A Rigoristic and Automated Analysis of Texts Applied to A Scientific Abstract by Mark Sergot and Others

Gregers Koch

Department of Computer Science (DIKU)
Copenhagen University
Universitetsparken 1, DK-2100 Copenhagen, Denmark
gregers@diku.dk

Abstract

We discuss the automated construction of a translation mechanism capable of translating from any given textual input into some preferred logical notation. By constructing a syntax tree for the input text, we can augment it with semantic features through a dataflow analysis, in such a way that the parser in the shape of a (DCG) Prolog program may be extracted immediately. Over the years, our methods building on those principles, have been developed so that they seem now to be ripe for manipulating normal complicated texts meant for human communication. Here scientific abstracts seem to be very interesting. We shall discuss the analysis of a particular scientific abstract, namely that from an article by M. Sergot et al. [10]. A highly tentative comparison is made with several important alternative approaches, known from the scientific literature.

1 The Method for Analysis of Texts

1.1 Introduction

As an activity combining logic programming and language technology, we are studying certain problems in computational linguistics, in particular problems concerning the automation of dataflow analysis leading to the automatic synthesis of a special kind of dataflow structures. This automated approach may be applied for inductive purposes. The approach is sometimes called logico-semantic induction, and it constitutes an efficient kind of automated program synthesis. Here we shall discuss the automated construction of such a translation mechanism capable of translating from any given textual input into some preferred logical notation. By means of a context-free grammar and using this for constructing a syntax tree for the input text, we can augment the syntax tree with semantic features through a data flow analysis [1]. This can be done in such a way that the parser in the shape of a Definite Clause Grammar (DCG) or Prolog program may be extracted immediately.

1.2 A Small Example

Before starting with the text, we might as well analyse at least one utterly simple example in order to present and understand the approach recommended here. The tiny example is the following sentence consisting of three words only:

```
Peter kicks Pluto
```

This sentence can trivially be described by the following context-free grammar

```
S -> NP VP.
NP -> Prop.
VP -> TV NP.
```

The syntax tree corresponding to the sentence may be augmented into a dataflow structure.

```
S :                                [        ^ ]
                                            |
  NP :               [ O<--^ ,       ^ , O ]
                       |   |         |
       PROP :          | [ O ]       |
                       |             |
  VP :              [  V ,           ^ ]
                       |             |
         TV :       [  V , ^ , O ] |
                           |   |   |
         NP :            [ ^ , V , O ]
                           |
             PROP :      [ O ]
```

This structure can easily be transformed into a small logic program

```
s(Z) --> np(X,Y,Z), vp(X,Y).
np(X,Y,Y) --> prop(X).
vp(X,W) --> tv(X,Y,Z), np(Y,Z,W).
```

Supplied with suitable lexical information, for instance

```
prop(peter) --> [peter].
prop(pluto) --> [pluto].
tv(X,Y,kicks(X,Y)) --> [kicks].
```

it constitutes a executable logic program, the parser. If we execute this program with the given sentence as input, we shall get the following matchings (here we change the names of the variables to obtain unambiguity)

```
                   s( Z )
                   |
   -------------------------------
   |                            |
  np( X , Y , Z )              vp( X , Y )
   |                            |
   |     Y = Z       ----------------------
   |                 |                    |
  prop( X )         tv( X , Y1 , Z1 )    np( Y1 , Z1 , Y )
   |                 |                    |
   | X = peter       | Z1 = kicks(X,Y1)   |       Z1 = Y
   |                 |                    |
   |                 |                   prop( Y1 )
   |                 |                    |
   |                 |                    | Y1 = pluto
   |                 |                    |
 peter              kicks                pluto
```

The matchings (unifications) obtained through the execution of the program would result in the following equations

```
Y = Z
X = peter
Z1 = kicks(X,Y1)
Z1 = Y
Y1 = pluto
```

This small equational system will have the following solution

```
[peter kicks pluto]
      = Z
      = Y
      = Z1
      = kicks(X,Y1)
      = kicks(peter,pluto)
```

And that gives us a precise suggestion for a semantic representation of the input sentence. Instead, we might have obtained the logic program from the suggested input and output. This can be done automatically by means of one of our inductive meta programs [6],[7],[8]. This parser needs test runs, both forward and backward. In case all of this has been successful, we definitely have made a very detailed scrutiny of the constructive situation. And now, looking at the control aspects, let us make a list or schema of the analytical steps performed here. The steps are

```
1. example
2. syntax
3. dataflow
4. definite clause grammar
5. lexical information
6. execution
7. equations
8. solution
9. induction
10. forward test run.
11. backward test run.
```

The most important description levels in this example seem to be: 1) the text (input), 2) the formula (output or semantic representation), 3) the dataflow, 4) the parser.

1.3 Application to a Scientific Abstract

Over the years, our methods building on those principles, have been developed so that they seem now to be ripe for manipulating normal complicated texts meant for human communication. Here we consider scientific abstracts to be highly interesting, mainly because they normally are concise and very complicated. In what follows we shall discuss the analysis of a particular scientific abstract, namely that from an article by M. Sergot et al. [10]. The text simply reads

```
"The formalization of legislation and the development
 of computer systems to assist with legal problem
 solving provide a rich domain for developing and
 testing artificial-intelligence technology".
```

The resulting semantic structure became the following

```
result(empty(x,legislation(x)&
  the(y,of(formalization(y),x)&
   empty(x1,computer(systems(x1))
    &empty(w,legal(problem(solving(w)))&with(assist(x1),w))
    &the(y1,of(development(y1),x1)
       &empty(v,artificialintelligence(technology(v))
        &exists(z,for(rich(domain(z)),
              developingAndTesting(t,v))
            &provide(y&y1,z)))))))).
```

Comparison with some important alternative approaches like Kamp et al. [2], Schank et al. [9], Kawaguchi et al. [4],[11], suggests a tentative conclusion that our approach seems to be unique in its capability of handling the design of representation and in its partial solutions to problems of manual programming and automated program synthesis.

References

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

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

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

[4] E. Kawaguchi et al., Toward development of multimedia database system for conversational natural language, 69-84 in [3].

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

[6] G. Koch, Some perspectives on induction in discourse representation, 318-327, in A. Gelbukh (ed.), *Proceedings of CICLing-2000*, Mexico City, Mexico, 2000.

[7] G. Koch, A method for making computational sense of Situation Semantics, 308-317, in A. Gelbukh (ed.), *Proceedings of CICLing-2000*, Mexico City, Mexico, 2000.

[8] G. Koch, A method of automated semantic parser generation with an application to language technology, 103-108 in [5].

[9] R. C. Schank, *Dynamic Memory*, Cambridge University Press, 1982.

[10] M. Sergot et al., *Communications of the ACM*, vol. 29 no. 5 pp. 370-386.

[11] S. Yoshihara et al., An experiment on Japanese sentence generation from SD-formed semantic data, 205-221, in [5].

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 90 scientific articles. He can be reached by email: gregers@diku.dk or Fax: (+45)35321401.