Abstract

The process of turning a “hand-written” HPSG theory into a working computational grammar requires complex considerations. Two leading platforms are available for implementing HPSG grammars: The LKB and TRALE. These platforms are based on different approaches, distinct in their underlying logics and implementation details. This paper adopts the perspective of a computational linguist whose goal is to implement an HPSG theory. It focuses on ten different dimensions, relevant to HPSG grammar implementation, and examines, compares, and evaluates the different means which the two approaches provide for implementing them. The paper concludes that the approaches occupy opposite positions on two axes: FAITHFULNESS to the “hand-written” theory and COMPUTATIONAL ACCESSIBILITY. The choice between them depends largely on the grammar writer’s preferences regarding those properties.

1 Overview

HPSG has logical and mathematical foundations which make it amenable to computational implementation. Yet it is seldom the case that this potential is in fact fulfilled, although there exist a number of platforms for implementing HPSG grammars. Thus, most descriptions and analyses of linguistic phenomena in the literature are not substantiated by a working computational grammar.

Two leading implementation platforms are available for implementing HPSG grammars. The Linguistic Knowledge Building (LKB) system (Copestake, 2002) is the primary engineering environment of the LinGo English Resource Grammar (ERG) at Stanford. The LKB is developed not particularly for implementing HPSG grammars, but rather, as a framework independent environment for typed feature structures grammar. TRALE, an extension of the Attribute Logic Engine (ALE) system, is a grammar implementation platform that was developed as part of the MiLCA project (Meurers et al., 2002), specifically for the implementation of theoretical HPSG grammars that were not explicitly written for language processing.[1] The two platforms are based on different approaches, distinct in their underlying logics and implementation details.

This paper adopts the perspective of a computational linguist whose goal is to implement an HPSG theory. It is based on the implementation of a “hand-written” grammar proposed by Melnik (2002) to account for verb initial constructions in Modern Hebrew. A representative subset of the grammar, including word order, agreement, and valence alternation phenomena, serves as a test case.

[†]This is a slightly revised version of the abstract that was submitted to the HPSG-2005 conference. The full paper is downloadable from http://cl.haifa.ac.il/projects/hebgrammar. This research was supported by the Israel Science Foundation (grant no. 136/01) and by The Caesarea Edmond Benjamin de Rothschild Foundation Institute for Interdisciplinary Applications of Computer Science. Many thanks to Shuly Wintner for his support and valuable comments and to anonymous reviewers for HPSG-2005 for their constructive comments.

[1]See http://milca.sfs.nphil.uni-tuebingen.de/A4/HomePage/English/beschr.html

The paper focuses on different dimensions, relevant to HPSG grammar implementation: type definition, grammar principles, lexical rules, exhaustive typing, definite relations, non-binary grammar rules, semantic representation, grammar evaluation, and user-interface. It examines, compares, and evaluates the different means which the two approaches provide for implementation, by referring to examples from a “hand-written” grammar fragment that was implemented in the two systems. The paper concludes that the approaches occupy diametrically opposed positions on two axes: FAITHFULNESS to the “hand-written’ theory and COMPUTATIONAL ACCESSIBILITY. The findings of this paper are valuable to linguists who are interested in implementing their grammar, as well as to those who develop implementation platforms.

2 Type Definition

Types in a typed feature-structure framework are defined by determining (i) the type’s hierarchical relation to other types, (ii) appropriateness conditions, (iii) constraints on the values of embedded features, and (iv) path equations.

TRALE separates the SIGNATURE, where the first two properties are defined, from the THEORY, in which the latter are stated. In the signature file, types are entered in a list format, where subtypes appear indented under their respective supertype(s). Features and values are introduced following the type. Constraints on embedded features and path equations are entered separately from the signature in the theory file as implicational constraints in which the type is the antecedent.

The LKB, on the other hand, takes a centralized bottom-up approach, where all the information related to a type is defined in one location, in the TYPES file. The definition of each type, then, includes a list of its immediate supertype(s) and introduced features, as well as all other type-related constraints. This approach facilitates the task of defining the type inventory and accessing this information while developing the grammar.

Although the hierarchies are defined differently in the two systems, they are both subject to the glb condition, which requires that the hierarchy be a bounded complete partial order (BCPO). Thus, when a non-BCPO hierarchy is defined, TRALE enforces the condition by producing an error message during compilation. The LKB, on the other hand, automatically creates a glb type in each case of violation and restructures the hierarchy accordingly.

On the one hand, by automatically fixing the violation, the LKB enables the grammar writer to maintain ignorance regarding a potentially confusing issue. This ignorance, however, turns into confusion once the grammar writer views the type hierarchy diagram. The automatic restructuring of the hierarchy, including the addition of generically named types, may be incomprehensible to the naive grammar writer. Moreover, the resulting hierarchy is reflected only in the display and not in the actual definitions, rendering the automatically created glb types, along with their generic names, inaccessible. A possible solution is to modify the hierarchy definition to reflect the corrected hierarchy, thus allowing the grammar writer to give the glb types more meaningful labels.

Multi-dimensional type hierarchies are widely used in the HPSG literature, yet multi-dimensionality is not a part of the formal type system itself (Penn and Hoetmer, 2003). Neither the LKB nor TRALE provide the grammar writer with a way to define partitions (or dimensions) in the hierarchy. Consequently, if partition labels are implemented as types in the hierarchy, they are not distinguished formally from other types, nor do the LKB and TRALE prevent the grammar writer from defining types that inherit from two subtypes under one pseudo-partition. Moreover, a multi-dimensional inheritance hierarchy in which partitions are defined as types does not respect the glb condition, and is therefore subjected to the systems' distinct treatments, described above. Although this omission does not prevent grammar writers from implementing their grammars, the result clearly does not reflect the source and the intention of the grammar writer.

3 Principles

Principles in HPSG are often defined as implicational constraints. Thus, for example, the Head Feature Principle (HFP), which states that the value of the HEAD feature of the headed-phrase is structure-shared with that of its head-daughter, is defined as a type constraint on the *hd-ph* type.

$$\textit{hd-ph} \rightarrow \begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{HD-DTR } \begin{bmatrix} \text{HEAD } \boxed{1} \end{bmatrix} \end{bmatrix}$$

In the LKB principles are necessarily linked to types and are stated as part of the type definition. Thus, the HFP is implemented as part of the definition of the type *hd-ph*. In TRALE, on the other hand, principles such as the HFP are stated as part of the theory, in the form of implicational constraints where the type is the antecedent, similarly to the definition above. TRALE, however, extends implicational constraints to express principles which do not target a particular type. More specifically, the antecedent of implicational constraints can be arbitrary function-free, inequation-free feature structures .

Consider, for example, the following complex-antecedent principle (Meurers, 2001).

$$\begin{bmatrix} \textit{word} \\ \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \textit{verb} \\ \text{VFORM } \textit{finite} \end{bmatrix} \\ \text{VAL} \mid \text{SUBJ} \left\langle \text{LOC} \mid \text{CAT} \mid \text{HEAD } \textit{noun} \right\rangle \end{bmatrix} \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} \text{SYNSEM} \mid \text{LOC} \mid ... \mid \text{SUBJ} \left\langle \begin{bmatrix} \text{LOC} \mid \text{CAT} \mid \text{HEAD} \mid \text{CASE } \textit{nominative} \end{bmatrix} \right\rangle \end{bmatrix}$$

The principle expresses the generalization that NP subjects of finite verbs are assigned nominative case. The complex antecedent singles out the relevant class of verbs without requiring there to be a corresponding type.

The ability to use implicational constraints with complex antecedents provides the grammar writer with additional means to express generalizations. When the given dimensions in the type hierarchy do not group together a particular set of objects to which a certain generalization applies, the grammar writer can choose not to expand the hierarchy, but rather to use a complex feature structure as an antecedent to an implicational constraint expressing the generalization. This solution can cut down on the size of the type hierarchy and its complexity.

4 Lexical Rules

The main issue that is pertinent to the implementation of lexical rules (LRs) is the "carrying over" of information from the input to the output of the rule. The descriptions of the input and output of lexical rules generally include only the features and values that are relevant for the particular rule; either those which constrain the types of objects on which to apply the rule or those which provide "information handles" (Meurers, 1994). All information which is not changed by the lexical rule is assumed to be copied over from the input to the output. An implementation platform thus has to implement the explicit as well as implicit copying of values.

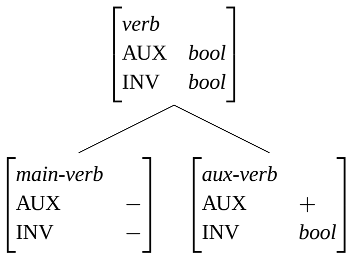
The LKB views lexical rules as unary grammar rules which relate a mother structure (the output) to its daughter (the input). Similarly to grammar rules, the description of the daughter is included in the ARGS feature of the mother. This provides a partial solution to the "carrying over" problem — the descriptions of both the mother and daughter are a part of a single feature structure. Nevertheless, the grammar writer is required to explicitly specify by structure-sharing the information that is copied over. Aside from deviating from HPSG conventions, this solution may result in a loss of generality.

TRALE provides two mechanisms for implementing lexical rules: the traditional ALE mechanism and a mechanism referred to as 'description-level lexical rules' (DLRs) which encodes the treatment proposed in Meurers and Minnen (1997). Unlike the format of the rules in the LKB, the TRALE syntax for both types of LRs is similar to the familiar 'X $\Rightarrow$ Y' notation. More importantly, from the perspective of the grammar writer, the main distinction between the two approaches is in the "carrying over" mechanism. ALE LRs, similarly to the LKB mechanism, require explicit specification of "carried over" information. The DLR version provides an automatic "carrying over" mechanism which implements the intuitions behind the "hand-written" version of lexical rules. This is a clear advantage in terms of approximating written theories and maintaining generality.

5 Exhaustive Typing and Subtype Covering

‘Exhaustive typing’ refers to a particular interpretation of the signature according to which subtypes exhaustively cover their supertypes. Consequently, if an object is of a certain non-maximal type *t* then it is also of some more specific subtype subsumed by *t*.[2]

A simple example is the HPSG analysis of subject-auxiliary inversion in English. In order to restrict the licensing of inversion to auxiliary verbs, verbs are defined as having two features: INV and AUX. Furthermore, the general type *verb* is assumed to have two subtypes: *main-verb* and *aux-verb*.



Under an exhaustive typing interpretation, objects of type *verb* which are not compatible with either *main-verb* or *aux-verb* (e.g., verbs specified with $\begin{bmatrix}\text{AUX} & -\end{bmatrix}$ and $\begin{bmatrix}\text{INV} & +\end{bmatrix}$) are rejected. This is the interpretation which TRALE employs. In the LKB such feature structures are accepted.

In addition, TRALE employs a subtype covering strategy whereby if the system recognizes that the values of a feature structure of a non-maximal type are consistent with the values of only one of its subtypes, it will promote those values to the values of the compatible subtype. This is justified only under an exhaustive typing interpretation, and is therefore not a part of the LKB system.

One advantage to TRALE’s approach is that it implements an implicit assumption in “standard” HPSG (e.g., Pollard and Sag (1994)) and is thus appropriate if the goal is to narrow the gap between “hand-written” theories and their implemented counterparts. Second, Meurers (1994) notes that “while both interpretations allow the inference that appropriateness information present on a type gets inherited to its subtypes, we can now additionally infer the appropriateness specifications on a type from the information present on its subtypes”. Moreover, in addition to increasing the expressive power, such a system facilitates syntactic detection of errors and increased efficiency in processing (Meurers, 1994).

The main reasons that are given for adopting the alternative approach, often referred to as ‘open-world reasoning’, are not theoretical, but rather, motivated by engineering considerations. This type of reasoning allows the grammar writer

[2]This interpretation is also referred to in the literature as ‘closed world’. However, as one reviewer pointed out, the terms ‘closed/open world’ have a different meaning in the study of programming languages and should therefore be avoided.

to be non-committal regarding the complete inventory of types needed to account for the language. This is particularly helpful during incremental grammar/lexicon development.

6 Definite Relations

"Hand-written" HPSG makes use of various relations which are external to the description language, many of which apply to lists and sets. One such relation is APPEND. The LKB and TRALE differ greatly in the solutions that they offer for implementing "hand-written" analyses which make use of definite relations. The LKB takes a conservative stance and adheres to the description language, while TRALE augments the description language with a programming language for implementing definite relations and incorporating them into type constraints and rules.

Programming definite relations in the TRALE environment is very similar to programming in Prolog, with the exception that first-order terms in Prolog are replaced with descriptions of feature structures. Thus, a list in this case is not a list of terms, but rather a list of descriptions of feature structures.

A thorough discussion of the benefits of adding recursive relations to the description language of implementation platforms for HPSG grammars is found in Meurers et al. (2003), which compares the treatment of unbounded dependencies and optional arguments in the ERG, implemented in the LKB, with that of TRALE. They conclude that the ability to express relational goals increases the grammar's modularity and its ability to express generalizations, and reduces the gap between "hand-written" theories and their implemented counterparts. This conclusion is echoed in the following section.

7 Non-binary Grammar Rules

Grammar rules in the HPSG literature are not restricted to binary rules. A prime example is the head-complement phrase, one of the most basic phrase structures in the grammar. In addition to being non-binary, the head-complement phrase rule is designed to account for phrases with a varying number of daughters. Implementing a rule for such a phrase type poses a number of challenges for a computational system, challenges which are handled differently by the two systems.

The assumption in the LKB is that the number of daughters associated with each rule is fixed. Thus, for grammars which are not restricted to binary branching trees the grammar writer needs to define phrase types and grammar rules for each arity. TRALE provides a special `cats>` operator to express rules with daughters lists of unspecified length. This, combined with the ability to incorporate definite recursive relations into the grammar provides the grammar writer with a way to implement non-binary grammar rules, such as the head-complement rule, in a concise and elegant manner, which closely approximates "hand-written" grammars. This, however, does require from the grammar writer the programming skills needed to be able to code using the definite logic programming language.

8 Semantic Representation

The LKB contains a module for processing Minimal Recursive Semantics (MRS) representations. The module is independent from the rest of the LKB and provides tools for manipulating MRS structures in feature structure representations (Copestake and Flickinger, 2000). TRALE provides an alternative module which is an implementation of Lexical Resource Semantics (Penn and Richter, 2004). A comparison and evaluation of the two systems will be given in the full paper.

9 Evaluating Competence and Performance

Implemented grammars can be evaluated according to two dimensions: competence and performance. The competence of a grammar refers to its coverage and accuracy, that is the ability to account for all and nothing but sentences which are assumed to be grammatical. Performance relates to the resources — mainly processor time and memory space — that are used during processing.

Both the LKB and TRALE provide a way for defining a test suite which can be used as a benchmarking facility. A batch parse returns for each sentence in the test suite the number of parses and passive edges. In terms of performance, TRALE indicates for each sentence the CPU time in seconds that it took to process the sentence. In the LKB only a total figure for all sentences is given. More sophisticated tools for evaluating competence and performance of grammars are available in both systems through the `[incr tsdb()]` package (Oepen, 2001).

10 User-Interface Issues and Features

Aside from major design differences between the two systems, the LKB and TRALE are distinguished by other more superficial user-interface type of differences.

- The LKB provides an interactive display of the grammar's type hierarchy. The user can click on types and examine their immediate and expanded definitions. TRALE produces static images of the hierarchy.
- Both systems provide ways for displaying and inspecting feature structures and syntactic trees. TRALE's Grisu graphical interface displays feature structures in AVMs that are identical to those of "hand-written" HPSG. The LKB display is less compact and more difficult to navigate.
- Parametric macros in TRALE are used as a shorthand for descriptions that are used frequently. Macros are especially useful for defining the lexicon when it is structured to minimize lexeme-specific information.

• The LKB is a graphic-user-interface system where commands are invoked through drop-down menus. In TRALE the user interacts with the program by using commands entered at the Prolog prompt.
• The LKB uses the same syntax to define types, lexical rules, grammar rules, and words in the lexicon. In TRALE distinct formats, similar to "hand-written" HPSG, are used for each of the grammar components.
• The LKB comes with the Matrix (Bender et al., 2002), an open-source starter-kit for rapid prototyping of precision broad-coverage grammars. TRALE grammars need to be implemented from scratch, or based on existing grammars.

11 Conclusion

Generally speaking, the characterization of HPSG as an implementable grammatical theory is justified, due to the computational effort that was put into designing and developing the two implementation platforms discussed in this paper. The major gap that was identified between "hand-written" HPSG and its implemented counterpart was in the multi-dimensional inheritance mechanism, which is not incorporated into neither implementation platforms.

The LKB and TRALE can be compared and evaluated along two different axes: FAITHFULNESS and ACCESSIBILITY. Faithfulness is the extent to which the implemented grammar resembles the original "hand-written" one. Accessibility, on the other hand, is the degree of computational skills that is required from a linguist in order to implement a grammar.

In some way, the LKB can be viewed as a simplified TRALE. Thus, when implicational constraints with complex antecedents, DLR lexical rules, the `cats>` operator, definite clauses, and macros are eliminated, one can implement an LKB-like grammar in TRALE. Of course, one LKB feature that cannot be assimilated is the automatic correction of glb condition violations.

The gap between the LKB-like TRALE grammar and a grammar implemented using the entire collection of tools provided by TRALE characterizes the differences between the systems. The 'true' TRALE grammar is positioned much higher on the faithfulness axis than the LKB-like TRALE grammar. The TRALE tools needed in order to elevate the LKB-like grammar on this axis require from the linguist more computational skills. This is especially true when writing (and debugging) Prolog definite clauses to express relational constraints.

In terms of accessibility, the menu-driven user interface of the LKB is more user-friendly than TRALE's command-line interface, making the LKB more attractive to the less computationally savvy linguist. However, tipping the balance a little on the accessibility scale towards TRALE is its AVM display, which is much easier to process than the LKB's. Consequently, a computational linguist interested in implementing an HPSG theory must consider these dimensions when choosing an implementation platform.

References

Bender, Emily M., Flickinger, Daniel P. and Oepen, Stephan. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*. Stanford, CA: CSLI publications.

Copestake, Ann and Flickinger, Dan. 2000. An Open Source Grammar Development Environment and Broad-coverage English Grammar Using HPSG. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, Athens, Greece.

Melnik, Nurit. 2002. *Verb-Initial Constructions in Modern Hebrew*. Ph. D.thesis, University of California at Berkeley.

Meurers, Detmar. 1994. On Implementing an HPSG Theory – Aspects of the Logical Architecture, the Formalization, and the Implementation of Head-Driven Phrase Structure Grammars. In Erhard W. Hinrichs, Detmar Meurers and Tsuneko Nakazawa (eds.), *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation*, pages 47–155, Tübingen, Germany: Eberhard-Karls-Universität Tübingen.

Meurers, Detmar. 2001. On expressing lexical generalizations in HPSG. *Nordic Journal of Linguistics* 24(2), 161–217, special issue on ‘The Lexicon in Linguistic Theory’.

Meurers, Detmar, Kuthy, Kordula De and Metcalf, Vanessa. 2003. Modularity of grammatical constraints in HPSG-based grammar implementations. In *Proceedings of the ESSLLI ’03 workshop “Ideas and Strategies for Multilingual Grammar Development”*, Vienna, Austria.

Meurers, Detmar and Minnen, Guido. 1997. A Computational Treatment of Lexical Rules in HPSG as Covariation in Lexical Entries. *Computational Linguistics* 23(4), 543–568.

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*, pages 18 – 25, New Brunswick, NJ: The Association for Computational Linguistics.

Oepen, Stephan. 2001. [incr tsdb()] — Competence and Performance Laboratory. User Manual. Technical report, Computational Linguistics, Saarland University, Saarbrücken, Germany, in preparation.

Penn, Gerald and Hoetmer, Kenneth. 2003. In Search of Epistemic Primitives in the English Resource Grammar. In *Proceedings of the 10th International Conference on Head-Driven Phrase Structure Grammar*, East Lansing, Michigan.

Penn, Gerald and Richter, Frank. 2004. Lexical Resource Semantics: From Theory to Implementation. In Stefan Müller (ed.), *Proceedings of the HPSG-2004 Conference, Center for Computational Linguistics, Katholieke Universiteit Leuven*, pages 423–443, Stanford: CSLI Publications.

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