Seminar für Sprachwissenschaft
Abteilung Computerlinguistik
Universität Tübingen

October 30, 2003

The MorphMoulder User's Manual

Frank Richter
Ekaterina Ovchinnikova

1 Introduction

The *Morph Moulder*, MoMo, is a tool for exploring the relationship between objects, signatures and descriptions in a typed feature logic. More specifically, MoMo is an 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, 2000], 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 is first and foremost designed as a tool for teaching the feature logic formalism of HPSG by projecting its highly abstract mathematical concepts on a graphical 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 easy to manipulate objects without having to wade through a considerable amount of Greek symbols and disturbing squiggles in abstract mathematical definitions first. Once the 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 at that point it is only necessary to put intuitively well-known ideas on an explicit mathematical basis. The compatibility of MoMo with the TRALE system highlights the relationship between theoretical grammars and approximations to them in computationally efficient environments.

2 Running MoMo

MoMo requires Java 1.3.1 (or higher) on Solaris 5.7, Solaris 5.8 and current versions of Redhat or SuSe Linux. 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 give a few hints on how to approach an installation of 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 calling
`./MoMoInstaller`
in the `MoMo/` directory. The installation script opens a few dialog boxes in which you can choose between a number of installation options. The first screen gives information about the permissions that Java needs for running MoMo; then the user may choose between running MoMo locally (offline installation) or remotely from a web page (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 somewhere on the Internet. This has the advantage that, as long as you use the on-line release, you do not have to update MoMo at any time, since you will automatically use 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 do an on-line and an offline installation simultaneously. Which MoMo you are using at any given time then depends on where you start it—on your local machine or off the Internet.

In the next step, MoMo needs information on the location of the MoMo code for the chosen installation type. For a local installation, the installation script suggests the directory in which MoMo resides at the time of the installation. Pressing the "Finish" button completes the installation.

To be able to use the command `momo` for starting MoMo from anywhere in your directory system, the operating system needs to be told where the binaries of MoMo are located. Make sure that the `PATH` variable of your shell contains the path to the MoMo binary file *momo* on your machine. If you do not know how to do that, please consult the manuals for your operating system or your systems administrator. If your shell does not know about the location of the MoMo binaries, MoMo can only be started by going into the `MoMo` directory and calling MoMo from there (by typing `./momo` or `./momo &`).

2.2 Remarks on Installing MoMo under Windows[1]

The following remarks might help you if you try to run MoMo under windows.

The MoMo tarball can be unpacked with WinZip.[2] Unpacking creates a directory `MoMo`. A good location for it may be `D:\MoMo`. To be able to start MoMo locally, you need to put a file named `.java.policy` into 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`.

We need to declare in the `.java.policy` file where the source code of MoMo lives on your account. For that purpose, you need to edit the first line of the file (fully shown below), which starts with `grant codeBase`. Assume that you have put MoMo in the directory `D:\MoMo`. Then your `.java.policy` file needs to contain at least the following statements:

[1]We are grateful to Holger Wunsch for sharing his ideas on a Windows installation of MoMo with us.
[2]http://www.winzip.com

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

If you would like to work with MoMo off a web server, you need to grant MoMo the same rights as if you started a local copy. Thus, you need to set the same permissions as for a local copy of MoMo in the `.java.policy` file by declaring the web address from which you want to start MoMo. Assume that, in addition to using a local copy of MoMo, you want to work with a copy of MoMo that is located at the URL `www.sfs.uni-tuebingen.de/~fr/teaching/ws02-03/momo/` Your `.java.policy` now needs to contain a second set of permissions that is identical to the previous one, except that in its first line it says

```
grant codeBase "http://www.sfs.uni-tuebingen.de/~fr/teaching/ws02-03/momo/-" {
```

In sum, for each place, be it on your local account or on the web, from which you would like to start a copy of MoMo, you need to declare the location and the rights of that instance 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.[3]

When `.java.policy` is installed, you can go to the directory of MoMo in your DOS box and start MoMo with `appletviewer applet.html`. You may also start 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 start MoMo by clicking on `momo.bat`.

3 Working with MoMo

MoMo has two main windows. The first window, which we will call the *note pad*, comes up when MoMo is started. From this window the second window, which we will refer to as the *graph window*, can be started as one of the possible functions of MoMo. Logically, the functions of the note pad have to do with the description language of HPSG. The graph window provides functions that have to do with *models* or, more generally speaking, *interpretations* of the logical expressions created in the note pad 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 it might contain error messages.

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:

On the left half of the note pad, you find the *signature* editor. Well-formed descriptions as well as interpretations are defined relative to signatures. They are dependent on signatures, and without a signature descriptions cannot be evaluated relative to interpretations. Even more crucially, if a signature is changed, interpretations that might 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 that the logical language

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

may talk about, and a change in the signature may render entities impossible that could very well exist under the previous signature. To exclude the appearance of impossible objects in interpretations, signatures may no longer be changed once interpretations have been created for them.

The *descriptions* area (an editor for descriptions) is in the upper right quarter 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 window.

The relative size of the three areas in the note pad can be adjusted as necessary. The left part of the window belongs to 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 on top of the interpretations area, but the bar that separates the two can be moved up and down to adjust their size 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 can give a title to your signature (together with its set of descriptions and its set of interpretations). Some input to this line is obligatory.

The signatures of MoMo are RSRL signatures [Richter, 2000, p. 156]. Signatures are written according to the notational conventions for signatures of TRALE,[4] with an additional convention added for declaring relation symbols and the arity of the corresponding relations. Figure 1 shows an example.

Following TRALE conventions, the signature must begin with the declaration `type_hierarchy` in the first line, and it is concluded with a full stop 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 (`brown`), there is a line with the keyword `relations`, followed by the declaration of relation symbols. They 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).

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 been determined yet. With a red background, the signature is not editable, and you can create

[4]Except that, (1), sort hierarchies in MoMo are not required to have a unique top element. With 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 1: A signature with relations

interpretations. Descriptions can be created in both modes of the signature window. There 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.

Switching between the two modes can be done 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 while there are descriptions or interpretations, you will be warned that you lose them if you make the signature editable.

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).

The *Check Syntax* button allows to perform a syntax check of the signature. Results of the syntax check will be communicated through an interaction window. *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 a flexible and powerful editor of your choice and then import the signature into MoMo. Especially if you are planning to work with a multitude of similar signatures, this may be a very useful function.

3.1.2 Descriptions

In the description area, arbitrarily many *cards* can be opened. Each card may contain descriptions, whose 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 the full expressive power of RSRL.[5] For a specification of the syntax of well-formed expressions, see Section 6. Each description in the description editor must end with a full stop. Just like the editor window for signatures, the editor for description cards has three editing functions, 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 also insert comments anywhere on a description card by inserting the symbol `%`. Everything succeeding 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 darker color. All logical functions involving descriptions in the graph window are performed relative to the active description card. Each description card may contain any number of descriptions. Each description ends with a full stop.

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 be 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 left 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 part of the graph window, which is described in Section 3.2. In the present section, we describe the functions of the interpretations area of the note pad.

With *New* you open a new set of graphs. At any given time, at most one graph window can be open. 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,

[5]With the exception of *chains*, which are not included in the MoMo implementation.

corresponding to the file cards of the description section. The active radio button is marked by a black dot in the middle, all others are empty circles. Each radio button is followed by the name that you assigned to the 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* the user may 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 your printer for printing 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. It will then switch position with the radio button over it.

3.1.4 The Note Pad Menu Bar

The menu bar of the note pad window collects the functions of its three areas described in the previous sections. In addition, it provides a number of additional functions for each area, and for the entire 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 file format with the suffix .mmp.

The menu item *Get Web Resource* contains a selection of links to web resources with MoMo files. In distributions of MoMo for use in classes, *Get Web Resource* provides links to examples and exercises. They are named after the course sections to which they belong. By editing the file `webresourceindex.xml` in the MoMo sources, you may easily adapt the selection of down-loadable files to your personal purposes.

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 to *Open Signature*. It saves signatures in ASCII format for use with other editors or with 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; the attribute declarations are likewise sorted alphabetically from left to right. This function can help significantly to find relevant declarations in an unfamiliar signature.

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 to save the active interpretation window in jpeg or gif format. The pictures may then be imported to other documents.

Options provides three useful functions:

Font Size for adjusting the size of the characters in the *signature* and *description* area. Useful for slide presentations with MoMo.

Top-sort for Lists for declaring the top sort of the part of the signature for which you want to use the abbreviatory bracketing notation for lists. See Section 4 for more explanations.

Preferences for general personal settings that are preserved after shutting down MoMo. The user may define default settings for opening particular files, interpretations and description cards after startup, and a permanent setting for the top sort symbol of lists for using the list notation in descriptions.

Settings in *Preferences* declare whether the interpretation window displays labels for the nodes and the arrows on the canvas, and whether or not the node numbers that MoMo automatically assigns to the nodes of graphs are displayed.

The default semantics used by MoMo can be set to either the feature structure semantics or to King-style interpretations.

Users may also define their preferred font size on the note pad and in interpretations.

All of these settings are described in much more detail in Section 4.

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 functionality.

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 among the balls representing entities of all the maximally specific sorts declared in the signature (of the note pad window) and among the arrows (for the attribute functions) that connect the entities on the canvas. The possible arrows are represented as colored lines with their attribute names next to them. The right column is a space for 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 performing 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 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. Section 4 is reserved for the description of some more advanced features of MoMo.

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, 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.[6]

The resources for drawing feature structures are the balls and arrows available from 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, there is a collection of arcs, labeled by the attributes of the underlying signature. If there are more nodes and arcs than what fits 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 each half of the left column, the one for nodes and the one 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, for creating an arc on the canvas, we need to activate it first on the panel in the left column. Nodes are then connected by a copy of the active arc by first clicking on the node where the arc is supposed to originate and then clicking on the node where the arc will 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 that makes it overlap 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 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.

Removing nodes and arcs Nodes are removed by clicking on them with the right mouse button. All arcs connected with them will automatically disappear together 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 feature structure can be reached by following a sequence of arcs that originates 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.

[6] More precisely, they are totally well-typed, sort resolved concrete feature structures. MoMo also provides a second kind of interpretation which—following a proposal by Paul King—does not work with feature structures. It is described in Section 4.

Dragging objects over the canvas The shape of feature structures on the canvas can easily be changed by dragging nodes to new places. Incoming and outgoing arcs will automatically follow the nodes. 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.

Editing relations Relations in HPSG are relations between nodes in feature structures.[7] Thus, relations are tuples of nodes, where the number of elements in the 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 relations, the part of the canvas where they would be represented is missing, while the part of the canvas for the graphs is bigger. 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 the user does anything, the set tuples of nodes that are in each relation is empty.

Nodes in relations are graphically represented by names that are assigned to them. MoMo assigns alphanumeric names to nodes when they are first created on the canvas.[8] To see the node names that 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.

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 put into a tuple in the active relation by clicking on it with the left mouse button while pressing the `<shift>` key. If there is not incomplete tuple in a relation, putting the next node into the relation will start a new tuple. The number of elements in the tuple is inferred by MoMo from the arity of the relation, and missing elements in the tuple are indicated by an underscore in each argument position of the relation that still needs to be filled by a node. Each click with the left mouse button on a node (while pressing `<shift>`) fills another argument slot with a node. When all slots of a tuple are filled, adding another node to a relation will start a new tuple again.

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

3.2.2 Logical Operations on Feature Structures

In the column to the right of the canvas we see the logical functions panel of MoMo. This is where the logical operations of MoMo can be invoked and where MoMo signals the result 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:

[7] More accurately, in those formalizations of HPSG that rely on a feature structure semantics, relations are relations between nodes in feature structures.

[8] 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 user may safely ignore the particular names of nodes.

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 will be outlined in gray that proved correct. 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 may ignore those parts of the configuration that can not 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 diagnostics of what is wrong. In that diagnostics, it refers to the nodes by the names that MoMo has assigned to them. The node names can be displayed next to each node on the canvas by setting *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 if 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, feature structure checking tests whether for each configuration of nodes on the canvas that is connected by arcs there is exactly one designated root node such that each other node in the connected configuration can be reached from the root node by following a sequence of arcs.[9]

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 information why.

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 there is more than one feature structure on the canvas, it is checked for each of them whether it satisfies the set of descriptions. If each feature structure satisfies the set of descriptions, it will be outlined in green, and the *Success* light lights up.

[9]MoMo is actually a bit more sophisticated than the general idea of feature structure checking we have just described. 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 needs to be well-formed in the sense that it has 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 no root node belongs to the substructure of any other root node.

In case the red light signals *Failure*, two cases can be distinguished:

First, 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 say why. Mistakes in appropriateness and well-formedness of feature structures are indicated the same way as with the operations *Obeys Signature* and *Check Feature Structure*.

Second, 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. In that case, the feature structure is outlined in gray to indicate the compatibility with the signature, 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 satisfaction checking. The red outline can be removed by pressing the *Proceed* button or by performing any other operation.

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

Just like 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.

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, model checking fails. MoMo provides a precise analysis of which parts of the feature structures lead to a failure of model checking. All parts of feature structures that could be 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 description card receive a black circle around them.

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 it 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 indicted in the same way as for satisfaction checking of feature structures.

In the lower right corner of the graph window there are three function keys with 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 after the completion of each logical operation on its contents.

3.2.3 The Menu Bar

The menu bar provides a number of additional functions for the graph window. We briefly describe each menu item in turn:

Interpretation The *Interpretation* option of the menu bundles a number of functions that concern the overall interpretation window. You may start a new interpretation (while preserving the current one), you may clone the current interpretation to preserve it in its current state while continue work with its duplicate, you may delete the current interpretation, or you may 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 jpeg or in 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 *Graph-Edit* options 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 were already described above. Then there are five additional functions for modifying graphs on the canvas. All of them are described in Section 4, *Advanced Features*.

Verify The *Verify* option repeats the logical functions of MoMo that are also available through the buttons in the right column of the graph window.

Options *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 the user 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.

Show Node Labels may be switched on to inspect the sort symbols and attribute symbols next to each node and arc on the canvas.

Show Node Numbers switches on the displaying of the names that MoMo automatically assigns to each node on the canvas. This is necessary to understand those messages in the interaction windows where MoMo talks about specific nodes on the canvas by referring to their name. It is also necessary to switch this function on to see which nodes are in the relation tuples.

Semantics allows the user to switch between a feature structure semantics for our descriptions and a King-style semantics. The alternative semantics is a feature of MoMo that might be of interest to advanced students of the semantics of HPSG and is described in Section 4. When MoMo is started, it is standardly set to the feature structure semantics.

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

Sort Attributes and Sorts Alphabetically is a function for reordering the selection of nodes and attribute arcs in the left column of the graph window. Executing the function orders them alphabetically in top down order.

Show Abstract Feature Structure converts all concrete feature structures on the canvas into an abstract feature structure representation. The results of the conversion are displayed in a separate interaction window.

The menu bar of the interaction window for displaying abstract feature structures contains the menu item *Option*. When the window is first opened, *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 understand their representation better: For each node in a concrete feature structure, *Show paths* displays its sort label with MoMo's identification number 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 that language can be found in Section 6.

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 Section 6. If you try to convert a concrete feature structure on which the transformation algorithm is not defined, a warning message appears.

Show Nodes in Relations can help you to 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 for the feature structure to be 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.

4 Advanced Features

- copying, cutting and pasting (pieces) of graphs
- changing the sort labels of nodes
- dragging node numbers and node labels across the canvas
- special list syntax
- global, permanent and user-specific options in the note pad menu
- font selection for presentations with a projector
- King style interpretations

5 Examples

to be added

6 Specification

This section contains the complete definition of the syntax and semantics of MoMo. The logical language of MoMo is a syntactic variant—omitting *chains*—of RSRL with two possible interpretations. The default semantics of MoMo gives the expressions an interpretation in a domain of concrete feature structures. In addition, MoMo gives the user the option of choosing a semantics based on the more general interpretations that [King, 1999] suggested for the HPSG grammar of [Pollard and Sag, 1994].

insert definitions

References

[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.

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

[Richter, 2000] Richter, Frank 2000. *A Mathematical Formalism for Linguistic Theories with an Application in Head-Driven Phrase Structure Grammar*. Phil. dissertation, 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.