supported by the installation script shipped with MoMo, but we will give a few hints as to how to go about installing MoMo under Windows in Section 2.2.

2.1 Installing MoMo under Linux and Solaris

Unpacking the tarball `momo.tar` (`tar -xvf momo.tar`) creates a directory `MoMo/` which contains the complete MoMo software package. It includes an installation script which is executed by entering `./MoMoInstaller` in the `MoMo/` directory. The installation script opens a few dialog boxes in which you can choose from among a number of installation options. The first screen provides information about the permissions that Java requires in order to run MoMo; then you may choose between running MoMo locally (i.e. offline installation) or remotely from a web page (i.e. on-line installation). The offline installation of MoMo results in a faster version but means that MoMo is installed on your local computer just like any other standard software program. The on-line version of MoMo allows you to run MoMo as an applet from an on-line installation of MoMo on the Internet. By default the on-line version is linked to a copy of MoMo at the Seminar für Sprachwissenschaft in Tübingen. The on-line option has the advantage that, as long as the on-line release is used, you do not have to update MoMo at any time, since you will automatically be using the latest version available at the given URL. The offline version, however, is considerably faster, which might be a worthwhile consideration if you are planning to use features of MoMo that have a considerable computational cost. If you wish, you can install MoMo on-line and offline simultaneously. Which MoMo you are using at any given time then depends on where you open it—on your local computer or off the Internet.

In the next step, MoMo needs information about the location of the MoMo code for the chosen type of installation. For local installation, the installation script suggests the directory in which MoMo is currently residing. Pressing the "Finish" button completes the installation.

After the installation is completed, MoMo can be opened by typing `.momo` in the `MoMo/` directory. To be able to use the command `momo` for starting MoMo from anywhere in your directory system, your system must be told where MoMo is located. Make sure that the `PATH` variable contains a path to a directory which provides a link to the file `momo` in the `MoMo/` directory. For example, you might want to use a toplevel directory `bin/`, in which a link called `momo` points to the file `momo` in the `MoMo/` directory.

Here is a short description of how this can be achieved: Go to your home directory and type `mkdir bin` to create a `bin/` directory.[1] Enter that directory (`cd ~/bin`) and type `ln -s /HOME/YOURLOGIN/MoMo/momo momo`, where the string `/HOME/YOURLOGIN` needs to be replaced by the path to your home directory on your system. If you do not know that path, you can find out by typing `pwd` in your home directory. For example, if the system answered with `/home/turing`, the complete command would be `ln -s /home/turing/MoMo/momo momo`.

In the very last step, we need to tell the system about the existence of your new `bin/` directory. How to do this depends on the shell that you are using. If you are using the tc-shell, there is a file called `.tcshrc` in your home directory. Edit that file and add `~/bin` to the settings of the path variable. The settings might then look similar to

```
set path = ( ~ ~/bin $lpath /usr/local /usr/bin.)
```

Usually there are more (and certainly different) paths set than in this brief example. After saving the `.tcshrc` file, any new shell will know about your `bin/` directory, and MoMo can be started from any directory in your account. If you are using bash shells, the general idea is the same, but the file that you need to edit has a different name, and the syntax is slightly different. Here, you need to edit the file called `.bashrc` in your home directory, and the `PATH` variable needs to include `~/bin`. Here is an example:

```
export PATH="$PATH:/usr/local/jdk1.3.1/bin:~/bin"
```

If you still do not know how to set the `PATH` variable and make the `momo` command known to your system, please consult the manual for your operating system or your system administrator. If your system does not know about the location of the MoMo program, MoMo can only be started by going into the `MoMo/` directory and opening MoMo from there (by typing `./momo` or `./momo &`).

2.2 Installing MoMo under Windows

The following is intended to help you if you want to run MoMo under Windows.[2] Since the installation script does not support Windows, the installation must be done manually.

The MoMo tarball can be unpacked with WinZip.[3] Unpacking this creates a directory `MoMo/`, containing files for installing MoMo on Solaris and Linux, and a second tarball, `MoMo.tar.gz`. You may delete the installation files for Linux, i.e. all files except the second tarball. Unpack the tarball to obtain another MoMo directory, called `MOMO/`. A good location for this directory may be `D:\MoMo`. To be able to start MoMo locally, you will need to add a file named `.java.policy` to an appropriate directory of Windows. For Windows 9x/Me, this will typically be `C:\WINDOWS`, for Windows 2000/XP it will probably be `C:\WINNT`.[4]

We need to declare in the `.java.policy` file where the source code of MoMo lives in your account. For this purpose, you need to edit the first line of the file (fully shown below),

[1]Unless this already exists, of course, in which case you will probably not have to inform your shell about the existence of this directory either, because it might already be in use.

[2]We are grateful to Holger Wunsch for sharing his insight into a Windows installation of MoMo with us, and to Armin Buch for additional suggestions.

[3]http://www.winzip.com

[4]For more information on Windows/XP, see the comments toward the end of this section.

which starts with `grant codeBase`. If you have put MoMo in the directory `D:\MoMo`, your `.java.policy` file needs to contain at least the following statements:

```
grant codeBase "file:/D:MOMO/-" {
        java.security.AllPermission;
};
```

If you would like to work with MoMo off of a web server, you need to grant the remote MoMo the same rights as a local copy. Thus, you need to set the same permissions as you would for a local copy of MoMo in the `.java.policy` file by entering the web address from which you want to start MoMo. Suppose that in addition to using a local copy of MoMo, you would like to work with a copy of MoMo that is located at the URL
`http://milca.sfs.uni-tuebingen.de/A4/Course/Momo/Online-Edition/MOMO/`
Your `.java.policy` now needs to contain a second set of permissions that is identical to the first one, except that the first line is

```
grant codeBase "http://milca.sfs.uni-tuebingen.de/A4/Course/Momo/Online-Edition/MOMO/-" {
```

In sum, for each location, be it on your local account or on the web, from which you would like to open a copy of MoMo, you must state the location and the rights of that particular use of MoMo in your `.java.policy` file. Simply copy the example settings given in the distributed file as often as needed and fill in the appropriate locations.[5]

Once `.java.policy` is installed, you can go to the directory of MoMo in your DOS box and open MoMo with `appletviewer applet.html`. You may also open MoMo with the command `java momo file_url`. Here, you have the option of providing a file name as an argument, which causes MoMo to open that file immediately. Alternatively, you may rename the file `momo`, calling it `momo.bat`. Then you may open MoMo by clicking on `momo.bat`.

Windows/XP has trouble with filenames that start with a dot. To solve this problem, you can either copy the `.java.policy` file from the `MOMO/` directory and edit it as explained above, or you may prefer to use Java's `policytool.exe`: Open it and click ok when you get a complaint saying that you do not yet have a policy file. Create a new entry and enter the exact path to your MoMo directory in the first field. Make sure to begin the entry with `file:/`... and to use / instead of \. For example, `file:/C:/Programme/MoMo` is an admissible format for an entry.

Finally, add the appropriate permissions. The easiest solution is to choose "All permissions" in the first pop-up field. Leave the other fields empty. Click ok, and file`->`save these in a file called `.java.policy` in your personal directory, e.g. `C:\Dokumente und Einstellungen\Name\` or `..\All Users`.

[5]A complete documentation of `.java.policy` is available on Sun's webpages at http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html.

Chapter 3

Working with MoMo – Essentials

MoMo has two main windows. The first window, which we will call the *note pad*, appears when MoMo is started. From this window, a second window, which we will refer to as the *graph window*, can be opened as one of the possible functions of MoMo. Logically, the functions of the note pad have to do with the description language of HPSG. Functions that have to do with *models* or, more generally speaking, *interpretations* of the logical expressions created in the note pad window are displayed in the graph window. In additional windows, which we will call *interaction windows*, the system provides feedback to the user. MoMo may communicate information about the state of the program, about the result of some action performed by the user, or display error messages.

In Section 3.1 we explain the most essential functions of the note pad window, and Section 3.2 is devoted to the graph window. A discussion of more advanced functions is deferred to Chapters 4 and 5.

3.1 The MoMo Note Pad

From a formal point of view, MoMo has three logical components: *Signatures*, *descriptions*, and *interpretations*, visualized as graphs. In the note pad, each of these three components is assigned a separate part of the window:

The *signature* editor is found on the left hand side of the note pad. Well-formed descriptions as well as interpretations are defined relative to signatures. They are dependent upon signatures, and without a signature descriptions cannot be evaluated relative to interpretations. Even more crucially, if a signature is changed, interpretations that may have been created for the old signature cannot automatically be carried over to a new signature. This is because a signature determines the space of abstract entities that may exist and the logical language may talk about, and a change in the signature may render entities impossible that could very well have existed under the previous signature. To prevent the appearance of impossible objects in interpretations, signatures can no longer be changed once interpretations have been created for them.[1]

The *descriptions area* (an editor for descriptions) is found in the upper right of the note pad. The *interpretations area* (an area where you can choose from a set of interpretations) of the note pad is on the right in the lower half of the note pad.

[1]Experienced users may override this safety measure. See the description of advanced functions in Section 4.6.

The relative sizes of the three areas in the note pad can be adjusted as necessary. The left part of the window always contains the signature, but the middle bar to the right of the signature can be moved left and right. On the right side of the note pad, the descriptions are always above the interpretations area, but the bar that separates the two can be moved up and down to adjust their sizes as needed.

We will now describe the functions of the three areas of the note pad in turn.

3.1.1 Signatures

At the very top of the signature area there is a white line in which you assign a name to your signature (together with its set of descriptions and its set of interpretations). It is obligatory to enter something in this space.

The signatures of MoMo are RSRL signatures [Richter, 2004b, p. 156]. Signatures are written according to the notational conventions for signatures of TRALE,[2] with an additional convention added for declaring relation symbols and the arity of the corresponding relations. Figure 3.1 shows an example.

Following TRALE conventions, the signature must begin with the declaration `type_hierarchy` in the first line, and end with a period in the last line. The set of sorts is declared using an indentation structure, where indentation expresses the subsort relationship. A sort may be followed by a sequence of attributes that are appropriate to it. Each attribute is in turn followed by a colon and the sort that is appropriate for the attribute at the sort in the hierarchy stated at the beginning of the line. Attributes and attribute values are inherited by subsorts.

After the last sort declaration (in our example in Figure 3.1 this is the line with the sort symbol `brown`), there is a line with the keyword `relations`, followed by the declaration of relation symbols. These are followed by a slash and the arity of each relation. In our example `to-the-right` is a three place relation (x is to the right of y on list z), and `member` is a two place relation (entity x is an element of set y). Signatures may lack relation declarations and may thus end with a period in the line immediately below the last sort declaration.

You may add comments to your signatures by putting the symbol `%` anywhere in your signature file. The rest of the line will be interpreted as a comment and thus be ignored by MoMo.

There are two modes of the signature editor. With a white background, the signature is editable and can be modified. As long as the signature editor is in editing mode, interpretations cannot be created, since the space of possible entities in interpretations has not yet been determined. With a red background, the signature is not editable, and you can create interpretations. Descriptions can be created in both modes of the signature window. Their well-formedness is checked relative to the current signature in the signature editor. If you already have interpretations, you can no longer change the signature without losing them (but see the remarks in Section 4.6, page 31). Make sure you saved your file prior to editing a signature in case you do not want to lose existing interpretations.

[2]Except that (1), sort hierarchies in MoMo are not required to have a unique top element. In this more liberal usage MoMo follows RSRL, whereas TRALE follows actual linguistic practice within the formal framework of RSRL. And (2), MoMo does not impose a *feature introduction condition* on the appropriateness of attributes to sorts like TRALE, which requires the existence of a unique greatest lower bound in the sort lattice for the introduction of each attribute: For each attribute in TRALE, there must be a unique sort in the sort hierarchy such that the attribute is appropriate to that sort and each of its subsorts.

```
type_hierarchy    % a signature for the description of (lists of) animals
bot
  list                                    % sorts for lists
     nelist head:animal tail:list
     elist
  animal legs:number color:color          % the animals
     bird legs:two
       parrot
       woodpecker
       canary
     pet legs:four
       cat
       dog
  number                                  % fixed numbers for legs
     one
     two
     three
     four
  color                                   % a few possible colors
     green
     red
     yellow
     brown
relations                                 % relationships on lists
totheright/3
member/2
.
```

Figure 3.1: A signature with relations

Switching between the two modes can be achieved with the check box *Signature is editable* directly above the buttons over the editor of the signature, or by pressing the button *Edit Signature* while in non-editing mode. If you try to switch to editing mode when there are already interpretations of the signature, making the signature editable will delete them.

The editor windows of MoMo have three simple editing functions, cut, copy and paste, which work in the familiar fashion. They are available from the note pad menu *Text-Edit*, but they can also be invoked by their key shortcuts, `<control>-u` (cut), `<control>-c` (copy), and `<control>-p` (paste).

Pressing the *Check Syntax* button initiates a syntax check of the signature. Results of the syntax check will be communicated through an interaction window. If the signature is syntactically ill-formed, MoMo provides a detailed error report. *Print Signature* starts a dialog box that asks for printer information and gives you a choice between sending the file to a printer or printing it to a postscript file. *Open Signature* starts a dialog box in which you can specify an arbitrary file that will then be opened in the signature editor. The name of that file will be assigned to the *Title* line at the top of the signature area and may be changed manually. The *Open Signature* function allows you to create signatures with your editor of choice and then import your externally created signature into MoMo. Especially if you are planning to work with a multitude of similar signatures, this can be a very useful function.

MoMo signatures have additional specialized features having to do with the possibility of importing parse output from TRALE. These are described in Section 4.6.

3.1.2 Descriptions

In the *description area*, arbitrarily many *cards* can be created. Each card may contain descriptions, in which well-formedness is determined relative to the signature in the signature area. MoMo's descriptions are TRALE descriptions augmented by all necessary syntactic constructs to give the language almost all the expressive means of RSRL.[3] For a specification of the syntax of well-formed expressions and their semantics, see Chapter 7. Each description in the description editor must end with a period. Just like the editor window for signatures, the editor for description cards also has three editing functions. They are available from the note pad menu *Text-Edit* or by their key shortcuts, `<control>-u` (cut), `<control>-c` (copy), and `<control>-p` (paste). You may insert comments anywhere on a description card by inserting the symbol `%`. Everything following that symbol in a given line will be taken as a comment and will be ignored by the logical functions of MoMo.

New opens a new description card. Each card is given a name. In a stack of cards, you activate each card by clicking on its name. The inactive cards are shown in a darker color. All logical functions involving descriptions in the graph window are performed relative to the active description card. A description card may contain any number of descriptions. Each description ends with a period.

Delete deletes the active card. With *Rename* you can give existing cards new names. *Check Syntax* performs a syntax check on the description(s) in the active card. MoMo will provide information on the syntactic well-formedness of the descriptions in the active card through an interaction window. Pressing *Print* opens the dialog box for printer information. The active card may be sent to a printer or printed to a postscript file.

With *Open* any file in your file system may be opened as a description card. Initially each card will be assigned the name of the file that you open. The name of the card may then be changed using the *Rename* button.

You can move a card on the stack to the left by clicking on it with the right mouse button. This function allows you to reshuffle the stack of description cards into any order you like.

3.1.3 Interpretations

Interpretations consist of graphs, which are created, modified, and displayed in a separate *graph window*. The *interpretations area* of the note pad keeps track of existing (sets of) graphs for the given signature. All logical functions (such as *satisfaction checking* relative to active descriptions, or *model checking* relative to active descriptions) are found in the graph window. The graph window and all logical functions connected to it are described in Section 3.2. In the present section, we will only describe the functions of the interpretations area of the note pad.

With *New* you open a new set of graphs. At most one graph window can be open at any given time. After having selected *New*, you are asked to name the new graph(s). For each collection of graphs, there will be a radio button in the interpretation section of the note pad.

[3]The only exception are *chains*, which are not included in MoMo.

The radio button of the open collection of graphs is marked by a black dot, all others are represented as empty circles. Each radio button is followed by the name that you assigned to the corresponding collections of graphs. When you click on a radio button with the left mouse button, it becomes active and the interpretation window shows the set of graphs assigned to it.

With *Duplicate* you can duplicate an existing collection of graphs and assign the duplicate a new name. The *Duplicate* function duplicates the collection of graphs designated by the active radio button. You may modify or work with the duplicate while leaving the original intact.

Delete deletes the collection of graphs corresponding to the active radio button. *Rename* allows you to assign a new name to an existing collection of graphs, i.e., you can change the names behind the radio buttons. *Print* starts the printing dialog, where you can select a printer to print the active collection of graphs. You can also print your graphs to a postscript file.

You may reshuffle the order of your radio buttons by clicking on the active radio button with the right mouse button. This active radio button will then switch position with the radio button above it.

3.1.4 The Note Pad Menu Bar

The menu bar of the note pad window comprises the functions of its three areas described in the previous (sub-) sections. In addition, it provides a number of additional functions for each area, and for the entire MoMo program. Here we describe the most important additional functions, sorted by menu items.

File The *File* menu item subsumes the standard functions for exiting the program and for opening and saving files. MoMo's files are encoded in a particular file format and receive the suffix `.mmp`.

The menu item *Get Web Resource* contains a selection of links to web resources with MoMo files. In the basic distribution of MoMo, *Get Web Resource* provides links to examples and exercises from the textbook *A Web-based Course on Grammar Formalisms and Parsing.*[4] The links are named after the sections of the textbook in which the examples and exercises are found. By editing the file `webresourceindex.xml` in the MoMo sources, you may easily adapt the selection of down-loadable files to your needs.

Text-Edit subsumes three standard text editing functions (cut, copy, paste) for the editors of the *signature* area and the *descriptions* area of the note pad.

Signature contains all functions of the *signature* area, plus two new functions:

Save Signature as is the counterpart of *Open Signature.* It saves signatures in ASCII format. These can then be opened in other editors or be used in TRALE.

Sort Signature Alphabetically can be useful for large, unfamiliar signatures. All sort names in the sort hierarchy are sorted alphabetically in top down order, preserving the indentation structure; attribute declarations are likewise sorted alphabetically from left to right. This

[4] http://milca.sfs.uni-tuebingen.de/A4/Course/PDF/gramandpars.pdf

function can significantly help in locating relevant declarations in an unfamiliar signature. An example for the effect of the sorting function is shown in Figures 3.2 and 3.3.

```
type_hierarchy
top
        c_sort  b_feature:top  a_feature:top
                d_sort
                a_sort
        b_sort
.
```

Figure 3.2: A signature before sorting

```
type_hierarchy
top
        b_sort
        c_sort  a_feature:top  b_feature:top
                a_sort
                d_sort
.
```

Figure 3.3: The signature of Figure 3.2 after alphabetical sorting

Description contains all functions of the *description* area; and the function *Save Description as* for saving description cards in ASCII format.

Interpretation contains all functions of the *interpretations* area; plus the option of saving the active interpretation in the graph window as a `jpeg` or `gif` file (*Save as Picture*). The pictures may then be imported to other documents. Note that screen shots, if they are created with appropriate tools, may lead to better quality than jpegs and gifs.

Options provides the check box *Advanced Functions*, the significance of which will be explained in Chapter 4, as well as three useful functions:

Font Size adjusts the size of the characters in the *signature* and *description areas*. This is very useful for slide presentations with MoMo.

Top-sort for Lists declares the top sort of the part of the signature for which you want to use the abbreviatory bracketing notation for lists. See Chapter 4 for a more detailed explanation.

Preferences: for general personal settings that are preserved after closing MoMo. You can define default settings for opening particular files, interpretations and description cards after startup, a permanent setting for the top sort symbol of lists for using the list notation in descriptions, and the web browser which MoMo will open upon calling the help function. Your preferences are saved to the file `.MoMoPreferences` in your home directory, i.e., to ∼/`.MoMoPreferences`.

Settings in *Preferences* indicate whether the interpretation window displays labels for the nodes and the arrows on the canvas and whether it displays the numbers that MoMo automatically assigns to the nodes of graphs. The default semantics used by MoMo can be set to either the feature structure semantics or to King-style interpretations. You can also define their preferred font size on the note pad and in interpretations. All of these settings are described in much more detail in Chapter 4, Section 4.3.

Info provides release information about MoMo.

3.2 The MoMo Graph Window

The user interface of the graph window MoMo is divided into three areas, which we will refer to as the *columns* of the graph window. Each column is characterized by a different function.

The most important column is the white drawing board—or canvas—in the center. This is the space where graphs can be created. The left column provides the building materials for the graphs: Here you select from among the balls representing entities of all the maximally specific sorts declared in the signature (of the note pad window) and from among the arrows (for the attribute functions) that connect the entities on the canvas. The possible arrows are represented as colored slanted lines with their attribute names next to them. The right column contains the logical functions and some simple editing functions of the interpretation window. At the top, a green and a red button signal the results of logical operations, success or failure. At the bottom of the right column, there are three buttons that control certain drawing functions on the canvas.

The menu bar at the top of the window provides various functions, which will be explained in Section 3.2.3.

In the following sections, we eill focus on the two tasks of the graph window, creating and manipulating feature graphs, and performing logical operations on feature graphs. Section 3.2.2 focuses on basic logical functions. Chapter 4 is reserved for the description of some more advanced features of MoMo. Throughout the present chapter, which focuses on the essential logical functions of MoMo, we presuppose a semantics of feature structures for our descriptions. Chapter 4, Section 4.5, provides information on how MoMo can be used with an alternative semantics for the same descriptions. This alternative semantics is based on a proposal first made by Paul King in connection with HPSG.

3.2.1 Drawing Graphs (Feature Structures)

In the middle column of the graph window, there is a large white drawing board, which we will call the *canvas*. The canvas is for drawing configurations of objects, that is, colored balls that are connected by arcs. The configurations of objects on the canvas can be thought of as the totally well-typed, sort resolved feature structures of HPSG grammars.[5]

[5]More precisely, they are totally well-typed, sort resolved *concrete* feature structures. In Section 3.2.3 (under *Additional Functions*) and in Chapter 5 we will introduce a function of MoMo which displays the *abstract* feature structures that correspond to the concrete feature structures on the canvas. It is abstract feature structures which are conventionally used in the feature structure semantics of HPSG, but these abstract feature structures are very inconvenient for graphical representations. A second kind of denotation for HPSG descriptions—following a proposal by Paul King and also available in MoMo—does not employ feature structures at all and thus does not involve concrete or abstract feature structures. See Section 4.5 for explanations. All of these

The resources for drawing feature structures are the balls and arrows available in the left column of the graph window. In the upper half of the left column, there is a collection of nodes, labeled by the maximally specific sorts or species of an underlying signature. In the lower half of this column, there is a collection of arcs (slanted lines), labeled by the attributes of the underlying signature. If there are more nodes and arcs than fit in half of the column, the scrollbar for the respective part of the column may be used for scrolling up and down to search for the attributes and sorts that you may want to use.

Creating nodes and arcs In both halves of the left column, the first for nodes and the second for attributes, one item can be activated by clicking on it with the left mouse button. Active elements are outlined in white and can be copied to the canvas.

A node is copied to the canvas by, first, activating it, and, second, clicking somewhere on the empty space on the canvas with the left mouse button. A copy of the active node will appear there.

Similarly, in order to create an arc on the canvas, we need to activate it first in the panel in the left column. You can connect nodes by first clicking on the desired arc in the attribute panel in order to activate it. After this, click on the node where the arc is to originate, then click on the node that the arc should point to (both clicks with the left mouse button). As soon as you click on the second—or target—node, the arc appears on the canvas. MoMo automatically gives it a shape so that it overlaps with other arcs on the canvas as little as possible.

When you create an arc and click on the originating node of the arc, a red square appears around the originating node in order to mark it as such. If you notice at that point that you have made a mistake, and you do not want to create an arc that originates from the node with the red square around it, you can simply deactivate that node as the originating node by pressing the `<backspace>` button on the keyboard. The red square around the node disappears and you may select another node as the originating node of a new arc.

Deleting nodes and arcs Nodes are deleted by clicking on them with the right mouse button. All arcs connected with them will automatically be deleted along with them. Arcs are deleted by clicking on their arrow head with the right mouse button.

Creating—and deleting—root nodes By definition each feature structure has a unique root node. Each node in a particular feature structure can be reached by following a sequence of arcs originating at the designated root node of the feature structure. Any node on the canvas may be assigned root node status by clicking on it with the left mouse button while pressing the `<control>` key. A red circle appears around it, marking it as a root node. Root node status may be removed from a node by clicking on a root node with the right mouse button while pressing the `<control>` key.

Dragging objects over the canvas The shape of feature structures on the canvas can easily be changed by dragging nodes to new places. Click on a node with the left mouse button, and keep it pressed while dragging the node across the canvas. Incoming and outgoing arcs will automatically be adjusted to the new position of a node.

technical ramifications can safely be ignored in the understanding of the essentials of MoMo.

Editing relations Relations in HPSG are relations between nodes in feature structures.[6] Thus, relations are tuples of nodes. The number of elements in each tuple depends on the arity of the relation. For example, the usual `append` relation is a set of triples, whereas the `member` relation is a set of pairs of nodes.

Relations are displayed at the bottom of the canvas. If the underlying signature does not contain any relations, the part of the canvas where they would be represented is missing, while the part of the canvas for the graphs is larger. The relation segment of the canvas contains each relation symbol of the signature, followed by an equation symbol and the set of nodes that are in each relation. Before any changes have been made, the set tuples of nodes that are in each relation is empty.

Nodes in relations are represented by the names that are assigned to them. MoMo automatically assigns alphanumeric names to nodes when they are first created on the canvas.[7] The names of the nodes consist of two components, a letter and an integer. All nodes within a single feature structure receive the same letter.[8] For example, if a node belongs to both the feature structure A and the feature structure B, and the numerical identifier of the node is 3, then its complete name is A/B3. To see the node names which MoMo assigns to every node on the canvas, the menu option *Show Node Numbers* under the menu item *Option* of the graph window must be activated. The display of the letters in the node names can be switched on and off by the settings of *Option → Show Node Letters*. For the computational consequences of switching on the node letters in node names, see the remarks in the description of the *Show Node Letters* function on page 20 (Section 3.2.3).

To add tuples of objects to a relation, the corresponding relation symbol is first activated by clicking on it with the left mouse button. Active relations are red. A node is then added to a tuple in the active relation by clicking on it with the left mouse button while pressing the `<shift>` key. If there is no incomplete tuple in the active relation, adding another node to the relation will start a new tuple. The number of elements in the tuples is inferred by MoMo from the arity of the relation, and missing elements in a tuple are indicated by an underscore in each argument position which still needs to be filled by a node. Each click of the left mouse button on a node (while pressing `<shift>`) fills another argument slot with a node. As before, when all slots of a tuple are filled, adding another node to a relation will restart a new tuple. This process can be repeated as many times as desired.

Tuples—no matter whether or not they are complete—are removed from a relation by clicking on them with the right mouse button. Alternatively, simultaneously pressing `<shift>` and `<backspace>` deletes the last tuple of the active relation in the graph window.

A number of additional functions for editing graphs are described in Chapter 4, Sections 4.4 and 4.6.

[6]More accurately, in those formalizations of HPSG which rely on a feature structure semantics, only relations that hold between nodes within a particular feature structure are used. In interpretations without feature structures this is, of course, not possible and relational expressions are interpreted differently.

[7]MoMo changes node names on the fly depending on the current configurations on the canvas. MoMo does this consistently for all occurrences of the node in an interpretation, and the particular names of nodes can be safely ignored.

[8]Should a node belong to two or more feature structures, its name contains the letter that corresponds to each of the feature structures. For MoMo, a node belongs to more than one feature structure if it is in the substructure of more than one root node.

3.2.2 Logical Operations on Feature Structures

In the column to the right of the canvas the panel for the logical functions of MoMo is located. This is where the logical operations of MoMo can be invoked and where MoMo signals the success or failure of a logical operation with a red and a green light. After performing an operation, one of these lights will light up. An interaction window provides further information on the result of the operation.

Four operations can be performed on entire feature structures, which will now be described in turn: (1) well-formedness checking relative to the signature, (2) feature structure checking, (3) satisfaction checking, and (4), model checking.

Well-formedness relative to the signature Clicking on the button *Obeys Signature* will perform a check on the configuration of nodes and arcs on the canvas to see whether it is well-formed with respect to the signature. If it is well-formed, the green *Success* light will light up and the configuration on the canvas will be outlined in gray. If it is not well-formed, the red *Failure* light will be on, and only that part of the feature structure that proved correct will be outlined in gray. Starting from each root node in each configuration of nodes that is connected by arcs, the gray outline will stop at the first node of each branch of the graph at which the appropriateness conditions of the signature are violated. If no root node is provided by the user, MoMo infers a potential root node for each configuration of nodes connected by arcs on the canvas. If a root node is provided by the user in a connected configuration of entities, signature checking will start at the root node and will ignore those parts of the configuration that cannot be reached by any sequence of arcs from the given root node.

If the configuration does not obey the signature, an interaction window gives a comprehensive analysis of where the problem lies.[9] In that analysis, it refers to the nodes by the alphanumeric names that MoMo has assigned to them. The node names can be displayed next to each node on the canvas by activating *Show Node Numbers* in the menu *Options* of the graph window.

After a signature check, the canvas is framed by a red line. The red outline can be removed by pressing the *Proceed* button or by performing any other operation.

Feature structure checking Pressing *Check Feature Structure* prompts MoMo to check whether the configuration of objects on the canvas obeys the algebraic definition of concrete feature structures, disregarding appropriateness conditions (which are independently checked by the previously described function). Simplifying a bit, clicking on the button *Check Feature Structure* tests whether there is at least one designated root node for each configuration of nodes on the canvas which is connected by arcs such that each other node in the connected configuration can be reached from the root node by following a sequence of arcs.[10]

[9]If the feature structure semantics is activated, MoMo enumerates only the problems at the first node at which it finds mistakes. In the King-style semantics, MoMo reports all problems anywhere in the configurations on the canvas.

[10]MoMo allows cases in which a node in a concrete feature structure belongs to several feature structures, i.e., these feature structures overlap in parts of their substructures, but none is properly embedded in the other. However, each of these overlapping concrete feature structures must be well-formed in the sense that must have exactly one root node from which all of its nodes can be reached, and there are no orphan nodes without a root node in our universe. To verify these conditions, MoMo checks whether each node belongs to a substructure of at least one root node, and that no root node belongs to the substructure of any other root node.

Feature structure checking also fails if a relation contains tuples with nodes from different feature structures on the canvas. According to the definition of relations in feature structure based HPSG, a tuple in a relation has to be a tuple of nodes from one and the same feature structure. Relations between nodes of distinct feature structures are not permitted.

If a well-formedness condition on feature structures is not satisfied, an interaction window provides further information.

Satisfaction checking Clicking on the *Check Satisfaction* button will invoke a satisfaction check of the feature structure on the canvas relative to the set of descriptions on the active description card on the note pad. If the active description card is empty or contains only an empty description (that is, it contains only a period), a satisfaction check cannot be performed.

If there is more than one feature structure on the canvas, each of these is checked to see if it satisfies the set of descriptions. If each feature structure satisfies the set of descriptions, all of them will be outlined in green, and the *Success* light goes on.

If the red light signals *Failure*, there are three possible reasons:

First, if the canvas is empty, no feature structure can be checked for satisfaction and the operation fails.

Second, a real satisfaction check may only be performed if the configuration of entities on the canvas obeys the signature and consists only of well-formed feature structures. If one of these preconditions fails, the test will always fail, and an interaction window will explain the reason. Mistakes in appropriateness and well-formedness of feature structures are indicated the same way as they were in the operations *Obeys Signature* and *Check Feature Structure.*

Third, if the feature structures on the canvas are well-formed, *Failure* indicates a failure of the feature structures to satisfy the set of descriptions on the active description card of the note pad, and the red light goes on for satisfaction failure. If there are several feature structures on the canvas, satisfaction will fail if one of them fails to satisfy the descriptions. However, MoMo tells you which feature structures satisfy the descriptions and which ones do not by outlining those feature structures that do in green.

Just like after any logical operation, the canvas is framed by a red line after a satisfaction check. The red outline can be removed by pressing the *Proceed* button or by performing any other operation.

A satisfaction check fails if it is performed with an empty description card or a description card containing an empty description. An empty description is a description which consists only of a period (since a period marks the end of descriptions).

Model checking Clicking the *Check Modeling* button starts a check to see whether the feature structure on the canvas models the set of descriptions on the active description card of the note pad.

Just as with feature structure satisfaction, model checking presupposes that the feature structures on the canvas are well-formed feature structures that obey the appropriateness conditions of the signature. If they do not, model checking fails prematurely and the information window provides an analysis of the failure.

If the preconditions are met and the feature structures model the description(s), the feature structures are outlined in red and the *Success* light goes on. Moreover, there is the usual red frame around the canvas.

If the feature structures do not model the descriptions, the model check has failed. MoMo provides a precise analysis of which parts of the feature structures led to a failure in the model check. All parts of feature structures that are possible models of the descriptions are outlined in red. All nodes on the canvas that do not satisfy at least one of the descriptions on the active description card receive a black circle around them.

In contrast to the satisfaction check, a model check succeeds with an empty canvas, since the empty set of structures is the trivial model of all theories. Similarly, the model check of every well-formed feature structure succeeds relative to the empty set of descriptions, indicated by an empty description card. Every feature structure models the empty theory. Analogous to the case of a satisfaction check, empty descriptions are not allowed, i.e., descriptions which consist of nothing apart from a period are syntactically ill-formed.

Two operations can be performed relative to activated nodes on the canvas. Nodes are activated by clicking on them with the left mouse button. A red square appears around them to signal that they are active:

Check Well-typedness Performing this function shows whether the substructure generated by the activated node is well-typed. It succeeds if every node in the substructure has exactly one outgoing arc for each appropriate attribute, and if at the end of each arc there is a node of a sort that is appropriate according to the signature.

Check Satisfaction This button invokes a satisfaction check relative to the activated node. MoMo checks whether the activated node satisfies the descriptions on the active description card of the note pad. The results are indicated in the same way as for satisfaction checking of feature structures.

In the lower right corner of the graph window there are three keys that can perform important actions for the entire canvas. Pressing *Clear Graph* removes the entire contents of the canvas. *Undo Last Add/Delete* undoes the last graphical operation on the canvas. *Proceed* removes the red outline around the canvas that appears upon completion of each logical operation on its contents.

3.2.3 The Menu Bar of the Graph Window

The menu bar provides a number of additional functions for the graph window. We will briefly describe each menu item in turn, as it appears on the screen:

Interpretation The menu item *Interpretation* bundles functions that concern the overall interpretation window. You can start a new interpretation (while preserving the current one), you can clone the current interpretation to preserve it in its current state while you continue your work with its duplicate, you can delete the current interpretation, or you can rename it. To export the contents of the canvas, *Save as Picture* can be used to save the graph(s) on the canvas in a file in either jpeg or gif format. Finally, you may send the content of the canvas to a printer, or print it to a postscript file. Each of these functions is supported by dialog boxes.

Graph-Edit The item *Graph-Edit* bundles functions that manipulate the content of the canvas. First, it repeats the buttons from the lower part of the right column of the graph window, which have already been described above. Then there are five additional functions for modifying graphs on the canvas. All of these are described in Chapter 4, *Advanced Features.*

Verify The *Verify* option repeats the logical functions of MoMo that can be performed by using the buttons in the right column of the graph window.

Options contains functions for adjusting the appearance of the screen and for modifying the logical functions by switching to a different kind of semantics for the descriptions.

Font Size allows you to choose a preferred font size for the sort labels, the attribute labels and the relation symbols. This is a very useful function if MoMo is used for presentations with a projector.

Grid Size is a feature of the automatic graph optimization function of MoMo, which is one of the *Additional Functions.* The grid size chosen here is used to arrange the nodes of the optimized graph on the canvas. It is also relevant for importing TRALE output into MoMo.

Size of Relation Field allows you to adapt the size of the relation field manually. The height of the relation field visible below the canvas can be changed, and it is also possible to vary the length of lines for displaying the tuples in the relations. Depending on the setting, the entire length of the lines may not be visible under the canvas, and you may have to scroll left and right if the lines are wider than the visible area.

Show Node Labels displays the sort symbols and attribute symbols next to each node and arc on the canvas. This function has usually been switched on by the corresponding setting in the *Preferences* (under the menu item *Options* of the note pad window).

Show Node Numbers displays the alphanumeric names that MoMo automatically assigns to each node on the canvas. This is necessary for interpreting those messages in the interaction windows in which MoMo talks about specific nodes on the canvas by referring to their names. It is also necessary to switch this function on in order to see which nodes are in which relation tuples. Just as with the node labels, the setting for the node numbers is pre-selected by MoMo *Preferences* under the menu item *Options* of the note pad window. The initial letter which starts each node name can be switched on and off using the next menu item:

Show Node Letters is switched on by default. The letter in front of each node number signals the feature structure to which the given node belongs. However, computing with large structures in which node names contain the initial letter is costly. Therefore it is advisable to switch off the initial node letters when computing with large structures. This will significantly speed up computation with all relevant algorithms.

Semantics allows you to choose between a feature structure semantics, for our descriptions, and a King-style semantics. The second semantics, not described in this section, is a feature of MoMo that might be of interest to advanced students of the semantics of HPSG and is discussed Chapter 4. When MoMo is opened, it is automatically set to the feature structure semantics (unless the default setting is changed in the *Preferences*).

Scrollbars for Canvas is switched off by default. This option provides scrollbars for the graph canvas (up/down and left/right). Scrollbars for the canvas are necessary if you would like to draw or inspect a graph that is larger than the section of the canvas that can be displayed on the computer screen.

Additional Functions *Additional Functions* subsumes a convenient sorting function and three advanced features of MoMo.

Sort Attributes and Sorts Alphabetically is a function that reorders the selection of nodes and attribute arcs in the left column of the graph window. Executing this function orders them alphabetically in top down order. The color of the nodes and arcs changes, because the coloring of the balls and arcs remains constant relative to their position in the left column, but the labels assigned to them change according to the alphabetical ordering.

Show Abstract Feature Structure converts all concrete feature structures on the canvas into abstract feature structures. The results of this conversion are displayed in a separate interaction window. The menu bar of the interaction window for displaying abstract feature structures contains two menu items, *Window* and *Options*:

With *Window* → *Save* the data displayed in the window can be saved in html format. The file will receive the html file extension. *Window* → *Close* closes the window.

The second menu item, *Options*, again provides two options. When the window is first opened, the first one of these, *Show AFS*, is active, and MoMo displays the abstract feature structures that correspond to the concrete feature structures on the canvas. *Show paths* gives compressed information on the abstract feature structures that helps to better understand their representation: For each node in each concrete feature structure in the graph window, *Show paths* displays the alphanumeric name assigned to it by MoMo, its sort label and the members of the equivalence class of paths that represent the node in the abstract feature structure.

For notating potentially infinite sets (of paths) in abstract feature structures, MoMo employs a variant of a regular expression language. A specification of this language can be found in Chapter 5.

MoMo's transformation algorithm is defined for most of the interesting shapes of concrete feature structures, but it is not a total function. Certain special cases with nested cycles in concrete feature structures are not captured. The relevant cases are specified in Chapter 5.3. If you try to convert a concrete feature structure for which the transformation algorithm is not defined, a warning message appears.

Show Nodes in Relations can help you construct models of relational descriptions. Given a concrete feature structure and a description that defines the meaning of a relation symbol, it computes the set of tuples of nodes that need to be in the relation extension of the feature structure so that the feature structure is a model of the description. If no restrictions on the relation symbol are provided, the function will tell you that any tuple of nodes is permitted in the respective relation. Note that if a relation is defined based on the meaning of a second relation, the clauses for the second relation must also be stated to sensibly compute the tuples which are in models of the first relation.

Certain conventions must be obeyed for this function. The description which defines the meaning of a relation must have the following syntactic form:

`VX VY ...  VZ (name-of-the-relation(X,Y, ..., Z)` $<*>$ δ`).`

δ is an expression in which the variables `X`, ..., `Z` may occur free. The number of these variables should, of course, be identical to the arity of the relation `name-of-the-relation`, as declared in the signature. In all other respects the expression(s) may be arbitrary description(s) written in the syntax specified in Chapter 7.

Let us briefly illustrate this with an example. Suppose that, on the basis of the signature in Figure 3.1, we write the following definition of the meaning of the `member` relation on the active description card:

`VXVY(member(X,Y)` $<*>$ `(Y:head:X;^Z(Y:tail:Z,member(X,Z))).`
In addition, we provide the feature structure graph shown Figure 3.4 as input.

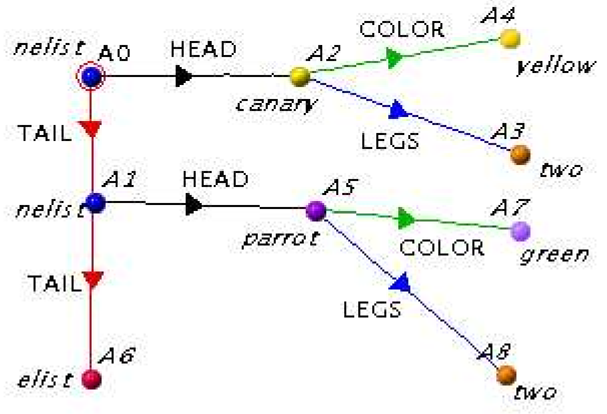

Figure 3.4: A feature structure as input for the function *Show Nodes in Relations*

Executing *Show Nodes in Relations* with the definition of the `member` relation above and the feature structure of Figure 3.4, will produce the following result in an interaction window:
`member` $= \{\langle A5, A0\rangle, \langle A5, A1\rangle, \langle A2, A0\rangle\}$
`totheright` $=$ {any possible tuples}

This means that the three pairs $\langle A5, A0\rangle, \langle A5, A1\rangle$ and $\langle A2, A0\rangle$ must be in the relation extension of the given feature structure for turning the feature structure into a model of the given relational description. Since no restrictions on the relation `totheright` are provided, any possible tuple may exist in this relation. Upon closer inspection of the relevant definition, however, you may notice that the arity of the possible tuples must correspond to the arity of the relation as specified in the signature. For our example of the `totheright` relation, this means that only triples of nodes may exist in the relation.

In the interaction window which displays the relation tuples there are two functions under *Window*: With *Save* the content of the window can be saved in an html file. *Close* is for closing the window. *Additional Functions* contains only one function: *Fill in the Relation Field* copies the result of the *Show Nodes in Relations* operation to the relation field in the graph window. Note that doing so deletes all tuples from all relations that have already been shown in the relation field. This is particularly important to keep in mind for those relations which may contain 'any possible tuples' according to the result. Their sets will be empty after filling in the relation field, even if they previously contained tuples.

Optimize Graph helps to make the graph on the canvas more readable. Each configuration of nodes connected with a root node will be represented in a form which resembles a tree. The root node is placed in the first position to the left in the first line of an imaginary grid on the canvas. All other nodes are distributed over additional lines in the grid according to the shortest path leading to them from the root node. If there is more than one feature structure with a root node on the canvas, the feature structures will be placed under one another. Nodes which are not connected to any root node are ignored.

A simple example of graph optimization is shown in Figure 3.5. The graph on the left is converted into the graph on the right.

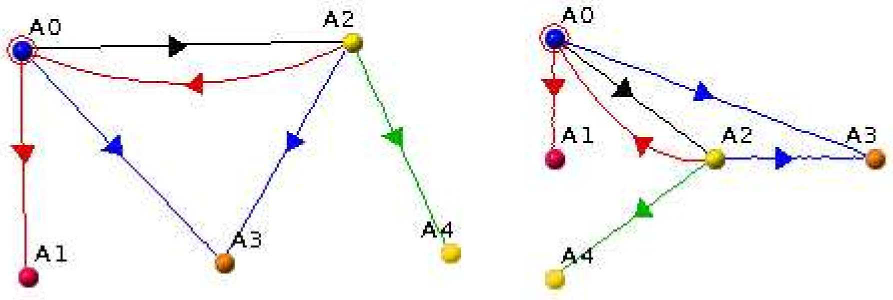

Figure 3.5: Graph optimization in MoMo

Chapter 4

Advanced Features

In this chapter, we will describe advanced features of MoMo which go beyond the basic functions initially needed in simple learning applications. The functions which we will discuss here are interesting for more advanced investigations of the logical properties of HPSG grammars and their models. They comprise, among other things, more flexible tools for dealing with large feature graphs, additional logical objects and operations, and an interface between the TRALE system and MoMo.

4.1 Special Syntax for List Descriptions

MoMo provides an interface which can recognize any appropriate set of logical symbols in signatures as symbols for the descriptions of lists. The advantage of declaring such symbols explicitly as symbols for describing lists is that their declaration creates an alternative, much more readable syntax for lists. If MoMo is told the list symbols in a signature, you can use conventional list notation with brackets (`[X, ..., Y]`) in addition to the canonical descriptions using the appropriate non-logical constants.

In order to activate the reserved list syntax, the supersort of the list signature must be declared. An example of a list signature is shown in Figure 4.1. In this example, the relevant supersort is the sort *list*.

```
type_signature
bot
   list
      nelist head:bot tail:list
      elist
.
```

Figure 4.1: A list signature

The supersort of the part of the signature which is used for the description of lists is declared in the menu *Options* of the note pad. *Options* → *Top-sort for Lists* opens a window in which you type in your top sort for lists. MoMo then checks whether your top sort has the right properties: It must have exactly two proper subsorts; one having no attributes appropriate to it, and one having two appropriate attributes with attribute values of the following kind: For one attribute the top sort of lists is appropriate, and for the other one, an arbitrary sort of the signature. If these conditions are not met, an error message will appear.

If you typically use a particular top sort for lists in your signatures, you may choose *Options → Preferences*. Under *Preferences*, you can declare a top sort for lists which MoMo will remember as the default setting.

For more details on the syntax of list descriptions, see the specifications in Chapter 7.

4.2 Font Settings for Presentations

The size of the fonts used for the signature and descriptions of the note pad and for the labels in the graph window can be set to any convenient value. The main application of this function is for the use of MoMo to make presentations with a data projector. The default setting of the font size is *12*, which is typically too small for classroom presentations.

The note pad window and the graph window contain the menu item *Font Size* under *Options*. Use *Options → Font Size* in the note pad to change the size of the letters in the signature and in the descriptions. Use *Options → Font Size* in the graph window menu to set the size of the attribute labels and sort labels of graphs at your convenience.

The default settings for the font size in the note pad and in the graph window can be changed independently from one another in the note pad menu under *Options → Preferences*. The dialog box offers two fields for font size settings, *Font Size in Note Pad* and *Font Size in Interpretation*. You may enter any standard numerical value for the font size, which MoMo will remember when you restart it.

4.3 Preferences

You can fix a number of convenient personal settings with *Options → Preferences* in the note pad window. Your preferences will be saved to the file `.MoMoPreferences` in your home directory. The default settings for opening a certain mmp file after starting MoMo and the preselection of a top sort for lists can remain empty.

Default .mmp file If an mmp file is specified in this field, it will automatically be opened each time MoMo is started. The default file may be a local file or a file on the Internet. Paths to local files are prefixed by '`file:`'. For example, with `file:/home/mueller/start.mmp` the user named mueller opens the file `start.mmp` in the home directory when MoMo is started. Files on the Internet are specified correspondingly with the prefix '`http:`'.

Default interpretation The default interpretation can be any name assigned to an interpretation in the *Interpretations* area of the note pad. If no default is specified, MoMo opens the first interpretation in this area, if there are any.

Default description Parallel to a default interpretation, a default description can be the name of any description card of the *Descriptions* area of the note pad.

Show labels Check this option if you would like sort labels and attribute labels to appear next to the nodes and attribute arcs in interpretation windows as a default.

Show node numbers Here you can select as the default setting the display of the alphanumeric names that MoMo assigns to each node in an interpretation.

Feature structure semantics and *Standard semantics* are alternative radio buttons for the selection of your preferred description interpretation.

Font size in note pad determines the font size in the signature area and on the description cards of the note pad.

Font size in interpretation determines the font size of the sort labels and attribute labels in the interpretation window.

Default browser gives you a selection of browsers from which you can choose the one you use. When the MoMo help function is called MoMo will try to open this browser first n order to display the MoMo manual. If it is not available, MoMo will open the first browser it finds on your system, just as in the case when no default setting is provided here.

4.4 Graph Manipulations

Besides the basic operations on graphs described in Section 3.2.1, MoMo also incorporates a few additional functions which make modifying graphs easier, especially when working with large feature graphs. These are described here in turn.

Copying, cutting and pasting (pieces of) graphs Creating a large graph or several large graphs that might differ slightly is made much easier by the possibility of cutting, copying and pasting (pieces of) graphs. These operations allow you to easily delete pieces of a graph on the canvas in one step, or to insert new parts from elsewhere on the canvas into a graph without designing it step by step.

To mark a collection of nodes and arcs for copying, cutting or pasting, click somewhere in an empty space on the canvas with the left mouse button and move the mouse over the canvas while keeping the button pressed. A red rectangle will appear on the canvas. All nodes within the rectangle and all arrows between these nodes are marked as being active.

To copy or cut a selected section of a graph, press the right mouse button. A small pop-up menu will appear next to the cursor. Select the appropriate menu item (*Copy to Clipboard* or *Cut to Clipboard*) in the pop-up menu. To paste the selected graph section from the clipboard onto the canvas, first choose a location on the canvas by clicking on the canvas with the left mouse button. Then select *Paste from Clipboard* in the pop-up menu to place the pieces of the graph from the clipboard on the canvas. The previously chosen spot on the canvas will be the upper left corner of the pasted graph element(s).

All functions described here are also available in the menu bar of the graph window under the item *Graph-Edit*.[1]

Change the sort of the node is an additional option for editing graphs. It is located under *Graph-Edit* → *Change Sort of the Node.* With this function it is possible to change the sort of the marked node to any maximally specific sort declared in the signature, without having to delete the node. Deleting a node also erases all arcs connected to it. Renaming an existing node might thus help you avoid extra work otherwise required in recreating arcs.

Dragging node and arrows labels Sort labels and attribute labels are automatically added to the canvas by MoMo if the check box *Options* → *Show Node Label* is activated in the graph window. Although MoMo intelligently distributes labels on the canvas, labels may overlap in complex graphs, or it may not be clear which label belongs to which node or arc

[1] See also the paragraph entitled *Known Problems* at the end of this section for bugs which exist in some Java environments.

if these labels and arcs or nodes are next to each other. To clarify you may move labels. To drag a node or arrow label across the canvas, simply seize it by clicking on it with the left mouse button and keeping the button pressed, while dragging the label to the place where you would like it to be.

Label positions which are the result of dragging the labels across the canvas are not preserved when you save the mmp file with the interpretation in question.

If you move a node, MoMo will again attach its label and the labels of all connected arrows next to their new positions, no matter whether or not the previous positions of the labels were determined automatically by MoMo or manually by you.

Known problems In some versions of Java, the functions of the pop-up menu for the copying, cutting and pasting of pieces of graphs do not work. If you encounter this problem, simply use the corresponding items under *Graph-Edit* in the graph window.

There are a few additional functions for manipulating graphs which will be described in Section 4.6 below, since these are most often needed when interpretations are imported from TRALE.

4.5 Interpretations and Models without Feature Structures

King-style interpretations MoMo was originally designed as a tool for becoming familiar with and exploring the kind of feature structure semantics often given to the logical languages of HPSG in connection with computational applications. As argued in [Richter, 2004a], the choice of feature structures as models of linguistic grammars is problematic from a philosophical point of view. Alternatives such as the one implemented here might be more adequate in theoretical linguistics.

MoMo provides a second denotation for its logical languages which is close to the semantics for HPSG grammars suggested in [Richter, 2004a]. Since this semantics is an extension of the semantics for HPSG description languages originally proposed by Paul King in [King, 1999] (and in earlier work by King), we will sometimes refer to this kind of semantics as *King-style semantics*. According to Paul King's metatheory, interpretations of a grammar do not contain feature structures. Feature structure models are viewed as a specialized kind of mathematical model which should be avoided in linguistics for theoretical reasons. The model theory of King is more general than the feature structure models of computational HPSG and uses standard constructions of logical models instead.[2]

At the graphical surface there is no striking difference between the two conceptions of the meaning of grammars. King-style semantics is switched on by activating the radio button *Options* → *Semantics* → *Standard Semantics*, which switches off *Options* → *Semantics* → *Feature Structure Semantics*. Alternatively, *Standard Semantics* can be chosen as the default setting of MoMo in *Options* → *Preferences* of the note pad.

An interpretation in standard semantics obviously does not contain feature structure graphs. The only graphical effect of this difference consists in the absence of root nodes.

This difference also affects the logical operations. The right column of the graph window is different when you work with standard semantics. Since there are no feature structures,

[2]For detailed discussions of these issues, see [Richter, 2004b, pp. 99–102, 106–108, and 114–134] and [Richter, 2004a, Section 2.4].

the button *Check Feature Structure* is not displayed. Since there is no root node which could be checked for constraint satisfaction, *Check Satisfaction* for the entire interpretation on the canvas is not displayed either.

The following operations are available:

- *Obeys Signature* performs a well-formedness check on the entire configuration of all nodes and arrows on the canvas.
- *Check Modelling* checks whether the configuration of nodes and arrows on the canvas models the set of descriptions on the active description card in the note pad.
- *Check Satisfaction* checks whether the selected node satisfies the description(s) on the active description card of the note pad.

Note that the function *Optimize Graph* and the option *Show Node Letters* do not work in the standard semantics mode of MoMo, because they presuppose the presence of a distinguished root node. Similarly, *Show Abstract Feature Structure* cannot produce any output, since there are no concrete feature structures for which corresponding abstract feature structures could be computed.

Additional Functions → *Show Nodes in Relations* behaves differently in standard interpretations from the feature structure-based semantics. In a feature structure based semantics, relations are relations within feature structures. In other words, the nodes in the relation tuples are defined as nodes of single feature structures, and the set of tuples belongs to the feature structure in which the nodes occur. In standard semantics, the tuples of nodes in relations exist independently of feature structures. They are not even necessarily connected configurations of nodes, although this is typically intended in HPSG grammars.[3] It is important to remember that the (sets of) tuples which form the relations are an integrated part of the interpretation as a whole. However, we can determine relative to each entity in the interpretation which entities under it must be in a relation in order to model a relation definition. This is what *Show Nodes in Relations* does in standard semantics: Relative to each entity in the feature graph(s) given in the interpretation, it computes the nodes which must be in a relation in order for the interpretation to be a model of the relational description(s), presupposing that the definition of the relation is provided on the active description card. If the definition of the relation contains a second relation symbol, the definition of this second relation must also be contained on the active description card.

The same syntactic restrictions apply to the relational descriptions as in feature structure semantics (see Section 3.2.3).

4.6 TRALE Interface and Related Tools

This section describes the interface between MoMo and TRALE and a number of functions for editing graphs which are particularly useful when large structures are created using TRALE output.

Special properties of models which are derived from parsing in TRALE are their size and their initial partiality, which require a somewhat different treatment from standard MoMo models constructed by MoMo users themselves. In order to shield the more standard use

[3]It may also be enforced with additional restrictions on interpretations. See [Richter, 2004a] for a discussion.

of MoMo from the effects of tools which can deal with partial models, these tools are only available after explicitly requesting them by typing in a password. The possibilities which then become available are meant only for experienced users who have a good grasp of the logical languages and the model theory of HPSG grammars. For inexperienced users it is advisable to avoid these while using MoMo as a tool for learning the logical foundations of HPSG.

Getting started: Advanced Functions To gain access to MoMo's TRALE-related functions, you must activate the extended menus. This is achieved by selecting *Options* → *Advanced Functions* in the note pad. A dialog box appears and requests a password. Type in `advanced` to activate the extended menus under the menu items *Graph-Edit* and *Additional Functions* in the menu bar of the graph window.

We will first turn to the MoMo-TRALE interface, located under *Additional Functions*, and then describe additional tools for editing graphs under *Graph-Edit*.

Importing files from TRALE With *Additional Functions* → *Import Model from TRALE* it is possible to import files which are in the format of the parse output of TRALE. The usual dialog box appears, and you can select any files generated as parse results. The file with TRALE's parse output that you wish to import must be in the pretty printer format of TRALE. To generate such an output, TRALE's graphical user interface Grisu needs to be switched off. This is done by typing the command `grisu_off` at the Prolog prompt. Then the desired string can be parsed using the regular parse predicate, `rec/1`, in TRALE. The output obtained from `rec/1` is finally saved to a text file and can be imported by MoMo.

Upon loading the TRALE output file, MoMo creates a corresponding graph as far as is specified by TRALE's output. MoMo infers the root node of the graph and displays it as a (partial) feature structure, and then executes graph optimization on the feature structure it computes from TRALE's parse output. Structure sharing occurs where it is explicitly required by tags in the parse result, other logically valid possibilities are not considered. The graph contains all attributes and sorts provided by the imported file, and nothing else. This means necessarily that the graph generated from the TRALE file will typically not be a well-formed interpretation, because it is not totally well-typed and sort resolved. The extended graph editing tools under *Graph-Edit* will provide the means to work with these *partial feature structures*.

The import function presupposes that the note pad of MoMo contains a signature that declares the sort symbols, attributes and appropriateness conditions of the TRALE output. Since TRALE signatures are special cases of MoMo signatures, the underlying TRALE signature can be imported for this purpose (using *Signature* → *Open Signature* in the note pad menu, or the *Open Signature* button directly above the signature window in the note pad). Note, however, that TRALE uses *atoms* for the elements of the phonology list, and these atoms are not declared in the TRALE signature. For this reason, you need to extend the TRALE signature manually, with declarations of these additional sorts in order to be able to import a parse output file containing atoms for the elements on the PHON list.

Assume the structure you would like to import has the phonology *she walks*, with the TRALE atoms `she` and `walks`. They are not declared in the TRALE signature you opened in the note pad. Therefore, the TRALE signature should be extended manually in the note pad with the following sort hierarchy under `bot`:

```
phonstring
  she
  walks
```

If you wish, you may declare the sorts `she` and `walks` as atoms.[4]

Before importing files from TRALE make sure to switch off the check box *Options → Show Node Letters* in the graph window as described in Section 3.2.3 on page 20. Working with node letters switched on is computationally expensive, which is an important consideration when working with graphs of the size typically obtained from the output of TRALE grammars.

If the feature structure which MoMo constructs from the TRALE file is partial—which it usually is—it is, of course, not a well-formed feature structure in the sense of HPSG as presented by [Pollard and Sag, 1994]. However, the function *Obeys Signature* will accept partial feature structures that comply with the demands of the signature. The partial structure is accepted as well-formed if the non-maximal sorts are appropriate, and if all nodes that are assigned a non-maximal sort symbol bear all attribute arcs that the signature declares appropriate to them.

Nodes which are not maximally specific or have attribute arcs missing are displayed in gray. Occasionally, we refer to such nodes as *virtual nodes*, since these are not real nodes of a totally well-typed and sort resolved feature structure. The extended menus provide tools which allow you to expand such virtual nodes to real structures automatically, depending on the shape of the underlying signature. The relevant functions are called *Fill Substructures of all Nodes* and *Fill Substructure of the Selected Node*, respectively. They are explained below. Conversely, you may want to *hide* a *substructure* by turning its local root into a virtual node if the substructure in question is not relevant to the satisfaction or model checking you would like to perform on the feature structure. This can considerably speed up your computations.

For your convenience, here's a short summary of the steps necessary in creating parse results and importing them into MoMo:

1. In TRALE: Switch off Grisu (`grisu_off` at the Prolog prompt) and parse the sentence the model(s) of which you are interested in using the parse predicate `rec/1`.

2. Save the parse result in a separate file, which we will call `output`.

3. Load the TRALE signature in MoMo (with *Signature → Open Signature*) and add undeclared sorts such as the elements on PHON lists to the signature, if there are any.

4. Load the file `output` (with *Additional Functions → Import Model from TRALE*).

5. The feature structure that MoMo will create will typically be a partial feature structure. It can (semi-) automatically be turned into a totally well-typed and sort resolved feature structure using the functions *Graph-Edit → Fill Substructure of the Selected Node* and *Graph-Edit → Fill Substructure of All Nodes.*

Extended graph editing When the *Advanced Functions* are activated, a few new items appear under *Graph-Edit* in the menu bar of the graph window. We will discuss them in top down order:

[4]For a discussion of extensional sorts and atoms, see the TRALE and ALE manual at `www.ale.cs.toronto.edu/docs/man/ale_trale_man/index.html`

Hide Substructure deletes the substructure of the selected node. The sort of the selected node changes to its closest supersort which has no attributes appropriate to it. This means the relevant supersort does not inherit any attributes from any of its own supersorts, and there are no attributes appropriate to it. If there is no such supersort for the sort of the selected node, an error message appears. If such a supersort is found, the selected node changes to a virtual one with the new sort label. The color of the node changes to gray.

This function can help you to efficiently reduce the number of nodes in large structures. The idea is to ignore parts of the structure which are not interesting for the computations one would like to perform. An example of such an 'uninteresting' structure is typically phonology, but any kind of structure might turn out to be uninteresting under certain circumstances. The structure resulting from hiding some substructure will still obey the signature, but it will no longer be a feature structure in the sense of HPSG, because not all its nodes are of a maximally specific sort.

Make Virtual Node is similar to *Change Sort of the Node*. However, in this case the sort assigned to the selected node must not be maximally specific. The node becomes gray after a new sort has been assigned to it.

Fill Substructure of the Selected Node creates a complete well-formed substructure of the selected node according to the signature. All arrows representing the attributes, appropriate for the sort of the selected node, are added to the graph, and their values are treated accordingly. The resulting graph is optimized and displayed in the graph window.

For certain signatures, MoMo constructs a cyclic structure, since a cyclic structure is the only possibility to create a finite substructure for some nodes. This situation arises if there is a recursive loop in the signature. Figure 4.2 shows an example.

```
type_signature
bot
   rec head:bot tail:rec
.
```

Figure 4.2: A signature fragment which leads to a cyclic extension

At a node of sort *rec* MoMo creates a TAIL arc which points back to the *rec* node because the attribute TAIL requires another *rec* value. If MoMo created a separate *rec* node, the completion process would lead to an infinite structure.

Fill Substructure of all Nodes is a generalization of the previous function and allows you to expand a partial feature structure quickly. When it is executed, MoMo constructs appropriate substructures for all nodes in the graph. Inappropriate arrows will not be deleted should they exist. These can easily be found using *Obeys Signature*.

When the *Advanced Functions* are turned on it is possible to edit the signature without losing existing interpretations. To edit the signature even when interpretations of the old signature already exist, select *Edit Signature*. The background color of the signature changes to white and the signature is editable. No warnings appear. After editing the signature make it uneditable again (using the *Edit Signature* function). All changes will automatically be transferred to the graph window. If the graph contains nodes and arrows for sorts and attributes which are no longer in the signature, error messages appear. The color of these nodes and arrows changes to gray. Relations which are no longer declared in the signature disappear.

7.3 Notation for Lists

A signature with the appropriate feature geometry is called a *list signature*.

Using the symbols provided by a list signature, we can describe feature structures representing lists in exactly the same way in which we describe any other feature structure with any other non-logical symbols of the signature. However, a more perspicuous notation would be very welcome and is quite often used. Instead of describing a singleton list containing one green parrot as `(head:(parrot,color:green),tail:elist)`, people prefer to write `[(parrot,color:green)]`, which is much more readable. The advantages of this notation become even more striking when the lists get longer. To appreciate this, just compare

```
(head:(parrot,color:green),
 tail:(head:(canary,color:yellow),
       tail:elist))
```

with an expression as transparent as
`[(parrot,color:green),(canary,color:yellow)]`.

The relationship between the two notations is so perspicuous that we do not really have to bother about integrating an additional kind of notation into the syntax of our formalism. In fact this would make the definition of our description languages unduly complicated. This kind of situation is usually handled by introducing a syntactic convention on top of the formalism itself.[1] We simply state our signature of lists as we have always done and explain how we want to use a few additional notational devices to make our lives easier. For example we might include the HEAD-and-TAIL variant of list signatures in our signature, and then we would say that, by convention, we may sometimes write `[]` for `elist` and enumerate the elements of lists using square brackets: `[`δ_1`, ..., `δ_n`]` stands for the expression in (1):

(1) head:δ_1,
...,
tail$_1$:...:tail$_{n-1}$:head:δ_n,
tail$_1$:...:tail$_n$:elist

A second abbreviatory notation which is often used together with the square bracket notation for lists is the vertical slash, |. The vertical slash separates a description of the first n elements on a list from a description of the tail of the list after the nth element. The following sketch pictures the use of this notation schematically:

(2) head:δ_1,
...,
tail$_1$:...:tail$_{n-1}$:head:δ_n,
tail$_1$:...:tail$_n$:$\mathcal{BOX}$

may be written as

[$\delta_1, \ldots, \delta_n$| $\mathcal{BOX}$]

[1]Sometimes additional alternative notations which do not change anything substantial in a formal language are called *syntactic sugar*.

In (2), $\mathcal{BOX}$ stands for an arbitrary box, which will normally be a box that describes lists. This notation is particularly useful when we want to refer to the tail of a list with some variable, as in the following example:

(3) $[(\texttt{cat}), (\texttt{dog}) | \boxed{1}]$

The expression in (3) is satisfied by feature structures representing a list whose first element is a cat and whose second element is a dog. We use the variable $\boxed{1}$ to refer to the tail of the list. This variable may be used again elsewhere in a larger description, of which our description may be a part. This would allow us to say that the tail of the list stands in a relation with other lists in feature structures which satisfy or model our description.

If we wanted to be really exact, we could describe these new notational conventions in an appropriate standard notation of mathematics. However, as long as it is plain enough what we want to abbreviate, and as long as we are consistent with our use of notation, an explanation such as the one given here is sufficient.

7.4 Concrete Feature Structures

### to be added in order to give technical substance to Section 5.3 ###

Appendix A

Error and Warning Messages

A.1 Alphabetical Ordering

- Feature structure check failed. The node number * belongs to a substructure without a root node.
- Feature structure check failed. The node number * is the second root node in the substructure.
- Feature structure check failed. The nodes of the tuple * do not all belong to the same feature structure.
- Feature structure check successful.
- Signature check and model check successful.
- Signature check failed.
- Signature check failed because a number (*) of arguments of the relation * in the relation field are missing.
- Signature check failed because the feature structure is not well-typed.
- Signature check failed because the feature structure is not well-typed. The attribute * is not appropriate to the sort * or has an incorrect value.
- Signature check was successful but model check failed.
- Signature check was successful but satisfaction check failed.
- Signature check, feature structure check and model check successful.
- Signature check, feature structure check and satisfaction check successful.
- Signature check successful.
- Syntax check failed. When using the special list notation in descriptions, you have to specify which part of your signature may be alternatively notated in list syntax. To do so you must specify the top sort of the list hierarchy in the menu OPTIONS, Top Sort for Lists.

- Syntax check failed. When using the special list notation in descriptions, you have to specify which part of your signature may be alternatively notated in list syntax. To do so you must specify the top sort of the list hierarchy in the menu OPTIONS, Top Sort for Lists. The top sort of lists has two subsorts. One of them has no attributes appropriate to it; the other one has two appropriate attributes. For one of them the top sort of lists is appropriate, for the other one an arbitrary sort of the signature is appropriate.
- Syntax check successful.
- The relation * is not well defined. Arity 0 is not allowed according to specification.
- The attribute * occurs more than once at node number * .
- The feature structure is not total. Attributes (*) are missing from node number *.
- The feature structure is not well-typed. The attribute * is not approppriate for node number * .
- The feature structure is not well-typed. The attribute * of node number * has an incorrect value.
- The node you queried is not totally well-typed.
- The node you queried is totally well-typed.
- The node you queried is totally well-typed, but satisfaction check failed.
- The node you queried is totally well-typed, satisfaction check successful.
- The relation * has an incorrect number of arguments.
- The signature does not contain an attribute * .
- The signature does not contain a relation * .
- The signature does not contain a sort * .
- The signature should begin with 'type_hierarchy'.
- The sort * is not a legitimate supersort of a list signature. The supersort of lists has two subsorts. One of them has no attributes appropriate to it; the other one has two appropriate attributes. For one of them the top sort of lists is appropriate, for the other one an arbitrary sort of the signature is appropriate.
- You are not working with the feature structure semantics. Chose Options->Semantics->Feature Structure Semantics if you want to see the AFS which corresponds to your graph.
- There is no description at all.
- There are no feature structures at all.

A.2 Thematic Ordering

A.2.1 Syntactic Messages

- Syntax check successful.

Errors in the signature

- The relation * is not well defined. Arity 0 is not allowed according to specification.
- Syntax checking failed because some values of the attributes are not defined in the signature : * .
- The signature should begin with 'type_hierarchy'.

Errors in the description

- Syntax check failed. When using the special list notation in descriptions, you have to specify which part of your signature may be alternatively notated in list syntax. To do so you must specify the top sort of the list hierarchy in the menu OPTIONS, Top Sort for Lists.
- Syntax check failed. When using the special list notation in descriptions, you have to specify which part of your signature may be alternatively notated in list syntax. To do so you must specify the top sort of the list hierarchy in the menu OPTIONS, Top Sort for Lists. The top sort of lists has two subsorts. One of them has no attributes appropriate to it; the other one has two appropriate attributes. For one of them the top sort of lists is appropriate, for the other one an arbitrary sort of the signature is appropriate.
- The relation * has an incorrect number of arguments.
- The signature does not contain an attribute * .
- The signature does not contain a relation * .
- The signature does not contain a sort * .

A.2.2 Semantic Messages

- There is no description at all.
- There are no feature structures at all.

Messages by well-typedness check

- Signature check failed.
- Signature check failed because a number (*) of arguments of the relation * in the relation field are missing.
- Signature check failed because the feature structure is not well-typed.

- Signature check failed because the feature structure is not well-typed. The attribute * is not appropriate to the sort * or has an incorrect value.
- Signature check was successful but satisfaction check failed.
- Signature check, feature structure check and satisfaction check successful.
- Signature check successful.
- The feature structure is not total. Attributes (*) are missing from node number *.
- The feature structure is not well-typed. The attribute * is not approppriate for node number * .
- The feature structure is not well-typed. The attribute * of node number * has an incorrect value.
- The node you queried is not totally well-typed.
- The node you queried is totally well-typed.

Messages by feature structure check

- Feature structure check failed. The node number * belongs to a substructure without a root node.
- Feature structure check failed. The node number * is the second root node in the substructure.
- Feature structure check failed. The nodes of the tuple * do not all belong to the same feature structure.
- Feature structure check successful.

Messages by satisfaction check

- The node you queried is totally well-typed, but satisfaction check failed.
- The node you queried is totally well-typed, satisfaction check successful.

Messages by model check

- Signature check and model check successful.
- Signature check was successful but model check failed.
- Signature check, feature structure check and model check successful.

A.2.3 Other Messages

- You are not working with the feature structure semantics. Chose Options->Semantics->Feature Structure Semantics if you want to see the AFS which corresponds to your graph.

Appendix B

Functions in MoMo

B.1 Note Pad Functions

B.1.1 Menu Bar

File

- New Signature Page
- Open
- Get Web Resource
- Save Page
- Save Page as
- Close

Text-Edit

- Cut to Clipboard
- Copy to Clipboard
- Paste from Clipboard

Sigtnature

- Edit Signature
- Open Signature
- Print Signature
- Check Syntax
- Edit Signature
- Open Signature

- Print Signature
- Save Signature as
- Sort Signature Alphabetically

Description

- New Description
- Delete Description
- Rename Description
- Save Description as
- Check Description Syntax
- Print Description
- Open Description

Interpretation

- New Interpretation
- Clone Interpretation
- Delete Interpretation
- Rename Interpretation
- Save as Picture
- Print Current Interpretation

Options

- Font Size
- Top-sort for Lists
- Advanced Functions

  Preferences

- Default .mmp file
- Default interpretation
- Default description
- Top-sort for lists
- Show labels

- Show node numbers

Semantics

- Feature Structure Semantics
- Standart Semantics

Font sizes

- Font size in notepad
- Font size in interpretation

Info

- About MoMo

Help

- MoMo Manual

B.1.2 Signature Area Functions

- Signature is editable
- Check Syntax
- Edit Signature
- Open Signature
- Print Signature

B.1.3 Description Area Functions

- New
- Delete
- Rename
- Check Syntax
- Print
- Open
- Reshuffle

B.1.4 Interpretation Area Functions

- New
- Dublicate
- Delete
- Rename
- Print
- Reshuffle

B.2 Graph Window Functions

B.2.1 Menu Bar

Interpretation

- New Interpretation
- Clone Interpretation
- Delete Interpretation
- Rename Interpretation
- Save as Picture
- Print Current Interpretation

Graph-Edit

- Undo Last Add/Delete
- Clear Graph
- Proceed
- Cut Substructure
- Chamge Sort of the Node
- Cut to Clipboard
- Copy to Clipboard
- Paste from Clipboard
- Hide Substructure
- Make Virtual Node
- Fill Substructure of the Selected Node
- Fill Substructure of all Nodes

Verify

Feature Structure

- Obeys Signature?
- Is Feature Structure?
- Satisfies Description?
- Models Description?

Selected Node

- Well-typed?
- Satisfies Description?

Options

- Font Size
- Grid Size
- Size of Relation Field
- Show Node Labels
- Show Node Letters
- Scrollbars for Canvas

Semantics

- Feature Structure Semantics
- Standart Semantics

Additional Functions

- Sort Attributes and Sorts Alphabetically
- Show Abstract Feature Structure
- Show Nodes in Relations
- Optimize Graph
- Import Model from TRALE

B.2.2 Drawing Tools

- Activating sorts and attributes

B.2.3 Canvas

- Creating nodes and arcs
- Removing nodes and arces
- Creating-and deleting-root nodes
- Dragging objects
- Editing relations
- Marking objects

B.2.4 Status Bar

B.2.5 Logical Fuctions

Structure

- Obeys Signature
- Check Feature Structure
- Check Satisfaction
- Check Modeling

Selected Node

- Check Well-typedness
- Check Satisfaction

B.2.6 Graph Processing Functions

Remove Graph Coloring

- Proceed

Edit

- Clear Graph
- Undo Last Add/Delete

Bibliography

[King, 1999] King, Paul J. 1999. Towards Truth in Head-driven Phrase Structure Grammar. In Valia Kordoni (ed), *Tübingen Studies in Head-Driven Phrase Structure Grammar*, (= *Arbeitspapiere des SFB 340, Nr. 132, Volume 2*), 301–352. Eberhard-Karls-Universität Tübingen.

[Meurers et al., 2002] Meurers, W. Detmar, Penn, Gerald, and Richter, Frank 2002. A web-based instructional platform for constraint-based grammar formalisms and parsing. In Dragomir Radev and Chris Brew (eds), *Effective Tools and Methodologies for Teaching NLP and CL*, 18–25, New Brunswick, NJ: The Association for Computational Linguistics. *Note:* Proceedings of the Workshop held at the 40th Annual Meeting of the Association for Computational Linguistics. 7.–12. July 2002. Philadelphia, PA.

[Pollard and Sag, 1994] Pollard, Carl and Sag, Ivan A. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

[Pollard, 1999] Pollard, Carl J. 1999. Strong generative capacity in HPSG. In Gert Webelhuth, Jean-Pierre Koenig, and Andreas Kathol (eds), *Lexical and Constructional Aspects of Linguistic Explanation*, 281–297. CSLI Publications.

[Richter, 2003] Richter, Frank 2003. Erfahrungen bei der Erstellung des Web-basierten Kurses “Grammar Formalisms and Parsing”. *Sprache und Datenverarbeitung. International Journal for Language Data Processing*, 27.1–2:95–111.

[Richter, 2004a] Richter, Frank 2004a. Foundations of Lexical Resource Semantics. *Note:* Habilitation Thesis. Seminar für Sprachwissenschaft, Universität Tübingen.

[Richter, 2004b] Richter, Frank 2004b. *A Mathematical Formalism for Linguistic Theories with an Application in Head-Driven Phrase Structure Grammar*. Phil. dissertation (2000), Eberhard-Karls Universität Tübingen.

[Richter et al., 2002] Richter, Frank, Ovchinnikova, Ekaterina, Trawiński, Beata, and Meurers, W. Detmar 2002. Interactive graphical software for teaching the formal foundations of Head-Driven Phrase Structure Grammar. In Gerhard Jäger, Paola Monachesi, Gerald Penn, and Shuly Wintner (eds), *Proceedings of Formal Grammar 2002*, 137–148.

[Richter et al., 1999] Richter, Frank, Sailer, Manfred, and Penn, Gerald 1999. A Formal Interpretation of Relations and Quantification in HPSG. In Gosse Bouma, Erhard Hinrichs, Geert-Jan M. Kruijff, and Richard T. Oehrle (eds), *Constraints and Resources in Natural Language Syntax and Semantics*, 281–298. CSLI Publications.