FUNCTIONAL STRUCTURES FOR PARSING DEPENDENCY CONSTRAINTS

Jäppinen, H., Lehtola, A., and Valkonen K.

SITRA Foundation*
P.O. Box 329, 00121 Helsinki, Finland
and
Helsinki University of Technology
Helsinki, Finland

Abstract

This paper outlines a high-level language FUNDPL for expressing functional structures for parsing dependency constraints. The goal of the language is to allow a grammar writer to pinn down his or her grammar with minimal commitment to control. FUNDPL interpreter has been implemented on top of a lower-level language DPL which we have earlier implemented.

---

* Current address of the authors

1. Introduction

In the theory of computation a new viewpoint is digging in: to compute is to pin down the constraints that hold in a given problem domain, and a goal for computation. It is up to an interpreter to perform search for the goal in the problem domain. The result of computation follows then indirectly from the search process.

Strongly pronounced and in wide use this vantage point becomes in Prolog. Recently fresh views of parsing as constraint systems have also surfaced, such as FUG (Kay, 1985; Karttunen and Kay, 1985), LFG (Bresnan, 1978), and PATR-II (Schieber, 1985). In these languages, a user writes only grammatical constraints and need not import control instructions. The interpreter searches for a grammatical configuration that "explains" a given input sentence without violating the constraints.

These new grammars advocate yet another, more abstract departure from procedural description. Grammars for parsing have predominantly used generative rewrite rules. The ideological underpinning of parsing has been in the past that of the emulation of generative histories of configurations. The new formalisms express grammars as functional structures.

We have defined a language DPL (Dependency Parsing Language) to meet the needs of parsing a highly inflectional, agglutinating language (Nelimarkka et al., 1984). The language enforces dependency approach which accords better than phrase structure rules with the needs of non-configurational languages. DPL language and its compiler constitutes just one component of a language-based environment we have implemented for the development of parsers (Lehtola et al., 1985).

In DPL, a grammar is comprised of functions, relations, and automata. The automata, which control the parsing process, have compelled a person who writes grammar to heed control unwanted extent. This paper describes a high-level language FUNDPL (FUNctional DPL) we have designed on top of DPL. In FUNDPL, a grammar is built out of functions, relations, and functional structures. FUNDPL is a constraint system which liberates a grammar writer from control anxieties.

2. Types

Type definitions in DPL as well as in FUNDPL list and classify linguistic properties used in a grammar description. A user has flexible tools in hand. CONSTITUENT statement defines the constituent structure, that is, what attributes terminal symbols have. The domains of names are spelled out with VALUE, FEATURE, or CATEGORY statements. VALUE is used for unary properties, FEATURE for binary features. CATEGORY assigns names in hierachies. Properties are automatically inherited in hierarchies.

Names can be associated together in SUBTREE statements. LEXICON-ENTRY statement is reserved for the definition of the lexical entry form. It accepts an arbitrary tree structure expressed in list notation.

DPL (and FUNDPL) opts for reference by value. This practice results in compact and convenient notation but requires discipline from the user, e.g., all properties must have unique names. For further details about types and reference, see Nelimarkka et al., 1984.

3. Binary Constraints

FUNDPL uses syntactic functions (and semantic relations) as binary constraints in a grammar in the following sense. In analysis two abstract levels exist (Fig. 1). On the regent level (**R-level**) are those constituents which lack dependants to fill some required

functional roles. On the dependant level (**D-level**) are those constituents which have become full phrases (marked by feature *+Phrase)* and are therefore candidates for functional roles. Syntactic functions (and semantic relations) mediate between these two levels.

The underlying abstract process view is this. A word enters the parsing process via R-level. When all dependants of the constituent (the word) have been bound (from D-level), it descends to D-level. There it remains until it itself becomes bound as a dependant. Then it vanishes from sight.

To visualize, Fig. 1 exhibits a snapshot of parsing the sentence "Nuori poika lauloi virren eilen kylän kirkossa." (A/the young boy sang a hymn yesterday in the village church.)

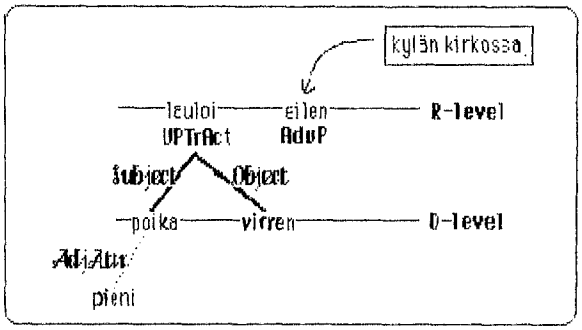

Fig. 1. A snaphot of parsing dependency constraints

Functions

FUNCTION statements define syntactic functions which are binary constraints of a grammar. Each statement declares for the function the property combinations that must simultaneously hold in a regent and a dependant (cf. Nelimarkka et al., 1984 or Lehtola et al., 1985 for details).

Function calls and function projections

Functions are called by name.s In numerous occasions verbs in particular take arguments in idiosyncratic manner. For example, the Finnish verb "rakastaa" (to love) is an ordinary transitive verb which has the one-who-is-loved in object position. But semantically closely related verb "pitää" (to like) takes the one-who-is-liked in adverbial position with elative surface case.

It is, therefore, necessary to be able to restrict the domain of a general function. Function projection is a device for that. Restrictions are written after names, a slash between. For example, to restrict the adverbial function call to elative surface cases only, one writes: **Adverbial/[Elat].** Full projections need not be explicitly shown. Instead of writing, say: **Subject/[],** one is allowed to write simply: **Subject.**

Relations

Relations give semantic interpretations to syntactically recognized functions. They are expressed in RELATION statements. Relation calls are implicit in a grammar.

4. Structural Constraints

FUNDPL uses functional schemas to express interdependent binary constraints. All syntactic functions are called via such schemas. When a word enters R-level it is associated with a schema which makes the constraints explicit for the interpreter.

Functional schemas

A schema has four parts - pattern, structural part, control part, and assignment part - and it reads as follows (required slots are underlined):

```
(F_SCHEMA:  name
            When=[properties]                    ;pattern
            .....
            Obligatory=(functions)
            Optional=(functions)                 ;structure
            Order=<conc.description>
            .....
            TryLeft=<functions>
            TryRight=<functions>                 ;control
            Down
            Up
            .....
            Assume=[properties]                  ;assignment
            Lift=function(attributes))
```

Schemas are triggered via their patterns. Pattern has a single slot **When** which indicates the required properties of a matching constituent.

Structural part may have up to three slots. One optional slot lists the obligatory functions if there are any, and another is for the optional ones. One optional slot describes what concatenation relations the expressed functions must fulfil on surface, if any.

In a concatenation description "R" stands for the regent itself. Two consequtive dots (..) signal positions of possible irrelevant intervening functions. Trailing dots may be omitted. For example, **Order**=<f1 f2 R>

requires that f1 is the first function to the left on the surface level (in a subtree dominated by R) and it is immediately followed by f2 and R in that order. **Order**=<..f1..R..f2> demands that f1 is somewhere to the left of R and f2 somewhere to the right of it.

Control part has up to four slots. Two of them are reserved for heuristic hints for the interpreter about the order it should test functions (when **Order** is not present). Control part can also raise or descend parsing between sentence levels. **Down** drops control to parse subordinate clauses, **Up** raises control to the next level. Operation **Up** is vacuous and harmless on the topmost level.

Assume slot in assignment part transfers new properties to the regent after the schema has been fully matched and bound. The other slot, **Lift**, is an optional one for the percolation of properties from a dependant via a named function link. For example, **Lift**=Subject(Case) has the effect of percolating the value of surface case to the regent from the dependant which has been bound through Subject function.

A functional schema for ordinary Finnish transitive verbs which may have unlimited number of adverbials on either side reads as follows. Notice how this single schema allows all permutations of arguments (resulting from topicalization), but it prefers SVO-ordering.

```
(F_SCHEMA: VPTrAct
           When=[Verb Act Ind +Transitive]
           Obligatory=(Subject Object)
           Optional=(Adverbial*)
           TryLeft=<Subject Object Adverbial>
           TryRight=<Object Adverbial Subject>
           Up
           Assume=[+Phrase +Sentence])
```

A simple schema suffices for relative pronouns. The following schema just marks the constituent complete and pushes control one level down to parse a subordinate relative clause. Incidently, the schema VPTrAct above parses main clauses and subordinate relative clauses as well. For the latter it raises control back to the main level.

```
(F_SCHEMA: RelPron
           When=[Relpron]
           Down
           Assume=[+Phrase +Nominal])
```

Up and **Down** commands are the only explicit pieces of control information a user has to write in FUNDPL. Implicitly he or she controls the parsing process by way of assigning properties to constituents in **Assume** slots. Heuristics is used in schemas only to speed up search.

When a schema has been fully matched and bound to its dependants through function links, it becomes a functional structure. A functional structure is an annotated tree whose branches are marked by functions.

Any number of functional structures may exist during parsing process on D-level. Process ends succesfully when all words have entered the process, R-level is empty, a single functional structure appears on D-level, and its root has properties *+Sentence, +Phrase*. Fig. 2 shows how the process in Fig. 1 terminates.

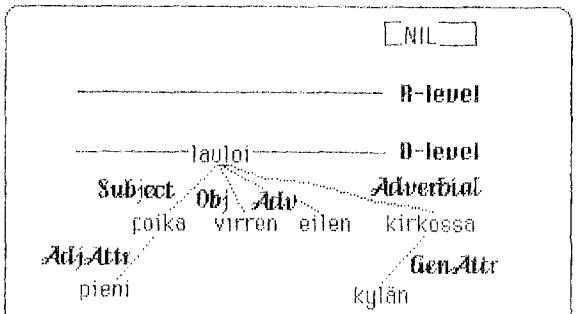

Fig. 2. A final functional structure

Conclusion

We have outlined a high level language for parsing dependency constraints. The language has been implemented on top of a lower-level language DPL which we have implemented earlier. In FUNDPL parsing process is driven by an interpreter which utilizes blackboard control strategy.

References

Bresnan, J., A realistic transformational grammar. In Halle, Bresnan, and Miller (Eds.), Linguistic Theory and Psychological Reality, MIT Press, 1978.

Jäppinen, H., and Ylilammi, M., Associative model of morphological analysis: an empirical inquiry. Computational Linguistics (to appear).

Karttunen, L., and Kay, M., Parsing in a free word order language. In Dowty, Karttunen, and Zwicky (Eds.), Natural Language Parsing. Cambridge University Press, 1985.

Kay, M., Parsing in functional unification grammar. In Dowty, Karttunen, and Zwicky (Eds.), Natural Language Parsing. Cambridge University Press, 1985.

Lehtola, A., Jäppinen, H., and Nelimarkka, E., Language-based environment for natural language parsing. 2nd European Conf. of ACL, Geneve, 1985.

Nelimarkka, E., Jäppinen, H., and Lehtola, A., Parsing an inflectional free word order language with two-way finite automata. 6th European Conf. of AI, Pisa, 1984.

Schieber, S., Using restriction to extent parsing algorithms for complex feature-based formalisms. In Proc. of the 22nd Annual Meeting of ACL, Chicago, 1985.