Basic Principles of Automated Information Extraction

Gregers Koch
Department of Computer Science, Copenhagen University
DIKU, Universitetsparken 1, DK-2100
Copenhagen, Denmark
Tel.: (+45) 35 32 14 00
Fax.: (+45) 35 32 14 01
Email: gregers@diku.dk

Abstract

Here is presented and discussed some principles for extracting the semantic or informational content of texts formulated in natural language. More precisely, as a study of computational semantics and information science we describe a method of logical translation that seems to constitute a kind of semantic analysis, but it may also be considered a kind of information extraction. We discuss the translation from Dataflow Structures partly to parser programs and partly to informational content. The methods are independent of any particular semantic theory and seem to fit nicely with a variety of available semantic theories.

1 Introduction

Here is presented and discussed some principles for extracting the semantic or informational content of texts formulated in natural language. More precisely, as a study of computational semantics and information science we describe a couple of methods of logical translation that may be considered a kind of information extraction. We discuss the translation from dataflow structures partly to parser programs or logic grammars and partly to informational content. The methods are independent of any particular semantic theory and seem to fit nicely with a variety of available semantic theories.

Information is a concept of crucial importance in any conceivable scientific endeavour. Also the modeling of information or semantic representation is becoming more and more important, not least in information systems. Databases, knowledge bases as well as knowledge management systems are continually growing. Modeling helps to understand, explain, predict, and reason on information manipulated in the systems, and to understand the role and function of components of the systems. Modeling can be made with many different purposes in mind and at different levels. It can be made by emphasising the users' conceptual understanding. It can be made on a domain level on which the application domain is described, on an algorithmic level, or on a representational level. Here the interest is focused on modeling of information on a representational level to obtain sensible semantic representations and in particular on the flow of information between the vertices of a structure describing a natural language utterance.

We are in the habit of considering the syntactic phenomena, and especially those concerning parsing, as essentially well understood and highly computational. Quite the opposite seems to be the case with semantics. We shall argue that certain central semantic phenomena (here termed logico-semantic) can be equally well understood and computational. So, in this rather limited sense we may claim that the semantic problem has been solved (meaning that there exists a computational solution). This paper contains a brief discussion and sketches a solution. A more comprehensive discussion may be found elsewhere.

The method presented will produce one single logico-semantic formula for each textual input. In case more solutions are required (and hence ambiguity is present) it is certainly possible to build together the resulting individual logic grammars.

Here we are exclusively concerned with parsing or textual analysis. Analogous considerations can be made concerning textual synthesis or generation (Kawaguchi, 1997).

We shall discuss a new method for extracting the informational content of (brief) texts formulated in natural language (NL). It makes sense to consider information extraction from NL texts to be essentially the same task as building simple kinds of information models when parsing the texts. Here we present a method that is distinguished by extreme simplicity and robustness. The simplicity makes programming of the method feasible, and so a kind of automatic program synthesis is obtained. The robustness causes wide applicability, and so the method has a high degree of generality (Koch, 1991),(Koch, 1994a),(Koch, 1994b),(Koch, 1997),(Koch, 2000a), (Koch, 2000b),(Koch, 2000c).

2 From Dataflow to Parser Programs

It is necessary to put certain restrictions on the information flow in the attributes of a syntactic tree produced by a logic grammar, in order to consider it well-formed.

Most importantly, a consistency criterion is required: multiple instances of a rule should give rise to the same information flow locally inside the instance.

Furthermore, we require the following:

- The information flow must follow the tree structure in the sense that information may flow directly from the parent's attributes to the children's attributes or vice versa, and among the attributes of the siblings.

- The starting point of the information flow has to be a terminal word in the grammar or a vertex where a new variable is created.

- The result attribute in the distinguished vertex of the syntax tree (the root or sentential vertex) is the terminal vertex of the information flow.

- There must be a path in the information flow from each starting point to the terminal vertex.

Hence, there is no general requirement (though it may well be the case) that every attribute in every vertex should be connected to the terminal vertex of the information flow.

An input text is called exhaustive if it exhausts the grammar in the sense that the syntax tree of the text contains at least one application of each syntactic production rule in the grammar (and if it contains at least one instance of each lexical category).

When we construct a parser by means of definite clause grammars (DCGs) (Covington, 1994) or other logic grammars (Abramson and Dahl, 1989),
(Deransart and Maluszynski, 1993) including the generation of a representation from a formalized logico-semantic theory, it is of course a necessary condition that the information flow in the attributes of the syntax tree corresponding to an exhaustive input text is a well-formed flow.

As an example, let us analyze the following English sentence.

"Every Swede tries to find a submarine".

Within the limits of a modestly extended first-order predicate calculus we may assign to the sentence the following three interpretations or logico-semantic representations:

$$\exists y[submarine(y) \land \forall x[swede(x) \Rightarrow try(x, find(x, y))]]$$

$$\forall x[swede(x) \Rightarrow \exists y[submarine(y) \land try(x, find(x, y))]]$$

$$\forall x[swede(x) \Rightarrow try(x, \exists y[submarine(y) \land find(x, y)]]$$

An absolutely central problem of semantics (here called the logico-semantic problem) is to assign to each input text from the appropriate linguistic universe one or several formalized semantic representations. As formalizations we shall consider here for instance logical formulae belonging to some particular logical calculus (like definite clauses or Horn clauses, first-order predicate logic, some extended first-order predicate logics, the lambda calculi, Montagovian intensional logics, situation theories, and Hans Kamp's Discourse Representation Theory) (Kamp and Reyle, 1993),(Coles, 1996),(Schank, 1982), (Devlin, 1991).

We shall discuss the problem of constructing a computational version of this assignment by displaying an analysis of the information flow in logic grammars. This leads to a rigorous method for the construction of a wide variety of logico-semantic assignments.

For instance, we may analyze the example sentence with respect to the third interpretation.

We choose a syntax in such a way that the sentence constitutes an exhaustive example. For instance, we may choose the following syntax:

```
S -> NP VP.
NP -> D N.
VP -> VPVP to VP | TV NP.
D -> a | every.
```

Notice that the determiners (D) "a" and "every" are here considered syncategorematic words (that is, they belong to the grammar rather than to the lexicon). We may make a guess as to what attributes should be available for each of the syntactic categories (S, NP, VP, VPVP, TV, N, and D). In case of mistakes, the construction of the information flow in the example will guide us into correction.

```
Res in S
Subj, Res in N, VP
Subj, Obj, Res in TV, Vpvp
Subj, Conc, Res in NP,
Subj, Restr, Scope, Res in D.
```

Here, Res designates the result attribute, Subj designates the focus or subject attribute, Obj designates the object attribute, and Prem and Conc are auxiliary attributes designating premise and conclusion, respectively. Of course, the actual choice of attribute names is immaterial. With this background we should be able to construct a well-formed information flow in the syntax tree belonging to the selected English input sentence and with respect to the intended interpretation.

Hence, by means of the constructed information flow we obtain the following result:

```
Res = Sem(every)(Subj,Prem,Conc)
    = Sem(every)(Subj,Prem,try(Subj,Obj))
    = Sem(every)(Subj,Prem,try(Subj,Sem(a)(Subj1,Prem1,Conc1)))
    = Sem(every)(Subj,Prem,
                try(Subj,Sem(a)(Subj1,Prem1,find(Subj,Subj1))))
    = Sem(every)(Subj,swede(Subj),
           try(Subj,Sem(a)(Subj1,
                          submarine(Subj1),find(Subj,Subj1)))).
```

From the information flow, we may extract the following logic grammar describing the language fragment:

```
S(Res) --> NP(Subj,Conc,Res),VP(Subj,Conc).
NP(Subj,Conc,Res) --> D(Subj,Prem,Conc,Res),N(Subj,Prem).
VP(Subj,Res) --> VPVP(Subj,Obj,Res),[to],Vp(Subj,Obj).
D(Subj,Prem,Conc,Sem(x)(Subj,Prem,Conc) --> [x]
          provided that x in {a,every}.
VP(Subj,Res) --> TV(Subj,Obj,Conc),NP(Obj,Conc,Res).
```

The corresponding lexical entries are

```
N(Subj,x(Subj)) --> [x]   provided that x in N.
TV(Subj,Obj,x(Subj,Obj)) --> [x]   provided that x in Tv.
VPVP(Subj,Obj,x(Subj,Obj)) --> [x]  provided that x in Vpvp.
```

3 From Dataflow to Informational Content

We want to argue that the rigorous method described above may be implemented in a computational fashion (that is, it is fully computable). This can be done by sketching a heuristic algorithm which generates from a single exhaustive example of an input text and its corresponding intended logico-semantic representation, a logic program that translates every text from the source language into the corresponding logico-semantic representation. The heuristic algorithm should try to analyse the logico-semantic representation of the exhaustive textual input in order to build a model of the relevant information flow in the corresponding syntax tree with attributes.

Let us illustrate the method by showing how another tiny little text will be treated. The text we choose consists of four words only:

"Peter eats an apple"

Step 2 is the choice of a syntactic description. Here we select an utterly traditional context-free description like

```
S -> NP VP.
NP -> PN | D N.
VP -> TV NP.
```

where S, NP, VP, PN, D, N, and TV designate sentence, noun phrase, verb phrase, proper name, determiner, noun, and transitive verb, respectively.

Step 3 (the analysis of information flow) is more complicated. Due to the fact that our syntax is context-free, it is possible to construct a syntactic tree for any well-formed text, so it makes sense to try to augment such a tree with further relevant information. In our little example the tree structure is

```
           NP   --   PN  --  Peter
          /
         /         TV  --  eats
        S        /
         \      /
          \    /       D  --  an
           VP       /
              \   /
               NP
                  \
                   \
                     N  --  apple
```

We shall illustrate the analysis by hinting at the resulting two nodes labelled NP and the one node labelled D.

The first NP node will be like this

```
      ^   |   ^
      |   v   |
NP (  X , 0 , 0 )
      ^   |   ^
      |   v___|
```

Here the node will be augmented with three arguments. The first argument is initialized to a new variable X that in turn will obtain a value from below (presumably the constant value 'peter'). The other two arguments will obtain the same value, as the value of the second argument is locally transported to the third argument as its value.

The second NP node will also get three arguments. The first argument is initialized to a new variable Y, and this (uninitialized) variable will be transported in the dataflow both upwards and downwards in two different directions (presumably to the daughter nodes, the D node and the N node). Hence we are getting something like

```
     ^    |   ^
     |    v   |
NP ( Y  , 0 , 0 )
     |    |   ^
    / \   |   |
   |   |  |   |
   v   v  v   |
```

The D node will obtain the following local dataflow:

```
    |   |   |   ^
    v   v   v   |
D ( 0 , 0 , 0 , 0 )
    |   |   |   ^
    v___v___v___|
```

This means that the three first arguments will get their value from above and those values will be combined to give a value for initialization of the fourth argument.

Step 4:

From the syntax structure augmented with the dataflow, we can easily synthesize a parser program, here in the form of a definite clause grammar (DCG):

```
S(  Z  )    --> NP(X, Y, Z), VP(X, Y).
NP(X, Y, Z) --> PN( X ).
NP(X, Z, W) --> D(X, Y, Z, W), N(X, Y).
VP(X, W)    --> TV(X, Y, Z), NP(Y, Z, W).
```

Step 5:
It becomes an entire parser when we supply some relevant lexical information like this:

```
PN(     peter   )           --> [Peter].
TV(X, Y, eats(X,Y))         --> [eats].
D(X, Y, Z, exists(X,Y & Z)) --> [an].
N(X, apple(X))              --> [apple].
```

Step 6 (symbolic execution):
This step amounts to keeping track of each argument when evaluating and along the way change the variable names to avoid confusion (we change the name conventions so that all variables have unique names and global scopes).

```
S ( Z )
|
|__ NP ( X, Y, Z )
|    |
|    |__ PN ( X )   &  Y = Z
|         |
|         |__ Peter   &  X = peter
|
|__ VP ( X, Y )
     |
     |__ TV ( X, Y1, Z1 )
     |    |
     |    |__ eats    &  Z1 = eats(X,Y1)
     |
     |__ NP ( Y1, Z1, Y )
          |
          |__ D ( Y1, Y2, Z1, Y )
          |   |
          |   |__ an   &  Y = exists(Y1,Y2 & Z1)
          |
          |__ N ( Y1, Y2 )
              |
              |__ apple   &   Y2 = apple(Y1)
```

Step 7:
All the possible symbolic equations are the following:

```
Y = Z
X = peter
Z1 = eats(X,Y1)
Y  = exists(Y1,Y2 & Z1)
Y2 = apple(Y1)
```

Step 8:
This system of equations is easily solved with respect to the variable Z:

```
Z = Y
  = exists(Y1,Y2 & Z1)
  = exists(Y1,Y2 & eats(X,Y1))
  = exists(Y1,apple(Y1) & eats(peter,Y1))
```

So this formula is the suggestion for the semantic representation obtained by a rigoristic and partly automated synthesis, through analysis of the information flow.

By means of some examples we can demonstrate that this method covers both simple logico-semantic representation theories in (extended) first-order logic and lambda calculatoric logico-semantic theories, and also Montagovian intensional logic and Situation Semantics (Devlin, 1991),(Koch, 1993),(Koch, 1999),(Loukanova, 1996).

References

Kamp, H. and Reyle U. *From Discourse to Logic.* Kluwer, Amsterdam, 1993.

H. Abramson and V. Dahl, *Logic Grammar*, (Springer, 1989).

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

Coles, ed., *Survey of Language Technology*, report, 1996.

M.A. Covington, *Natural Language Processing for Prolog Programmers*, (Prentice Hall, Englewood Cliffs, 1994).

P. Deransart and J. Maluszynski, *A Grammatical View of Logic Programming*, (MIT Press, 1993).

K. Devlin, *Logic and Information*, Cambridge University Press, 1991.

E. Kawaguchi et al., Toward Development of Multimedia Database System for Conversational Natural Language, 69-84, in (Kangassalo et al., 1997).

G. Koch, A method of automated semantic parser generation with an application to language technology, 103-108, in (Kawaguchi et al., 2000a).

G. Koch, A method for making computational sense of situation semantics, 308-317, A. Gelbukh, ed., *CICLing'2000, Proceedings*, Instituto Politecnico Nacional, Mexico City, 2000b.

G. Koch, Some perspectives on induction in discourse representation, 318-327, A. Gelbukh, ed., *CICLing'2000, Proceedings*, Instituto Politecnico Nacional, Mexico City, 2000c.

G. Koch, Discourse Representation Theory and induction, 401-403, H. Bunt and E. Thijsse, eds., *Proceedings of the Third International Workshop on Computational Semantics (IWCS-3)*, Holland 1999.

G. Koch, Semantic analysis of a scientific abstract using a rigoristic approach, 361-370, in (Kangassalo et al., 1997).

G. Koch, An inductive method for automated natural language parser generation, 373-380, P. Jorrand and V. Sgurev, eds., *Artificial Intelligence: Methodology, Systems, Applications*, World Scientific, 1994a.

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

R. Schank, *Dynamic Memory*, Cambridge University Press, 1982.

H. Kangassalo et al., eds. *Information Modelling and Knowledge Bases VIII*, IOS, 1997.

E. Kawaguchi et al., eds., *Information Modelling and Knowledge Bases XI*, IOS, 2000.

G. Koch, Linguistic data-flow structures, 293-308, in (Brown and Koch, 1991).

G. Koch, A discussion of semantic abstraction, 350-356, in *Information Modelling and Knowledge Bases V*, H. Jaakkola et al., eds., IOS Press, Amsterdam, 1994b.

R. Loukanova, Solving Natural Language Ambiguities in Situation Semantics, 7-16, *Bits and Bytes*, Proceedings from the 5th Danish Computational Linguistics Meeting, University of Odense, Denmark, 1996.