Some Perspectives on Induction in Discourse Representation

Gregers Koch

Here are discussed some perspectives on induction in language processing, mainly natural language analysis. In particular, we discuss some perspectives concerning the application of our system for performing a special kind of inductive inference sometimes called logico-semantic induction. This system can be seen as a meta system producing automatically logical parsers (in the form of application-specific definite clause grammars) to perform translations.

1 LOGICAL INDUCTION

1.1 *Introduction to a System Performing Logical Induction*

In this section we present an application of our system for a kind of logical induction [Koch 1988, 1991]. It deals with the automated generation of logical parsers or translators.

A logical parser is a logical formula describing a translation process into some logical notation. More precisely, we add the requirement to the meta system that the logical description is capable of producing by deduction a feasible description of the translation process. The translation takes place from a specified language fragment L, typically from a fragment of a natural language like English or French, but a programming language like Pascal may also be used. The target language of the translation is an appropriately chosen logic Log1.

As to the logical host language, the meta level translation takes place in one logical language Log2, and the object level translation takes place in a possibly different logical language Log3. Here we shall confine ourselves to discussion of the situation where both logical host languages coincide with the Horn Clause Logic HCL. Other possibilities for Log2 and Log3 can certainly also be dealt with, but here we confine ourselves to Log2 = Log3 = HCL.

As target logic Log1 our system can accomodate virtually any possible logic, and this flexibility seems to be one of the really strong features of this approach. In that particular sense we may consider it a system with high semantic generality. As an illustration we can exemplify with a variant of Montague's intensional logic ([Koch

1993]) or with a sort of situation schemata like Jens-Erik Fenstad et al. [Fenstad et al. 1987].

1.2 *The System*

Our meta program translates in seven steps for obtaining a so-called "logico-semantic induction" [Koch 1988, 1991]. In this section I shall describe the method by means of an example. For the matter of convenience, I shall select a tiny example (this should not be taken as an indication however, that only toy cases can be handled by this kind of systems).

The method presupposes that a context-free grammar has been created, including a lexicon, and it performs semantic induction for a pair of text and semantic structure. The result of the induction is a definite clause grammar (DCG) corresponding to the grammar and it will be annotated with variables. Our example uses the following little grammar

```
s -> np vp
np -> det n
vp -> v np
```

The sample text is

```
"a horse eats an apple"
```

and the intended semantic structure is

```
a(x, horse(x), a(y, apple(y), eats(x,y)))
```

representing the predicate-logic formula

$\exists x[horse(x) \wedge \exists y[apple(y) \wedge eats(x, y)]]$

First step: Reformulate the intended semantic structure into a functional tree structure where each functor has a label or a number attached.

Second step: Create a syntax tree through parsing of the sample text in accordance with the grammar.

Third step: Label the terminals of the syntax tree with the same labels or numbers as under step one in such a way that each functor in step one constitutes

an element from the category of the syntax tree carrying the corresponding label. More precisely, make a connection from a numbered functor in the result structure to the lexical category in the syntax structure to which the word (lexical or syncategorimatic) belongs.

Fourth step: Here we want to create one referential index (sometimes called a focus variable), for each noun phrase, and we shall construct a flow between the focus variable and certain other constituents. The referential indices correspond to those variables (here x and y) being part of the semantic structure.

The aim is to obtain that during parsing the resulting DCG must create a variable (a referential index) as an identifier for one of the semantic objects occuring in the semantic structure (here horse, apple etc.).

Fifth step: Here the lexical flow is constructed as a flow connecting each textual word with an element of the semantic structure. In the example a lexical flow connects the word ”horse” with the arguments of the noun category.

Sixth step: Each edge in the semantic structure of step one can be designated by the labels of the two ends of the edge. Connect the nodes with the same labels in the syntax tree through a flow following the edges of the syntax tree.

Seventh step: In this step we control that the arity is the same for each occurrence of a functor and we control the consistency of the local flow. This means that each nonterminal function symbol has the same number of arguments in every occurrence, and these arguments are connected to the surrounding nodes in the same way for every occurrence of the same syntax rule.

In our example the method will give us the following DCG:

```
s(A) --> np(B,C,A),vp(B,C).
np(D,E,F) --> det(D,G,E,F), n(D,G).
vp(H,I) --> v(H,J,K), np(J,K,I).
```

2 SOME PERSPECTIVES ON SEMANTIC THEORIES

3 Some perspectives on semantic theories

3.1 *Field Grammars*

Field grammars as invented by Paul Diderichsen ([Diderichsen 1947]) may get a sort of semantic component in the form of some so-called data-flow structures that

are also rather central concepts in logico-semantic induction. Furthermore, in an automated fashion logico-semantic induction can be used to provide Diderichsen's field grammars with a translator from natural language (in this case a fraction of Danish or some other Nordic languages) into some logical notation. Of course, this requires a decision as to precisely which logical notation to utilize.

3.2 *Montagovian Intensional Logic*

Montague's PTQ may at least partially be implemented automatically by means of logico-semantic induction ([Koch 1993]). The fact that such an implementation is created in a totally mechanical fashion could be interpreted as an indication of a kind of canonical character of PTQ, a character that is very difficult indeed to grasp when reading the forbiddingly incomprehensible paper.

3.3 *Discourse Representation Theory*

Apparently the essential parts of Hans Kamp's so-called Discourse Representation Theory ([Kamp 1981], [Kamp and Reyle 1993]) (DRT) may be implemented by means of logico-semantic induction. An exception is the precise handling of anaphora.

3.4 *Situation Semantics*

It seems to be true that at least parts of situation semantics, as described in the book by Jens-Erik Fenstad ([Fenstad et al. 1987]) may be implemented without any programming effort by means of logico-semantic induction [Koch 2000].

It would hardly be worthwhile to utilize the original text by Barwise and Perry [1993], because that book seems to describe a development of ideas rather than a single system, and so contradictions show up in several places.

3.5 *McCord's System*

McCord's chapter 5 in the 1990 book written by IBM staff members ([Walker 1990]) describes a rather comprehensive system for the analysis of simple English for database queries. It could be interesting to see precisely how big a fraction of his system might be implemented in an automated fashion and without any real programming effort by means of logico-semantic induction. At this institute, it would be possible to compare with a hand-made implementation of McCord's system [Christiansen 1993].

3.6 *Cognitive Grammar*

I share the point of view that it is possible to utilize the logico-semantic induction principles for the implementation of some of the ideas in cognitive grammars, as described by Ronald Langacker.

Inger Lytje at Aalborg University Centre has been working on a kind of handmade implementation of aspects of cognitive grammar, so the task here might deal with the reimplementation of parts of her system.

It is possible that these ideas seem to be rather alien to cognitive semantics or at least to the proponents of cognitive grammar, but nonetheless the methods discussed here seem to apply for the partial implementation of this type of systems.

4 SOME PERSPECTIVES ON TEXTUAL ANALYSIS

4.1 *Natural Language Interfaces to Data Bases and Knowledge Based Systems*

The distinction between data bases and knowledge based systems appears to be rather vague. In any case, apparently it is rather easy to provide that kind of systems with a simple natural language interface by means of logico-semantic induction. This sort of application may well be considered the prototypical way of applying logico-semantic induction.

4.2 *Machine Translation*

It may well be relevant to distinguish between two different cases. In one case, where the two languages are rather closely related (like for instance two Nordic languages or translation from English into Danish or vice versa) it may seem rather clumsy and cumbersome to use a logical notation as an intermediate representation. Whereas in the other case, where the two human languages are not very much related (like for instance when translating from Japanese into English or vice versa), it may be more appropriate to think of the use of a logical notation as an intermediate representation between the two languages.

5 SOME PERSPECTIVES ON PROGRAMMING LANGUAGES

5.1 *Compiler Construction*

The logic Log1 may be a logical representation of machine code, and the language may be a programming language. In that case we have a compiler compiler, that is, an automatic mechanism for the construction of compilers.

As opposed to possibly all other compiler compilers this one is functioning from examples of the intended compilation.

This application has an interesting perspective: when designing the original mechanism for logico-semantic induction, concepts from compilation techniques were absolutely central, in particular Don Knuth's ideas about dependency graphs [Knuth 1968]. So it was all a matter of applying concepts from the computer science discipline of compiling techniques in the context of problems from computational linguistics. The new requirements to the methods of technical solutions have probably influenced and changed and sharpened the ideas. And so it is quite interesting, after the ripening of the topic and of the methods, to reapply the methods in the original setting of compiler techniques. It is possible that the methods described here represent certain steps forward compared to the traditional methods. For instance, the focusing on the example based approach may possibly be unique.

5.2 *Program Synthesis*

We have continued to claim that the input should be a language fragment, and the output should be a logic. Here we may question that claim. After all, what is a language, and what is a logic?

It seems to be possible to apply a logico-semantic induction system on whatever problem from computer science where the task is to construct a program or a computational system to perform a kind of data transformation.

The unique aspect in this method is the indirect controlling of the automated program construction process, by specification of examples of intended data transformation. Besides a syntactic description of permitted input strings is required, and also a description of output as a logic or at least as a (formalism or) formal notational system. It is possible that professor Naur's comments on proof versus formalization ([Naur 1994]) are of some relevance here, though I tend to think that here we deal with formalisms or formal notations rather than a process of formalization.

From the point of view of the programmer the approach here seems to be a method of program synthesis with the characteristics of being rather indirect. The pro-

grammer will obtain new products by inventing still better and more appropriate samples of textual input and corresponding output.

We have to admit that we are not particularly skilled in the immediate handling of such a mechanism. But everything new takes learning.

The programmer will probably experience his own role more like a control function in relation to the logico-semantic induction system which when allowed, tends to produce one program after another. He may also experience his role as a kind of pedagogical task where he is supposed to help the computer to construct some useful programs, and he can do that by starting with some simple examples and then gradually make the examples more and more complex and refined.

By and large, this seems to me to be a good and possibly ideal sharing of cooperative work between a human being and a machine, where the man is in charge of all the ideas, the initiative, and all of the control, whereas the machine is supposed to fill in exactly what it is skilled at, efficient and complicated symbolic manipulations.

5.3 *Natural Language Programming*

This system may also be applied for a kind of natural language programming, that is, programming in natural language.

We have to admit that such terminology has been used before, or perhaps even abused in the realm of computer science. As far as I remember, the Cobol Language was promoted originally as a system that translates natural language into code. Today there would probably be widespread agreement that it was abuse of the words (or perhaps we might talk about ”massive over-selling”).

Such abuse or over-selling should not prevent us from promoting similar ideas once again, in particular in the convinction that this time the phrase is appropriate.

5.4 *Implementation Issues*

The first implementations relied heavily on the concept of data flow structures and on the recipy of logico-semantic induction mentioned in the first section of this paper.

Some later implementations elaborated on a variant of logico-semantic induction which applied unification to a higher degree. One of these implementations, worked out by students of computer science, resulted in a rather elegant, small Prolog program, no more than 5 pages in length.

An interesting aspect of the implementation of logico-semantic induction is the

problem of self-applicability. It may comprise attempts to divide the process of logico-semantic induction into minor subprocesses in such a way that the individual subprocesses may be constructed independently by application of one of the existing systems for logico-semantic induction. Such an implementation may obtain the flavor of a sort of bootstrapping process where the hand coded parts will successively be eliminated and replaced by machine generated modules. We are still in the process of working on it.

REFERENCES

J. Barwise & J. Perry, *Situations and Attitudes*, Bradford Books, Cambridge, Mass., 1983.

C.G. Brown & G. Koch (eds.), *Natural Language Understanding and Logic Programming III*, North-Holland, Amsterdam, 1991.

T.B.A. Christiansen, master thesis, 1993.

P. Diderichsen, *Elementary Danish Grammar* (in Danish), 1947.

J.-E. Fenstad et al., *Situations, Language and Logic*, D. Reidel Publishing Company, Dordrecht, Holland, 1987.

H. Kamp, A theory of truth and semantic representation, in *Formal Methods in the Study of Language*, eds. J.A.G. Groenendijk et al., Math. Centre, Amsterdam, 1981.

H. Kamp & U. Reyle, *From Discourse to Logic*, Kluwer Academic Publishers, Dordrecht, Holland, 1993.

D.E. Knuth, Semantics of Context-free Languages, *Math. Systems Theory 2(2)*, 127-145, 1968. Corrections in *Math. Systems Theory 5(1)*, 95-96, 1971.

G. Koch, Computational logico-semantic induction, in *Natural Language Understanding and Logic Programming II*, 107-134, eds. V. Dahl and P. Saint-Dizier, North-Holland, Amsterdam, 1988.

G. Koch, Linguistic data flow structures, in *Natural Language Understanding and Logic Programming III*, 293-308, eds. C.G. Brown and G. Koch, North-Holland, Amsterdam, 1991.

G. Koch, Montague's PTQ as a Case of Advanced Text Comprehension, in *Information Modelling and Knowledge Bases IV*, 377-387, eds. H. Kangassalo et al., IOS, Amsterdam, 1993.

G. Koch, A method for making computational sense of Situation Semantics, CI-

CLING 2000, Mexico, 2000.

M. McCord, Natural Language Processing in Prolog, chapter 5 in A. Walker(ed.) 1990.

P. Naur, article in BIT 34 (1994) 148-164.

R.H. Thomason, *Formal Philosophy: Selected Papers of Richard Montague* Yale University Press, London, 1974.

A. Walker(ed.), *Knowledge Systems and Prolog*, Addison-Wesley Publishing Company, 1990.

Gregers Koch is a professor of computer science and computational linguistics at the Department of Computer Science (DIKU) at Copenhagen University, Universitetsparken 1, DK 2100 Copenhagen, Denmark. He has published around 80 papers. He can be reached by email: gregers@diku.dk or Fax: (+45)35321401.

APPENDIX

```
/* grammar  */
startsymbol(t).
t ---> [s].
t ---> [s,point,s].
s ---> [np,vp].
s ---> [if,s,comma,s].
np ---> [prop].
np ---> [pron].
np ---> [d,n].
vp ---> [tv,np].

/* terminals */
terminal(pron,he,[he]).
terminal(pron,it,[it]).
terminal(prop,smith,[smith]).
terminal(prop,jones,[jones]).
terminal(prop,semantics,[semantics]).
terminal(prop,philosophy,[philosophy]).
terminal(if,if,[A,B,e(d(A),d(B))]).
terminal(point,point,[A,B,f(A,B)]).
terminal(comma,comma,[]).
terminal(tv,teaches,[A,B,teaches(A,B)]).
```

```
 terminal(tv,owns,[A,B,owns(A,B)]).
 terminal(tv,uses,[A,B,uses(A,B)]).
 terminal(n,book,[A,book(A)]).
 terminal(d,a,[A,B,C,g(d(A),B,C)]).

 /* input  */
 "jones teaches philosophy point
 if he owns a book comma he uses it".
 result(f(teaches(jones,philosophy),
   e(d(g(d(x),book(x),owns(he,x))),d(uses(he,it))))).

 /* Program synthesized */
pron(G)-->[Leksem],{terminal(pron,Leksem,[G])}.
np(G)-->pron(G).
tv(F,G,E)-->[Leksem],{terminal(tv,Leksem,[F,G,E])}.
vp(F,E)-->tv(F,G,E),np(G).
pron(F)-->[Leksem],{terminal(pron,Leksem,[F])}.
np(F)-->pron(F).
s(E,F)-->np(F),vp(F,E).
comma-->[Leksem],{terminal(comma,Leksem,[])}.
n(H,J)-->[Leksem],{terminal(n,Leksem,[H,J])}.
d(H,J,I,D)-->[Leksem],{terminal(d,Leksem,[H,J,I,D])}.
np(H,I,D)-->d(H,J,I,D),n(H,J).
tv(F,H,I)-->[Leksem],{terminal(tv,Leksem,[F,H,I])}.
vp(D,F)-->tv(F,H,I),np(H,I,D).
pron(F)-->[Leksem],{terminal(pron,Leksem,[F])}.
np(F)-->pron(F).
s(D,F)-->np(F),vp(D,F).
if(D,E,C)-->[Leksem],{terminal(if,Leksem,[D,E,C])}.
s(C)-->if(D,E,C),s(D,F),comma,s(E,F).
point(B,C,A)-->[Leksem],{terminal(point,Leksem,[B,C,A])}.
prop(L)-->[Leksem],{terminal(prop,Leksem,[L])}.
np(L)-->prop(L).
tv(K,L,B)-->[Leksem],{terminal(tv,Leksem,[K,L,B])}.
vp(K,B)-->tv(K,L,B),np(L).
prop(K)-->[Leksem],{terminal(prop,Leksem,[K])}.
np(K)-->prop(K).
s(B)-->np(K),vp(K,B).
t(A)-->s(B),point(B,C,A),s(C).
```