The design of a computer language for linguistic information

The Design of a Computer Language for Linguistic Information

Stuart M. Shieber
Artificial Intelligence Center
SRI International
and
Center for the Study of Language and Information
Stanford University

Abstract

A considerable body of accumulated knowledge about the design of languages for communicating information to computers has been derived from the subfields of programming language design and semantics. It has been the goal of the PATR group at SRI to utilize a relevant portion of this knowledge in implementing tools to facilitate communication of linguistic information to computers. The PATR-II formalism is our current computer language for encoding linguistic information. This paper, a brief overview of that formalism, attempts to explicate our design decisions in terms of a set of properties that effective computer languages should incorporate.

1. Introduction[1]

The goal of natural-language processing research can be stated quite simply: to endow computers with human language capability. The pursuit of this objective, however, has been a difficult task for at least two reasons: first, this capability is far from being a well-understood phenomenon; second, the tools for teaching computers what we do know about human language are still very primitive. The solution of these problems lies within the respective domains of linguistics and computer science.

Similar problems have arisen previously in computer science. Whenever a new computer application area emerges, there follow new modes of communication with computers that are geared towards such areas. Computer languages are a direct result of this need for effective communication with computers. A considerable body of accumulated knowledge about the design of languages for communicating information to computers has been derived from the subfields of programming language design and semantics. It has been the goal of the PATR group at SRI[2] to utilize a relevant portion of this knowledge in implementing tools to facilitate communication of linguistic information to computers.

The PATR-II formalism is our current computer language for encoding linguistic information. This paper, a brief overview of that formalism, attempts to explicate our design decisions in terms of a set of properties that effective computer languages should incorporate, namely: simplicity, power, mathematical well-foundedness, flexibility, implementability, modularity, and declarativeness. More extensive discussions of various aspects of the PATR-II formalism and systems can be found in papers by Shieber *et al.*, [83], Pereira and Shieber [84] and Karttunen [84].

The notion of designing specialized computer languages and systems to encode linguistic information is not new; PROGRAMMAR [Winograd, 72], ATNs [Woods, 70], and DIALOGIC [Grosz, *et al.*, 82] are but a few of the better-known examples. Furthermore, a trend has arisen recently in linguistics towards declarativeness in grammar formalisms—for instance, lexical-functional grammar (LFG) [Bresnan, 83], generalized phrase-structure grammar (GPSG) [Gazdar and Pullum, 82] and functional unification grammar (UG) [Kay, 83]. Finally, in computer science there has been a great deal of interest in declarative languages (e.g., logic programming and specification languages), and their supporting denotational semantics. But to our knowledge, no attempt has yet been made to combine the three approaches so as to yield a declarative computer language with clear semantics designed specifically for encoding linguistic information. Such a language, of which PATR-II is an example, would reflect a felicitous convergence of ideas from linguistics, artificial intelligence, and computer science.

2. The Critical Properties of the Language

It is not the purpose of this paper to provide a comprehensive description of the PATR-II project, or even of the formalism itself. Rather, we will discuss briefly the critical

[1] This research has been made possible in part by a gift from the Systems Development Foundation, and was also supported by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0575 with the Naval Electronic Systems Command. The views and conclusions contained in this document are those of the author and should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, or the United States government.

The author is indebted to Fernando Pereira, Barbara Grosz, and Ray Perrault for their comments on earlier drafts.

[2] This rather liquid group has included at various times: John Bear, Lauri Karttunen, Fernando Pereira, Jane Robinson, Stan Rosenschein, Susan Stucky, Mabry Tyson, Hans Uszkoreit, and the author.

properties of PATR-II to give a flavor for our approach to the design of the language. References to papers with more complete descriptions of particular aspects of the project are provided when appropriate.

2.1. Simplicity: An Introduction to the PATR-II Formalism

Building on a convergence of ideas from the linguistics and AI communities, PATR-II takes as its primitive operation an extended pattern-matching technique, ***unification***, first used in logic and theorem-proving research and lately finding its way into research in linguistics [Kay, 79; Gazdar and Pullum, 82] and knowledge representation [Reynolds, 70; Ait-Kaci, 83]. Instead of unifying logic terms, however, PATR unification operates on directed acyclic graphs (DAG).[3]

DAGs can be atomic symbols or sets of label/value pairs, where the values are themselves DAGs (either atomic or complex). Two labels can have the same value—thus the use of the term *graph* rather than *tree*. DAGs are notated either by drawing the graph structure itself, with the labels marking the arcs, or, as in this paper, by notating the sets of label/value pairs in square brackets, with the labels separated from their values by a colon; e.g., a DAG associated with the verb "knight" (as in "Uther wants to knight Arthur") would appear (in at least one of our grammars) as

```
[cat: v
 head: [aux: false
        form: nonfinite
        voice: active
        trans: [pred: knight
                arg1: <f1134>
                      []
                arg2: <f1138>
                      []]]
 syncat: [first: [cat: np
                  head: [trans: <f1134>]]
          rest: [first: [cat: np
                         head: [trans: <f1138>]]
                 rest: <f1140>
                       lambda]
          tail: <f1140>]]
```

Reentrant structure is notated by labeling the DAG with an arbitrary label (in angle brackets), then using that label for future references to the DAG.

Associated with each entry in the lexicon is a set of DAGs.[4] The root of each DAG will have an arc labeled *cat*

[3] Technically, these are rooted, directed, acyclic graphs with labeled arcs. Formal definition of these and other technical notions can be found in Appendix A of Shieber *et al.* [83]. Note that some implementations have been extended to handle cyclic graph structures as well as graph structures with disjunction and negation [Karttunen, 84].

[4] In our implementation, this association is not directly encoded—since this would yield a grossly inefficient characterization of the lexicon—but is mediated by a morphological analyzer. See Section 2.6 for further details.

whose value will be the category of the associated lexical entry. Other arcs may encode information about the syntactic features, translation, or syntactic subcategorization of the entry. But only the label *cat* has any special significance; it provides the link between context-free phrase structure rules and the DAGs, as explicated below.

PATR-II grammars consist of rules with a context-free phrase structure portion and a set of unifications on the DAGs associated with the constituents that participate in the application of the rule. The grammar rules describe how constituents can be built up to form new constituents with associated DAGs. The right side of the rule lists the *cat* values of the DAGs associated with the filial constituents; the left side, the *cat* of the parent. The associated unifications specify equivalences that must exist among the various DAGs and sub-DAGs of the parent and children. Thus, the formalism uses only one representation--DAGs—for lexical, syntactic, and semantic information, and one operation--unification--on this representation.

By way of example, we present a trivial grammar for a fragment of English with a lexicon associating words with DAGs.

```
S  →  NP VP
        <VP agr> = <NP agr>
VP →  V NP
        <VP agr> = <V agr>
Uther:
              <cat> = np
       <agr number> = singular
       <agr person> = third
Arthur:
              <cat> = np
       <agr number> = singular
       <agr person> = third
knights:
              <cat> = v
       <agr number> = singular
       <agr person> = third
```

This grammar (plus lexicon) admits the two sentences "Uther knights Arthur" and "Arthur knights Uther." The phrase structure associated with the first of these is:

$[_S\ [_{NP}\ \text{Uther}]\ [_{VP}\ [_V\ \text{knights}]\ [_{NP}\ \text{Arthur}]]]$

The VP rule requires that the *agr* feature of the DAG associated with the VP be the same as (unified with) the *agr* of the V. Thus, the VP's *agr* feature will have as its value the *same* node as the V's *agr*, and hence the same values for the *person* and *number* features. Similarly, by virtue of the unification associated with the S rule, the NP will have the same *agr* value as the VP and, consequently, the V. We have thus encoded a form of subject-verb agreement.

Note that the process of unification is *order-independent*. For instance, we would get the same effect regardless of whether the unifications at the top of the parse tree were effected before or after those at the bottom. In either case, the DAG associated with, e.g., the VP node would be

```
[cat: vp
 agr: [person: third
       number: singular]]
```

These trivial examples of grammars and lexicons offer but a glimpse of the techniques used in writing PATR-II grammars, and do not begin to employ the power of unification as a general information-passing mechanism. Examples of the use of PATR-II for encoding much more complex linguistic phenomena can be found in Shieber *et al.* [83].

2.2. Power: Two Variants

Augmented phrase-structure grammars such as PATR-II can in fact be quite powerful. The ability to encode unbounded amounts of information in the augmentations (which PATR-II obviously allows) gives this formalism the power of a Turing machine. As a linguistic theory, this much power might be considered disadvantageous; as a computer language, however, such power is clearly desirable, since the intent of the language is to enable the modeling of many kinds of linguistic analyses from a range of theories. As such, PATR-II is a tool, not a result.

Nevertheless, a good case could be made for maintaining at least the decidability of determining whether a string is admitted by a PATR-II grammar. This property can be ensured by requiring the context-free skeleton to have the property of *off-line parsability* [Pereira, 83], which was used originally in the definition of LFG to maintain the decidability of that formalism [Kaplan and Bresnan, 83]. Off-line parsability requires that the context-free "skeleton" of the grammar allows no trivial cyclic derivations of the form $A \stackrel{+}{\Rightarrow} A$.

2.3. Mathematical Well-Foundedness: A Denotational Semantics

One reason for maintaining the simplicity of the bare PATR-II formalism is to permit a clean semantics for the language. We have provided a denotational semantics for PATR-II [Pereira and Shieber, 84] based on the information systems domain theory of Dana Scott [Scott, 82]. Insofar as more complex formalisms, such as GPSG and LFG, can be modeled as appropriate notations for PATR-II grammars, PATR-II's denotational semantics constitutes a framework in which the semantics of these formalisms can also be defined, discussed, and compared. As it appears that not all the power of domain theory is needed for the semantics of PATR-II, we are currently pursuing the possibility of building a semantics based on a less powerful model.[5]

2.4. Flexibility: Modeling Linguistic Constructs

Clearly, the bare PATR-II formalism, as it was presented in Section 2.1, is sorely inadequate for any major attempt at building natural-language grammars because of its verbosity and redundancy. Efficiency of encoding was temporarily sacrificed in an attempt to keep the underlying formalism simple, general, and semantically well-founded. However, given a simple underlying formalism, we can build more efficient, specialized languages on top of it, much as MACLISP might be built on top of pure LISP. And just as MACLISP need not be implemented (and is not implemented) directly in pure LISP, specialized formalisms built conceptually on top of pure PATR-II need not be so implemented (although currently we do implement them directly through pure PATR-II). The effectiveness of this approach can be seen in the fact that at least a sizable portion of English syntax has been encoded in various experimental PATR-II grammars constructed to date. The syntactic constructs encoded include subcategorization of various complement types (*NP*s, $\bar{S}$s, etc.), active, passive, "there" insertion, extraposition, raising, and equi-NP constructions, and unbounded dependencies (such as Wh-movement and relative clauses). Other theory-dependent devices that have been modeled with PATR-II include head-feature percolation [Gazdar and Pullum, 82], and LFG-like semantic forms [Kaplan and Bresnan, 83]. Note that none of these constructs and techniques required expansion of the underlying formalism; indeed, the constructions all make use of the techniques described in this section. See Shieber *et al.* [83] for a detailed discussion of the modeling of some of these phenomena.

The devices now available for molding PATR-II to conform to a particular intended usage or linguistic theory are in their nascent stage. However, because of their great importance in making the PATR-II system a usable one, we will discuss them briefly. It is important to keep in mind that these methods should not be considered a part of the underlying formalism, but merely "syntactic sugar" to increase the system's utility and allow it to conform to a user's intentions.

2.4.1. Templates

Because so much of the information in the PATR-II grammars under actual development tends to be encoded in the lexicon, most of our research has been devoted to methods for removing redundancy in the lexicon by allowing the users themselves to define primitive constructs and operations on lexical items. Primitive constructs, such as the transitive, dyadic, or equi-NP properties of a verb, can be defined by means of *templates*, that is, DAGs that encode some linguistically isolable portion of the DAG of a lexical item. These template DAGs can then be combined to build the lexical item out of the user-defined primitives.

As a simple example, we could define (with the following syntax) the template *Verb* as

Let *Verb* be

$$\langle cat \rangle = V$$

and the template *ThirdSing* as

Let *ThirdSing* be

$$\langle agr\ number \rangle = singular$$
$$\langle agr\ person \rangle = third$$

The lexical entry for "knights" would then be

[5] But see Pereira and Shieber [84] for arguments in favor of using domain theory even if all the available power is not utilized.

knights:

Verb ThirdSing

Templates can themselves refer to other templates, enabling definition of abstract linguistic concepts hierarchically. For instance, a ***modal verb*** template may use an ***auxiliary verb*** template, which in term may be defined using the ***verb*** template above. In fact, templates are currently employed for abstracting notions of subcategorization, verb form, semantic type, and a host of other concepts.

2.4.2. Lexical Rules

More complex relationships among lexical items can be encoded by means of lexical rules. These rules, such as passive and "there" insertion, are user-definable operations on the lexical items, enabling one variant of a word to be built from the specification of another variant. A lexical rule is specified as a set of selective unifications relating an input DAG and an output DAG. Thus, unification is the primitive used in this device as well.

Lexical rules are used to encode the relationships among various lexical entries that would typically be thought of as transformations or relation-changing rules (depending on one's ideological outlook). Because lexical rules perform these operations, the lexicon need include only a prototype entry for each verb. The variant forms can be derived through lexical rules applied in accordance with the morphology actually found on the verb. (The morphological analysis in the implementations of PATR-II is performed by a program based on the system of Koskenniemi [83] and was written by Lauri Karttunen [83].)

For instance, given a PATR-II grammar in which the DAGs are used to emulate the f-structures of LFG, we might write a *passive* lexical rule as follows (following Bresnan [83]):[6]

Define *Passive* as

$$\begin{aligned} \langle out\ cat \rangle &= \langle in\ cat \rangle \\ \langle out\ form \rangle &= passprt \\ \langle out\ subj \rangle &= \langle in\ obj \rangle \\ \langle out\ obj \rangle &= \langle in\ subj \rangle \end{aligned}$$

The rule states in effect that the output DAG (the one associated with the passive verb form) marks the lexical item as being a passive verb whose object is the input DAG's subject and whose subject is the input's object. Such lexical rules have been used for encoding the active/passive dichotomy, "there" insertion, extraposition, and other so-called relation-changing rules.

2.5. Modularity and Declarativeness

The PATR-II formalism is a completely declarative formalism, as evidenced by its denotational semantics and the order-independence of its definition. Modularity is achieved through the ability to define primitive templates and lexical rules that are shared among lexical items, as well as by the declarative nature of the grammar formalism itself, removing problems of interaction of rules. Rules are guaranteed to always mean the same thing, regardless of the environment of other rules in which they are placed.

[6] The example is merely meant to be indicative of the syntax for and operation of lexical rules. We do not present this as a valid definition of *Passive* for any grammar we have written in PATR-II.

2.6. Implementability

Implementability is an empirical matter, given credence by the fact that we now have three implementations of the formalism. One desirable aspect of the simplicity and declarative nature of the formalism is that even though the three implementations differ substantially from one another, using different parsing algorithms (with both top down and bottom up properties), different implementations of unification, different methods of compiling the rules, all are able to run on exactly the same grammars yielding the identical results.

The three implementations of the PATR-II system currently in operation at SRI are as follows:

- An INTERLISP version for the DEC-2060 using a variant of the Cocke-Kasami-Younger parsing algorithm and the KIMMO morphological analyzer [Karttunen, 83], and a limited programming environment.
- A ZETALISP version for the Symbolics 3600 using a left-corner parsing algorithm and the KIMMO morphological analyzer, with an extensive programming environment (due primarily to Mabry Tyson) that includes incremental compilation, multiple window debugging facilities, tracing, and an integrated editor.
- A Prolog version (DEC-10 Prolog) running on the DEC-2060 by Fernando Pereira, designed primarily as a testbed for experimentation with efficient structure-sharing DAG unification algorithms, and incorporating an Earley-style parsing algorithm.

In addition, Lauri Karttunen and his students at the University of Texas have implemented a system based on PATR-II but with several interesting extensions, including disjunction and negation in the graph structures [Karttunen, 84]. These extensions will undoubtedly be integrated into the SRI systems and formal semantics for them are being pursued.

3. Conclusion

The PATR-II formalism was designed as a computer language for encoding linguistic information. The design was influenced by current theory and practice in computer science, and especially in the areas of programming language design and semantics. The formalism is ***simple*** (consisting of just one primitive operation, unification), ***powerful*** (although it can be constrained to be decidable), ***mathematically well-founded*** (with a complete denotational semantics), ***flexible*** (as demonstrated by its ability to model analyses in GPSG, LFG, DCG and other formalisms), ***modular*** (because of its higher-level notational devices such as templates and lexical rules), ***declarative*** (yielding order-independence of operations), and ***implementable*** (as demonstrated by three quite dissimilar implemented systems and one highly developed programming environment).

As we have emphasized herein, PATR-II seems to represent a convergence of techniques from several domains—computer science, programming language design, natural language processing and linguistics. Its positioning at the center of these trends arises, however, not from the admixture of many discrete techniques, but rather from the application of a single simple yet powerful concept to the encoding of linguistic information.

References

Ait-Kaci, H., 1983: "A new Model of Computation Based on a Calculus of Type Subsumption," Doctoral Dissertation Proposal, Dept. of Computer and Information Science, University of Pennsylvania (November).

Bresnan, Joan, 1983: *The mental representation of grammatical relations* (ed.), Cambridge: MIT Press.

Gazdar, G. and G.K. Pullum, 1982: "GPSG: A Theoretical Synopsis," Indiana University Linguistics Club, Bloomington, Indiana.

Grosz, B., N. Haas, G. Hendrix, J. Hobbs, P. Martin, R. Moore, J. Robinson and S. Rosenschein, 1982: "DIALOGIC: a core natural-language processing system," *Proceedings of the Ninth International Conference on Computational Linguistics*, Prague, Czechoslavakia (July), pp. 95-100.

Kaplan, R. and J. Bresnan, 1983: "Lexical-Functional Grammar: A Formal System for Grammatical Representation," in J. Bresnan (ed.), *The mental representation of grammatical relations* (ed.), Cambridge: MIT Press.

Karttunen, L., 1984: "Features and Values," *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford California (4-7 July, 1984).

Karttunen, L., 1983: "KIMMO: a general morphological processor," *Texas Linguistic Forum*, Volume 22 (December), pp. 161-185.

Kay, M., 1979: "Functional Grammar," in *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley, California (17-19 February).

Kay, M., 1983: "Unification Grammar," unpublished memo, Xerox Palo Alto Research Center.

Koskenniemi, K., 1983: "A Two level Model for Morphological Analysis and Synthesis," forthcoming Ph.D. dissertation, University of Helsinki, Helsinki, Finland.

Pereira, F. and D.H.D. Warren, 1983: "Parsing as Deduction," in *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics* (15-17 June), pp. 137-144.

Pereira, F. and S. Shieber, 1984: "The Semantics of Grammar Formalisms Seen as Computer Languages," *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford California (4-7 July, 1984).

Reynolds, J., 1970: "Transformational Systems and the Algebraic Structure of Atomic Formulas," in D. Michie (ed.), *Machine Intelligence*, Vol. 5, Chapter 7, Edinburgh, Scotland: Edinburgh University Press, pp. 135-151.

Scott, D., 1982: "Domains for Denotational Semantics," ICALP '82, Aarhus, Denmark (July).

Shieber, S., H. Uszkoreit, F. Pereira, J. Robinson, and M. Tyson, 1983: "The Formalism and Implementation of PATR-II," in B. Grosz and M. Stickel, *Research on Interactive Acquisition and Use of Knowledge*, SRI Final Report 1894, SRI International, Menlo Park, California (November).

Winograd, T., 1972: *Understanding Natural Language*, New York, New York: Academic Press.

Woods, W., 1970: "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, Vol. 13, No. 10 (October).