Chapter 11

Interactive Graphical Software for Teaching the Formal Foundations of Head-Driven Phrase Structure Grammar[1]

FRANK RICHTER, EKATERINA OVCHINNIKOVA, BEATA TRAWIŃSKI, W. DETMAR MEURERS

ABSTRACT. Here we will present a graphical software tool called *Morph Moulder (MoMo)* for teaching the formal foundations of a language with a denotation in a domain of relational typed feature structures as used in Head-Driven Phrase Structure Grammar. With MoMo, students learn the properties of totally well-typed, sort resolved relational feature structures, the use of formal languages to describe typed feature structures and the notions of constraint satisfaction and models of grammars written in a formal language. MoMo was realized and conceived within the context of a set of courses in the format of web-based training, that focuses on the concept of typed feature structures in a curriculum in grammar formalisms and parsing. The formal language of MoMo amends the constraint language of TRALE (an implementation platform for HPSG grammars based on ALE) to accommodate the expressive power of HPSG.

11.1 Motivation

Head-Driven Phrase Structure Grammar is one of the mathematically most rigorously developed grammar formalisms, currently in use in computational and in theoretical linguistics. Many educational institutions offer a variety of courses in HPSG from many different perspectives. Prominent among them are general introductory courses for modern formal syntactic theories, courses on issues of analyses of linguistic phenomena in and across particular languages, and grammar development and implementation courses. Even more specialized courses focus on the formal foundations in the logic of typed feature structures or more general logical frameworks such as RSRL (Richter, 2000). The nature of these courses and the way in which they refer to the main concepts underlying HPSG can easily differ so much as to make it very difficult for students to see the common ground underlying grammar implementation on a particular implementation platform, linguistic analyses in purely theoretical HPSG grammars, and the design of various parsing algorithms based on different approaches to processing HPSG-style grammars.

[1]We are grateful to Carmella Payne for help with the challenges of English.

Proceedings of Formal Grammar 2002
G. Jäger, P. Monachesi, G. Penn & S. Wintner (editors).

The problem increases upon considering the diverse backgrounds of the students themselves, whose backgrounds may be a range of the areas of computer science, knowledge representation, artificial intelligence and linguistics, in any combination of these subjects.

The *Morph Moulder* (MoMo), discussed in this paper, is an educational software tool for the graphical exploration of the formal foundations of typed feature logic as used in the formalization of grammars written in the HPSG framework. A key idea behind the system of courses built around MoMo is to use the theme of feature structures as the unifying focal point of a set of compatible and freely combinable course modules on topics as diverse as syntactic theory, parsing, mathematical foundations of linguistic theory and feature logics. The purpose of MoMo in this setting is to provide the necessary introduction and common ground for a potentially diverse student audience. To make the abstract mathematical nature of the topic more accessible, MoMo projects the formality of its subject, the formal foundations of constraint languages over typed feature structures, onto a graphical level at which it can be grasped more intuitively by novices. Working with the graphical interface, students gain a firm intuitive understanding of the basic concepts before they are confronted with abstract mathematical definitions. On the graphical level, they can rely on prior world knowledge in drawing connected, well-formed structures. Moreover, an interactive graphical interface can be much more easily adapted to individual learning paths through the course material. It is thus possible for the individual learners to focus on those aspects of the material that are new to them while skipping familiar ones. We expect that prior knowledge in the area of logical languages will differ rather drastically between computer scientists and linguists, for example.

After acquiring a thorough understanding on an intuitive level of what HPSG grammars are and how they are interpreted, students will find it much easier to grasp the mathematical definitions of the underlying concepts, and we can then directly introduce the new jargon of the intuitively well-understood ideas. The mathematical definitions make previously implicit knowledge explicit. The firm grounding of the course in the foundations of HPSG in a formal language interpreted over a domain of totally well-typed, sort resolved relational feature structures will give us a common point of reference for all later specializations in our system of courses in grammar formalisms and parsing. We thus achieve a comprehensive and still coherent set of teaching material that covers various topics in HPSG, all grouped around the focal point of feature structures. It includes mathematical foundations of linguistics, grammar development, grammar implementation and design, and constraint-based parsing.

MoMo is being developed as a module in a web-based course in *Grammar Formalisms and Parsing*, funded by the German Federal Ministry for Research Technology (BMBF) as part of the consortium *Media-intensive teaching modules in the computational linguistics curriculum (Medienintensive Lehrmodule in der*

Computerlinguistik-Ausbildung, MiLCA).[2]

11.2 The Morph Moulder

The MoMo tool allows the user to explore the relationship between the two levels of the formal architecture of HPSG: the descriptions and the elements described. To this end, the user works with a graphical interface on a whiteboard. Labeled graphs representing totally well-typed and sort resolved relational feature structures can be constructed on the whiteboard from their basic components, nodes and arcs. The nodes are depicted as colored balls, which are assigned sorts, and the arcs are depicted as arrows that may be labeled by feature names. Once a feature structure has been constructed, the user may examine its logical properties. The three main functions of the MoMo tool allow one to check (1) whether a feature structure complies with a given signature, (2) whether a well-formed feature structure satisfies a description or a set of descriptions, and (3) whether a well-formed feature structure is a model of a description or a set of descriptions. The functions of MoMo thus lead the user from understanding the well-formedness of feature structures with respect to a signature to an understanding of feature structures in their role as a logical model of a theory. If a student has chosen course modules that include a focus on formal foundations of feature logics or feature logics based linguistic theory, the first introduction to the subject by MoMo can easily be followed up by a course module with rigorous mathematical definitions.

11.2.1 Signatures

In constraint-based frameworks like HPSG, the user declares the primitives of the empirical domain in terms of a sort hierarchy with appropriate attributes and attribute values.[3] Consider a signature that licenses lists of various animals, which may then be classified according to certain properties. First of all, the signature needs to comprise a sort hierarchy and feature appropriateness conditions for lists. Let sort *list* be an immediate supersort of the sorts *non-empty-list* and *empty-list* in the sort hierarchy (henceforth abbreviated as *nelist* and *elist*). Let the appropriateness conditions declare the attributes HEAD and TAIL appropriate for (objects of) sort *nelist*, the values of TAIL at *nelist* be of sort *list*, and the values of HEAD at sort *nelist* be of sort *animal* (for lists of animals). Finally no attributes are appropriate for the sort *elist*. A typical choice for the interpretation—and the one adopted

[2]See http://milca.sfs.nphil.uni-tuebingen.de/ for the main MiLCA pages and http://milca.sfs.uni-tuebingen.de/A4/index.html for the pages of the subproject *Grammar Formalisms and Parsing*. MoMo is implemented in Java by Ekaterina Ovchinnikova.

[3]Following most of the linguistic HPSG literature in the tradition of Pollard and Sag (1994), we will talk of *sort* hierarchies instead of *type* hierarchies, which is the usual term for the same concept in computer science. Pollard and Sag use the term *type* for a different concept in their book, namely for distinguishing utterance *tokens* from utterance *types*. This is a good example of the many gaps between the cultures of linguistics and computer science that our system of courses strives to bridge.

here—of that kind of signature in constraint-based formalisms is the collection of totally well-typed and sort resolved feature structures. All nodes of totally well-typed and sort resolved feature structures are of a maximally specific sort (sorts with no subsorts); and they have outgoing arcs for all and only those features that are appropriate to their sort, with the feature values again obeying appropriateness. Our signature for lists thus declares an ontology of feature structures with nodes of sort *nelist* or *elist* (but never of sort *list*), where the former must bear the outgoing arcs HEAD and TAIL, and the latter have no outgoing arcs. They signal the end of the list. The HEAD values of non-empty lists must be in the denotation of the sort *animal*.

```
type_hierarchy
bot
  list
     nelist head:animal tail:list
     elist
  animal legs:number color:color
     bird legs:two
       parrot
       woodpecker
       canary
     pet legs:four
       cat
       dog
  number
     two
     four
  color
     green
     yellow
     brown
relations
totheright/3
member/2
.
```

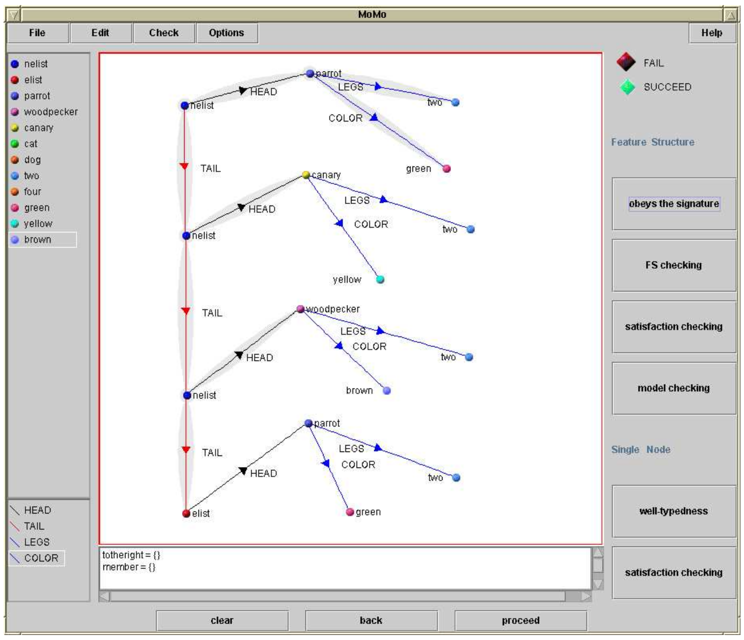
Figure 11.1: An HPSG signature in MoMo notation

Figure 11.1 shows a complete signature from a course using the MoMo tool that comprises the declarations for lists we have just discussed. A signature in MoMo starts with the declaration *type_hierarchy* and a top sort which is canonically called *bot*.[4] The subsort relationship is indicated by indentation: *list*, *animal*, *number* and *color* are the immediate subsorts of *bot*. Appropriate attributes and attribute values are listed in the line behind each sort. In accordance with the HPSG formalism,

[4]The convention for naming the top sort of linguistics bottom once more derives from computer science. In linguistics, the convention of calling it *top* derives from the metaphor of thinking about it as the most general sort that denotes everything. Computer scientists tend to think of it as the least informative sort and thus call it *bottom*.

attributes and attribute values are inherited by subsorts. Since the attribute COLOR is appropriate for *animal*, it is appropriate for *bird* and its subsorts and *pet* and its subsorts as well, and its value at these sorts is at least as specific as (the sort) *color*. The sort hierarchy proper is followed by the keyword *relations* in a new line, which indicates the beginning of the declaration of relation symbols. All relation symbols that the learner wants to use in descriptions are declared, followed by a slash and the intended arity of the relations. The present signature declares the relations `member` with arity two (which we want to use to express list membership) and `totheright` (which we want to use to express that element *x* occurs to the right of element *y* on list *z*).

Figure 11.2 illustrates how the MoMo tool can be used to study the relationship between signatures and the feature structures they license by letting the user construct feature structures and interactively explore whether particular feature structures are well-formed according to the signature. To the left of the whiteboard there are two clickable graphics consoles of possible nodes and arcs from which the user may choose to draw feature structures. The consoles offer nodes of all maximally specific sorts and arcs of all attributes that are declared in the signature. In the present example of a simple signature declaring lists of animals, *parrot*, *woodpecker*, *canary*, *cat* and *dog* are the maximally specific subsorts of *animal*. Animals have one of three possible colors and two or four legs, depending on whether the animal is a *bird* or a *pet*.



```
totheright = {}
member = {}
```

Figure 11.2: Well-typedness of feature structures in MoMo

On the whiteboard each color of edge represents a different attribute, and each color of node represents a different sort. The displaying of attribute and sort labels of arcs and nodes can be optionally switched on and off. The grayed outlines on edges and nodes indicate that all of the respective edges and nodes in this particular example are licensed by the signature that was provided. The HEAD arc originating at the node of sort *elist*, however, violates the appropriateness conditions of the signature. The feature structure depicted here, therefore, is not well-formed. The signature check thus fails on the given feature structure, as indicated by the red light in the upper function console to the right of the whiteboard.[5]

11.2.2 Feature Structure Checking

Configurations of objects on the whiteboard can violate other well-formedness conditions on totally well-typed and sort resolved feature structures that are independent of their well-formedness with respect to a given signature. The button *FS checking* allows to check whether the configuration of nodes and arcs on the whiteboard conforms to these additional requirements. A well-formed feature structure must at least have exactly one root node (graphically distinguished by MoMo by a red circle around the node); and from the root node all other nodes must be reachable by traversing the structure along the arcs in the direction of their arrows. Well-formedness of the depicted feature structure is a precondition for the evaluation of the truth of descriptions relative to feature structures discussed in the next section. The configuration of objects of Figure 11.2 illustrating well-formedness with respect to a given signature is actually not a well-formed feature structure, even if we eliminate the part that violates the signature. The reason for that is that there is no root node. Feature structure checking thus fails, as shown in Figure 11.3.

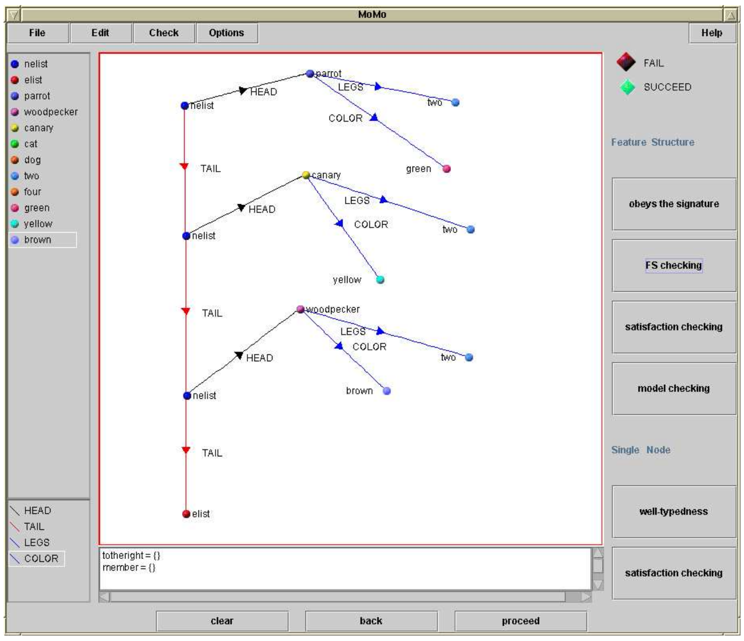
Relational feature structures may also violate well-formedness conditions in their relational extension, which is empty in the present example (but see Section 11.2.4).

11.2.3 Descriptions

Similar to signature checking, MoMo can graphically depict satisfiability and modellability of a single description or set of descriptions. To this end, the user may be asked to construct a description that a given feature structure satisfies or models; or she may be asked to construct feature structures that satisfy or model a given description (or set of descriptions). The system will give feedback on the correct or incorrect usage of the syntax of the description language as well as on the extent to which a feature structure satisfies or models descriptions, systematically guiding the user to correct solutions.

Figure 11.4 shows a successful satisfiability check of a well-formed feature structure. The feature structure is derived from the one in Figure 11.2 by removing the incorrect HEAD arc and its substructure from the node of sort *elist*. Moreover, the node on the upper left corner, from which all other nodes can be reached by

[5]Signature checking may also fail in the relational extension as will be discussed in Section 11.2.4.



```
totheright = {}
member = {}
```

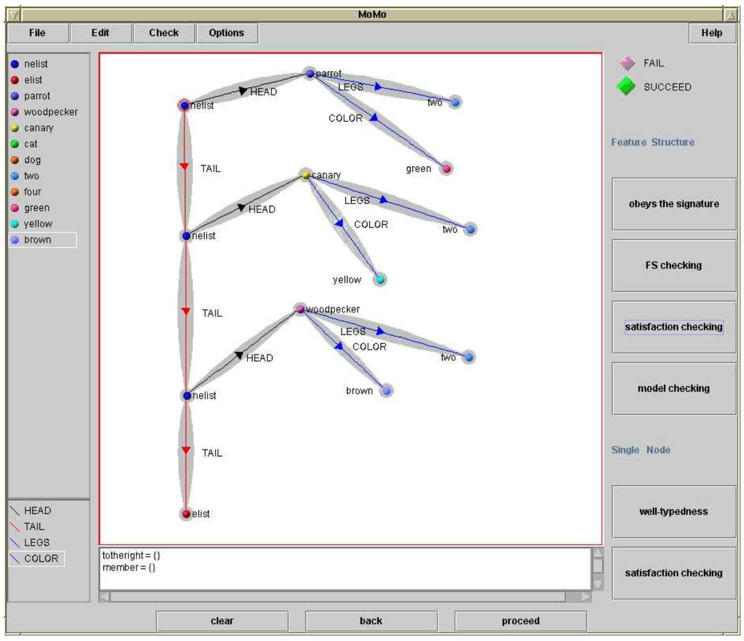
Figure 11.3: Checking the well-formedness of feature structures

(subsequently) following arcs, is now marked as the root node. Queries are asked in a special notepad, which provides additional functions such as browsing in, modifying or creating signatures, browsing in graphs provided with the system, and browsing in, modifying or creating descriptions. The query in our example asks whether the feature structure satisfies the constraint `(nelist, head: (parrot, color:green), tail:nelist)`. This constraint is true of non-empty lists with a green parrot as first element, followed by more elements.

As the feature structure satisfies the description, the green light on the function console to the right is signaling *succeed*, and the feature structure is outlined in green. If we, instead, enter the above description as a modeling query, the result is different, as can be seen in Figure 11.5.

The given feature structure does not model the description, since it has nodes that do not satisfy it. For a feature structure to model a description, each node has to satisfy it. Figure 11.5 shows that in the case of this particular feature structure, only the root node satisfies the description, whereas all others do not. For that reason, they are all marked with black circles, and the red light on the function console is signaling *fail*. As we will see below, successful modeling queries will be answered by a green light on the function console and outlining feature structures in red.

The description language of HPSG is very rich and expressive. Not only does it provide a means for notating attributes and attribute values, it also provides all standard logical connectives (with symbols for disjunction, implication and equivalence), including (classically interpreted) negation of complex expressions, existential and universal quantification over the nodes of feature structures, and rela-



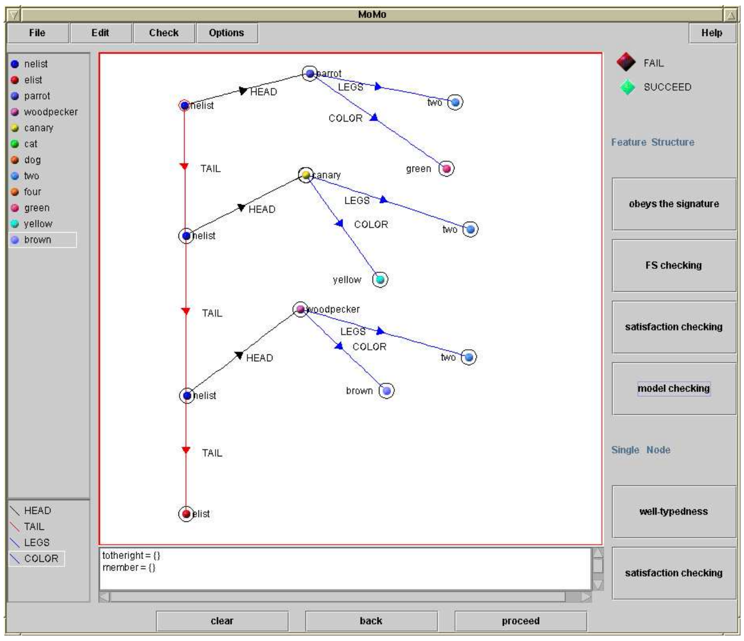
```
totheright = {}
member = {}
```

Figure 11.4: Evaluating constraint satisfaction of feature structures

tional expressions. This is more than what typical feature logics contain as their syntactic inventory, but it certainly comprises all constructs of less powerful feature logics. For the purposes of our system of courses, it is most important to note that the formal language of MoMo is a syntactic and semantic parallel to the constraint language of TRALE, an implementation platform for HPSG grammars that is an extension of ALE and is being developed by Gerald Penn at the University of Toronto as part of our web-based courses (Haji-Abdolhosseini and Penn, 2002). In our regular curriculum MoMo thus introduces the student not only to the syntax and semantics of constraint languages but also to the language that will be used for the implementation of grammars later in the course.

Similarly, the format for HPSG signatures of MoMo is almost identical to the format of signatures of TRALE. The only difference is that TRALE does not declare relations and their arity in signatures, since TRALE's relations are definite clause attachments in Prolog. The MoMo signature of Figure 11.1 becomes a TRALE signature by simply deleting the three lines concerning relations before the full stop at the very end.

With MoMo students learn the language of a formalism specifically designed to express HPSG grammars in the tradition of Pollard and Sag (1994) completely and directly. In addition, they learn the formal language of HPSG in a syntax that is optimized for direct use in computational systems, and they see immediately which syntactic constructs of the language of HPSG are not currently supported by efficient implementation platforms. The subset relationship between the constraint languages of MoMo and TRALE facilitates a comparison of the meaning



```
totheright = {}
member = {}
```

Figure 11.5: Evaluating model checking of feature structures

of grammars in theoretical HPSG and of implementations of these grammars using advanced techniques of constraint-based parsing. Finally, their common structure and properties allows for a tight network of hyperlinks in electronic course materials across the boundaries of different course modules and course topics, linking them to a common source of mathematical, implementational and linguistic indices, which explain the usage of common mathematical concepts across the different areas of application of typed feature structures.

11.2.4 Advanced Features

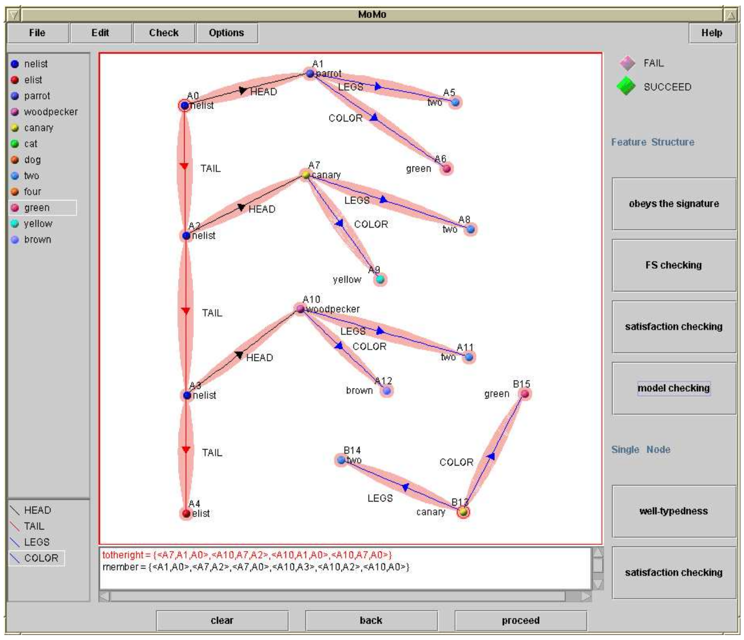
We conclude our discussion of the most important functionalities of MoMo with an example illustrating the usage of the complete syntax and semantics of the HPSG formalism in MoMo. In the expressions in (1) and (2), 'V' stands for the universal quantifier, and '^' for the existential quantifier (MoMo notation for quantifiers).

1.
```
VXVY(member(X,Y)<*>
               ^Z(Y:head:X);Y:tail:Z,member(X,Z)).
```

2.
```
VXVYVZ(totheright(X,Y,Z)<*>^W(Z:tail:W,
                            (Z:head:Y,member(X,W);
                            totheright(X,Y,W)))).
```

(1) says that for each X and for each Y in a relational feature structure, X and Y are in the `member` relation if and only if either X is the first element of Y or there is

a list Z that is the tail of Y and X is a member of Z (membership of X on a list Y). The relation `totheright` is the obliqueness relation of HPSG in models of (2). It says that for each X, Y and Z, they are in the `totheright` relation if and only if Y is the first element of list Z and X follows later on the tail of Z; or `totheright` holds between X, Y and the tail of Z. In other words, X stands to the right of Y on list Z. Figure 11.6 shows a relational feature structure that models the expressions in (1) and (2), since all the nodes have been entered in the relations as required by those two constraints. In other words, the member relation and the to-the-right relation are modeled as one would intuitively understand them. MoMo visualizes relations by first automatically assigning names composed of a letter and a number to the nodes. The relation field of the feature structure in the lower part of the whiteboard then shows the tuples of nodes that are in the relations by displaying the names of the nodes in the tuples. In Figure 11.6, the learner has correctly put $\langle A1, A0\rangle, \langle A10, A0\rangle, \langle A7, A0\rangle, \langle A7, A2\rangle, \langle A10, A2\rangle, \langle A10, A3\rangle$ into member.



```
totheright = {<A7,A1,A0>,<A10,A7,A2>,<A10,A1,A0>,<A10,A7,A0>}
member = {<A1,A0>,<A7,A2>,<A7,A0>,<A10,A3>,<A10,A2>,<A10,A0>}
```

Figure 11.6: Model checking for a feature structure with nonempty relations

Finally, Figure 11.6 reveals other important properties of MoMo's treatment of (relational) feature structures. First of all, multiple feature structures may be on the whiteboard simultaneously, and they are all checked for well-formedness and their properties relative to a given (set of) description(s). Since feature structures may be evaluated relative to a set of descriptions as well as to just one description, it can in effect be checked whether a set of feature structures models a set of constraints under a given signature. But this is nothing but checking if some set of feature structures is in the denotation of a grammar. Furthermore, MoMo recognizes which nodes on the screen are connected by arcs, and it assigns names to the nodes accordingly. Nodes with the same letter in their name belong to a single feature structure. This is relevant for the relational extension of the feature structures, since, according to the definition of relational feature structures (or *morphs*) in (Richter, 2000), only nodes of single feature structures can be in their relational extension. If the user tries to put nodes of different feature structures into the same tuple of the relational extension, feature structure checking will fail. Similarly, signature checking fails if the number of nodes in a tuple does not meet the arity of the relation according to the signature.

For lack of space, we have not discussed the complete logical functionality of MoMo. Besides the semantics of the description language in a domain of feature structures, MoMo also comprises the logically more general semantics of RSRL that goes back to Paul King's semantics of SRL (King, 1999). This gives advanced students the opportunity to go beyond the canonical semantics of feature structure models and explore alternative ways of assigning meaning to HPSG grammars like the *exhaustive models* of King or the *strong generative capacity* of Pollard (1999).

11.3 Courses on Grammar Formalisms and Parsing

MoMo is being developed as part of a set of courses on grammar formalisms and parsing in the format of web-based training. In our opinion, online courses are better suited for seminars with a highly diverse audience with varying background and different prior knowledge than courses in a traditional seminar-style setting. The course system comprising MoMo to some degree allows students to find their own learning paths through the course materials.

The course modules currently under development are (1) a module on the formal foundations of HPSG, (2) a module on grammar implementation, and (3) a module on constraint-based parsing. The module on formal foundations will be centered around MoMo, which is an implementation of large portions of RSRL (Richter et al., 1999; Richter, 2000), a comprehensive logical formalism for model-theoretic linguistic theories that was specifically designed to capture the informally presented ideas of Pollard and Sag (1994). The module on grammar implementation will include a set of increasingly complex teaching grammars in TRALE. These grammars will include the usage of a lexical rule compiler developed by Detmar Meurers, and linearization grammars that will be implemented using a linearization module of TRALE specifically designed by Gerald Penn of the University of Toronto for the purposes of these courses. Finally, we are currently porting the LinGO[6] English Resource Grammar (ERG) from LKB (on which the ERG was designed) to the TRALE system. The LinGO port is meant to provide a big exemplary grammar for the exploration by students who have taken this course module, and it is intended to show the viability of our logical approach to grammar design for large scale grammars. The module on constraint-based parsing, finally, will link discussions of algorithms to the actual annotated system source code in the

[6]http://lingo.stanford.edu/csli

TRALE system used to implement them, and mathematical definitions and discussions of linguistic constructions to the actual annotated grammar source code used to represent them in a typical implementation. The TRALE system will include a graphical user interface for interleaved visualization and interaction with trees and attribute value matrices, and an Emacs-based source level debugger for grammar development.

The course materials under development are being integrated with the web-based training platform ILIAS, which is a freely available open source software.[7] A more complete overview of the nature of our courses, the extensive system of hyperlinks employed throughout, and the educational methods that underly their implementation can be found in (Meurers et al., 2002). We expect to conclude work on all modules by the end of 2003, when the project will have to be completed and the courses will be made publicly available.

Bibliography

Haji-Abdolhosseini, M. and G. Penn (2002). *TRALE 1.0. User's Manual*. University of Toronto.

King, P. J. (1999). Towards truth in HPSG. In V. Kordoni, ed., *Tübingen Studies in HPSG*, Arbeitspapiere des SFB 340, Nr. 132, Volume 2, pp. 301–352. Universität Tübingen.

Meurers, W. D., G. Penn, and F. Richter (2002). A web-based instructional platform for constraint-based grammar formalisms and parsing. In *Proceedings of the ACL 2002 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*.

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

Pollard, C. J. (1999). Strong generative capacity in HPSG. In G. Webelhuth, J.-P. Koenig, and A. Kathol, eds., *Lexical and Constructional Aspects of Linguistic Explanation*, chapter 18, pp. 281–297. CSLI Publications.

Richter, F. (2000). A mathematical formalism for linguistic theories with an application in head-driven phrase structure grammar. Dissertation, Universität Tübingen.

Richter, F., M. Sailer, and G. Penn (1999). A Formal Interpretation of Relations and Quantification in HPSG. In G. Bouma, E. Hinrichs, G.-J. M. Kruijff, and R. T. Oehrle, eds., *Constraints and Resources in Natural Language Syntax and Semantics*, pp. 281–298. CSLI Publications.

[7] See http://www.ilias.uni-koeln.de/ios/index-e.html.