A Semantic Interface for Logic Grammars

Seiki Akama and Akira Ishikawa

We introduce a semantic interface into logic grammars to give a natural link to the semantic component of natural language understanding systems. The semantic interface contains five parameters, i.e. input, semantic information, new devices, generator, and output. These parameters specify the procedure of compiling a desired semantic theory into its corresponding logic programs. The semantic interface can thus achieve a meta-level representation of required semantic theories using a meta-programming technique. We sketch how Montague semantics and discourse representation theory can be simulated in a logic-based natural language understanding system.

1. Introduction

The close relationships between logic programming and natural language understanding is testified by the early efforts in logic programming at describing the parsing of natural language sentences in logic. For one thing, the syntax of logic programming languages is very close to that of context-free grammars. For another, backward chaining in the resolution principle resembles the behavior of top-down parsers. These insights gave rise to the metamorphosis grammars of Colmerauer (1978), and the definite clause grammars (DCGs) of Pereira and Warren (1980). These formalisms belong to what are now called logic grammars (Abramson (1984, 1988), Abramson & Dahl (1989), Dahl (1981), Dahl & Saint-Dizier (1987), Pereira (1983)). Logic grammars in turn, have given a new insight into grammatical formalisms, because they can be considered as a sort of meta-grammar independent of any particular grammatical framework. The use of logic grammars in natural language analysis has also been taken to offer

a clear declarative semantics based on first-order logic. Unfortunately, this aspect of logic grammars has not been taken full advantage of. There seems to be a gap between the expressive power of Horn clauses and what should be captured by the semantic analysis of natural language. The insights of logic grammars will not be fully appreciated unless their semantic component is sufficiently enriched to be able to incorporate the current semantic theories for natural language. This is the main consideration behind our conception of what we term a semantic interface. In this paper, we introduce a semantic interface into logic grammars to give a natural link to the semantic component of natural language understanding systems. The semantic interface contains five parameters, i.e. input, semantic information, new devices, generator, and output. These parameters specify the procedure of compiling a desired semantic theory into its corresponding logic programs. The semantic interface can thus achieve meta-level representations of required semantic theories using a meta-programming technique. We sketch how Montague semantics and discourse representation theory can be simulated in a logic-based natural language understanding system.

2. Logic Grammars

The first attempt at defining logic-based grammars is seen in the *metamorphosis grammars* of Colmerauer (1978). It is based on the idea that the top-down parsing process can be identified with deduction in logic, or resolution. Thus, various grammatical formalisms can be interpreted as some sort of logic programs. A simplified version of Colmerauer's system was later outlined by Pereira and Warren (1980), which is what we now call *definite clause grammars*. Grammatical formalisms justified through logic programming have since been called *logic grammars*. Some have been devised out of particular linguistic motivations : *extraposition grammars* of Pereira (1983), *definite clause translataion grammars* of Pereira (1983), *definite clause translation grammars* of Abramson (1984, 1988) and *discontinuous grammars* of Dahl and Saint-Dizier (1987). These grammars use the expressive power of logic programming in order to formalize linguistic phenomena. However, logic is only utilized as a meta-theory of the syntactic theory required by logic grammar. Thus interaction with natural

language semantics is scarce in such formalisms. Semantic aspects, however, should not be neglected if logic grammars are to be applied to natural language understanding. As a typical example of logic grammars, we here review Abramson's definite clause translation grammars briefly. Definite clause translation grammars(DCTGs) are logic grammars based on the idea of Knuth's *attribute grammars*. The main characteristic of DCTGs is that it can specify both syntax and semantics by attaching to context-free rules semantic information called attributes. For instance, the semantic rules enable the parser to invoke procedures concerning extra tasks, e.g. plurality checking and dictionary lookup. Syntactic processing is thus cleverly controlled by appropriate semantic rules. This mechanism is skilfully supported by the inference engine of logic programming. In fact, the implementation of attribute grammars had been a fairly difficult task before we had logic programming. A definite clause translation grammar rule is of the form :

```
Left_part :: = Right_part
   〈:〉 Attributes :: - Semantics.                    (2.1)
```

where the Left_part consists of a non-terminal, and the Right_part a conjunction (possibly empty) of terminals (or non-terminals) and Prolog terms enclosed in braces {and}. In the Right-part, if it is a non-terminal, a logical variable NT may be attached to it by the ô̂perator, which will be instantiated to the subtree of the parse tree corresponding to the sub-derivation of nt. The symbol 〈:〉 separates the syntactic and semantic portions of a translation rule. Attributes ::-Semantics is read as : the Attributes of the node corresponding to this use of the rule are specified by a term or a conjunction of terms in Semantics. In summary, the syntax is described as in DGGs and the semantics is added as Horn clauses attached to each node of the parse tree. In this way, DCTs can handle syntax and semantics as separate modules. Here are some sample rules of DCTGs (for more discussion, see Abramson (1984, 1988)) :

```
Sentence ::=noun_phrase^N, verb_phrase^V,
                              {agree (N, V)}
 〈:〉
logic(P) ::-N^logic(x, P1, P),
            V^logic(X, P1)                          (2.2)
```

Here logic (P) corresponds to the logic component of the syntactic category P. The DCTG rule (2.2) shows the agreement of the noun phrase and the verb phrase together with the semantic specification in the semantic portion. The second example is a specification of the verb 'love' :

```
verb ::= [love]
〈:〉
agree (singular),
logic (transitive, X, Y, loves (X, Y)),
logic (intransitive, X, loves (X)).                    (2.3)
```

By (2.2) we can check agreement of noun phrases and verb phrases with respect to number. The third example is about the determiner 'every'. Consider the next sentence:

Every man who loves loves Mary. (2.4)

where the noun phrase is "every man who loves" and the verb phrase "loves Mary". A difficult problem with (2.4) is that the first verb 'love' is interpreted in connection with the relative clause "who loves", say P1. We here designate the verb applied to the noun as P2. Then the required DCTG rule for 'every' is:

```
determiner ::=[every]
〈:〉
plurality (singular),
logic (X, P1, P2, for-all (X, implies (P1, P2)).       (2.5)
```

With this rule, the above sentence is translated as (2.6)

```
for-all (X, implies)
    and (man (X), loves (X)),
    loves (X, Mary))                                   (2.6)
```

As we can see from the above examples, DCTGs offer better modularity than DCGs because the syntactic part and the semantic part are clearly separated within a rule. However, since semantic specification is closely tied up with the expressive power of Horn clauses, it is not easy to incorporate such semantic theories as DRT, in which the underlying semantics is clearly not first-order.

3. Semantic Interface

In order to use logic grammars in logic-based natural language understanding, we have to be able to incorporate a really powerful semantic component into the system based on a logic grammar. In other words, we need a natural link which connects a logic grammar with a semantic theory which is required for a particular purpose. Unfortunately, such a natural link has not been identifiable in the current formalisms of logic grammars. This may be because researchers in this field are content with the use of first-order logic as their semantic basis. But first-order logic is not adequate in expressing the semantics of natural language sentences (Akama (1986)). Most semantic phenomena seen in natural language are beyond first-order logic and require certain higher-order logics. Thus, unless we provide a natural link to the semantic theories of natural language, logic grammars cannot handle its semantics properly. Some people might claim that some logic grammars are already equipped with a suitable semantic component. This is not convincing, either. One reason for reviewing DCTGs has been that this grammatical formalism comes closest to providing an interface to semantics. But its aim seems very restrictive. Since DCTGs are closely related to attribute grammars, they can use semantic information while preforming syntactic processing. But the circumstances of natural language understanding are more complicated. For example, discourse analysis requires pragmatic information during syntactic processing to resolve anaphora, say by using world knowledge stored in a database. This kind of information cannot be treated in DCTGs. All logic grammars seem to suffer from the same difficulty. Even if we refine the formalisms of logic grammars for particular linguistic phenomena, the fundamental difficulty will not go away. The *semantic interface* is a meta-level representation of an interface for converting the logical forms generated by the syntax of a logic grammar into forms to be used as input to a particular semantic theory. It is independent of any logic grammar and can be connected with any semantic theory. This means that we can implement existing semantic theories within the framework of logic programming. In a sequential processing of natural language sentences, we first input a sentence to the syntactic component. Next the syntactic output is fed into the semantic component to produce the semantic form. The semantic interface can link

both components through meta-level control. A natural language sentence is analyzed by the logic grammar to give a logical representation in Horn clauses. We call it a *quasi-logical form.* But the form is not always appropriate as a semantic representation and we treat it as a syntactic form written in a logic program. Before it is sent to the semantic component, we need to know what the semantic theory is, e.g. Montague semantics and discourse representation theory. Particular devices may be required for each semantic theory, e.g., lambda abstraction, meaning postulates in the case of Montague semantics. They have to be defined meta-theoretically in the case of a non-Horn-clause theory. When the semantic theory is decided, the meta-interpreter called *generator* compiles the quasi-logical form into a corresponding logical form. The resulting logical form is the output of the semantic interface which reflects the idea of the particular semantic theory. In using the semantic interface, we need not write complex programs to accommodate a particular semantic theory we use. Instead, we just specify the meta-rules for compiling quasi-logical forms into logical forms according to the characteristics of the semantic theory. But it is not easy to directly specify the logic programs for a semantic theory because its descriptive power may be higher-order. The semantic interface makes it possible to reinterpret such a theory in logic programs by means of suitable meta-level control. Thus we can connect a logic grammar with an existing semantic theory for natural language by simulating the semantics as meta-program. Meta-programming in logic programming is originally due to Bowen and Kowalski (1982), also see Bowen (1982). The basic philosophy of meta-programming is that program can be dealt with as a *first-class object.* In other words, programs are considered to be data.

It implies that some programs can become input to other programs. This enables us to control the behavior of programs. In logic programming, this sort of idea can be implemented in various ways. For instance, the Bowen and Kowalski *amalgamation* (1982) consists in a meta-level formulation of the Horn clause theory in which the meta-theory is based on SLD resolution. The point of amalgamation is that both object-language and meta-language are interchangeable by the so-called reflection principle, guaranteeing efficiency in computing. Meta-programming has various application, e.g. dynamic change and partial computation. Our semantic inter-

face is also formalized on this idea. The semantic interface is defined by means of a meta-predicate 'semantic interface' which takes five parameters, i.e. input, semantic information, new devices, generator and output. It is described as follows :

```
semantic-interface (i, s, n, g, o)
  :- input (i),
     semantic-information (s),
     new-devices (n),
     generator (g),
     output (o).                                    (3.1)
```

The input for the first parameter of the semantic interface is in a quasi-logical form. This implies that the input is any Horn clause logic program, but the semantic specification of sentences is not so obvious. If we want to use a particular semantic theory, then devices specific to the theory may be needed for its implementation. This specification is done using the third parameter. Since such devices are not always formalizable in first-order formulas, they are introduced as meta-programs using a generator which outputs logical forms to be used in the semantics. The semantic interface will give a clarifying perspective on the arrangement of natural language understanding. We will have the following as a general schema of logic-based natural language understanding.

NL Understanding
= Syntax + Semantic Interface + Semantics (3.2)

If we assume the semantic interface as a metaprogram, the situation is more appropriately represented as follows :

NL Undertanding
= Semantic-interface (Syntax, Semantics) (3.3)

In (3.3), the semantic interface is used as the control component of a logic grammar and its semantics. At this point, let us look at some advantages involved in the use of the semantic interface in logic grammars. First, it enables us to implement existing semantic theories in logic grammars without getting bogged in the non-trivial features of these theories not expressible

in first-order theory. Our framework mimicks these features in Horn clauses at the meta-level, with the generator handling new devices for the semantic theory by compiling them into appropriate logic programs. In the next section, we will look at some concrete examples. Second, the semantic interface makes it easier to coordinate the components of a natural language understanding system. In other words, any combination of a logic grammar and a semantic framework can be achieved in this setting. From a practical point of view, this is desirable, because logic grammars are, in general, designed to focus on some particular linguistic phenomena, and so the corresponding semantic component is more or less dependent on the formalism of the logic grammar. As we have argued, the semantic interface can specify the semantics geared to the given logic grammar. In contrast, current logic grammars seem to fail to tell how to connect them with their semantic component. Third, ad hoc solutions to providing a logic grammar with its semantic component can be avoided through the use of the semantic interface. Certain special aspects of semantics may be treated as built-in mechanisms in logic programming. This, however, adds to the non-logical aspects of the systems. Rather, we should be able to delineate the semantic component in more consistent ways. Our approach aims to achieve this through a meta-level representation of the semantic theory.

4. Simulation of Current Semantic Theories

In this section, we describe how to simulate two current semantic theories by using our semantic interface. The examples given are *Montague Semantics* (MS) of Montague (1974) and *discourse representation theory* (DRT) of Kamp (1982) and Hein (1982).

4. 1. Montague Semantics

Montague semantics (MS) is one of the most successful approaches to natural language semantics based on formal logic. The most important feature of MS is that it is a *compositional semantics* for natural language in the sense that the meaning of a sentence is interpreted as a function of its parts. Montague combined a *categorial grammar* with *intensional logic* (IL) to achieve compositionality in giving the logical form to a sentence. Anoth-

er notable feature is its use of a higher-order logic as the basis of natural language analysis. Lambda reduction with its corresponding interpretation plays the most crucial role in the compositional semantics. The truth conditions of the resulting logical form are determined in an appropriate model by semantic rules. This is the basic scenario of MS. To utilize IL, we are bound to give a computatinally reasonable account of possible-worlds semantics. In fact, there have been several methods proposed to establish it, but none of such efforts seems conclusive enough to be applied to natural language understanding. Without IL, we at least have to give up formalization of certain important issues, such as modality and tense. This is, however, a topic for modal logic programming, and we do not go further into it here (cf. Akama and Kawamori (1988) for some observations). In any case, we cannot avoid formulation of intensionality in the semantics for natural language. To deal with intensionality, we must support such mechanisms as found in lambda calculus, although it is not necessary to faithfully adopt Montague's formulation of intension and extension. What we need at least is to be able to represent higher-order notions. One way of fulfilling this is the use of higher-order logic. Unfortunately, higher-order logic has some undesirable features. For one thing, it is a non-trivial task to develop a theorem-proving technique for higher-order logic. For example, unification in second-order logic is known to be undecidable. In addition, any formalization of second-order logic is in general incomplete. These defects prevent us from utilizing higher-order logic for natural language understanding. There are two possible ways to overcome these shortcomings. The first line of solution is to provide higher-order logic by restricting it to a computationally feasible fragment of higher-order logic. For instance, λ-Prolog of Nadathur (1987) uses the Horn clause counterpart of higher-order logic. An alternative solution is a meta-level simulation of higher-order logic in Horn clauses by means of meta-level control. We have chosen the latter solution because the semantic interface is based on ordinary first-order logic programming, whereas the former requires an extension of the syntax of logic programming. Higher-order concepts are of special importance for MS, since they play vital roles in the translation of natural language sentences in MS, with the help of lambda calculus. Thus, in order to incorporate MS into logic grammars, we have to formalize lambda calcu-

lus within logic programming. Some authors, such as D. H. D. Warren (1982) and D. S. Warren (1983) attempted to introduce lambda abstraction into logic programming, but their solutions seem rather ad hoc. This is partly because these approaches have no theoretical justification. In contrast, we intend to install it at the meta-level. Our strategy is to simulate the devices of lambda calculus in first-order logic. It is possible to handle β-reduction in logic programming using the following meta-predicate 'reduce' :

reduce (function, argument, result) ::- (4.1)

where 'result' is obtained by β-reduction of 'function' with 'argument', see Pereira and Shieber (1988). By means of (4.1), we can carry out Montagovian compositional semantics. for example, consider the next DCG rule :

s (S) ::-np (NP), vp (VP) (4.2)

Here, the notation is obvious. As is well-known, Montague's semantic rule for(4.2) is :

NP (VP) (4.3)

In MS (4.3) can be interpreted by lambda reduction according to an appropriate semantic rule. We write this in the following way :

s (S) ::- np (NP), vp (VP),
{reduce (NP, VP, S)} (4.4)

We here follow Montague's semantic rule based on the idea that noun phrases are to be generalized quantifiers, as opposed to the application VP (NP). Montague's translation of the determiner 'every' is represented as follows :

det ((X/P)/(X/Q)/ALL(X, P → Q)))→ [every] (4.5)

Here the symbol '/' corresponds to 'λ' in lambda calculus. Clearly, the description of lambda terms in logic programs is less elegant because of the modifications of its syntax. We here encode (4.5) in the syntax similar to that of λ-Prolog as follows :

```
det (P/Q/ALL(X/(P(X)→ Q(X)) → [every]                    (4.6)
```

This is the usual representation in MS. Now, we show the semantic interface of MS with a device for lambda abstraction :

```
semantic-interface (i, s, n, g, o)
  ::- input (i),
      semantic-information (MS),
      new-devices (lambda-operator '/'),
      generator (g),
      output (o).                                        (4.7)
```

Here the output is produced from the input in a quasi-logical form by using the desired generator. Consider the case of the above example, where the input is 'every' and the output is (4.6) for DCG. (4.6) is in a quasi-logical form which, in turn, is interpreted by the generator g :

```
generator (g)
  ::-reduce (every, N, NP),
     reduce (P/Q, P, Q)                                  (4.8)
```

As is seen in this example, the semantic interface can manipulate new devices required for a particular semantic theory by compiling them into logic programs through specifications of the generator. In this way, we can accommodate lambda calculus without revising the whole syntax of logic programming as in λ-Prolog. The method shown above in this semantic interface is, of course, not the only way to realize lambda calculus. For example, unification in logic programming is so powerful that it can treat the higher-order mechanisms in MS. And such a built-in device is very close to that used in the so-called unification-based grammars in Pereira and Shieber (1988). It would deserve special attention. We have seen that the semantic interface enables us to simulate lambda abstraction, which is essential in MS. The user of logic grammars thus need not take account of this non-trivial feature of MS in his programs. This is an advantage of introducing the semantic interface. It should be noticed that the simulation of lambda calculus cannot deal with all the characteristics of MS which are helpful to natural language semantics. For example, meaning postulates are vital to semantic interpretation in MS for selecting the most suitable model. For the

full implementation of MS, there is still a lot of to be worked out. The reader is referred to van Benthem (1986) for recent trends in formal semantics for natural language.

4. 2. Discourse Representation Theory

It is natural to consider extending the idea of MS to pragmatics. But such an extension is very difficult to establish since it involves formalizing partial information about the world. In this respect, Kamp's discourse respresentation theory (DRT) offers one of the most promising formalisms for discourse analysis. DRT includes two main devices for representing the semantics of natural language : an algorithm for translating natural language sentences into representations called DRSs, and mapping from the representations to a first-order model for determining their truth-conditions. It should be emphasized that the essense of DRT does not lie in DRSs themselves, but in their partial interpretation in terms of classical models. More formally, a DRS is a partially ordered set of DRs, each of which is a pair 〈U, Con〉, where U is a set of discourse referents and Con is a set of constraints. A DRS for a discourse D may be constructed by *DRS construction rules* which are applied in parallel to syntactic rules. Thus a DRs can be interpreted as a partial possible world in which a set of discourse referents are specified along with the relations they stand in. This implies that more information enables us to introduce new discourse referents, making the meaning of the discourse more precise. For the interpretation of discourse, DRT has another important notion called *embedding*. An embedding is a mapping of partial models to a total model, connecting the abstract representations with a total world, i.e. real world. Truth-conditions are thus definable within a first-order model. In other words, a discourse D is shown to be true in a model M given its DRS K if there is a *proper embedding* of K in M. Thus, we can obtain information about a discourse from a series of DRSs (or Heim's files) and know its truth-conditions from a proper embedding. Therefore, meaning in DRT consists in a stage of DRSs, i.e. it is a sequence of embeddings. Now, we present the semantic interface for DRT. There are perhaps two possible methods for implementing it. One is to identify a DRS with a database and run a query to verify whether it is a logical consequence of the database in view of DRT. The other is to encode Kamp's

theory faithfully in logic programs. The latter type of approach was in fact explored in Sedgbo (1988) with a logical description system SYLOG. The choice betwen these two depends on the user's design policy. But something crucial from a theoretical point of view is also involved. For example, the (declarative) semantics of logic programming is not compatible with the semantics of DRT since DRT requires *non-classical semantics*. From a practical point of view, the former is more promising. As explained above, a DR is more promising. As explained above, a DR is represented as a pair of a set of discourse referents and a set of constraints. At first glance, discourse referents may seem to correspond to variables and constraints to clauses in logic programming. It is, however, not possible to reduce them to their counterparts in first-order logic. The reason is that such a reduction cannot capture the dynamic aspect of models for DRT, which is not congenial to the static formulation of first-order models. This is because the interpretation of certain logical symbols in DRT differs from that in first-order logic. It is thus necessary to codify DRSs for semantic processing. After the above consideration, we specify DRT by means of the following semantic interface :

```
semantic-interface (i, s, n, g, o)
  ::-input (i),
     semantic-information (DRT),
     new-devices (DRS),
     generator (g),
     output (o).                                    (4.9)
```

Since nothing important hinges on the distinction between DRs and DRSs in the subsequent exposition, we just conflate the two. Our interpretation of DRS is descibed in the following way:

```
DRS (d) ::- discourse-referent (d-r),
            constraint (c), link (d, d')            (4.10)
```

where 'd-r' and 'c' denote lists of discourse referents and constraints, respectively. The link predicate relates one DRS to another DRS and it denotes an increase of information about the discourse. This is what Kamp called accessibility. The intuitive meaning of this relation is as follows :If

we have more information, it is possible to obtain DRSs closer to first-order models. A proper embedding means that the truth-conditions for DRSs (partial models) can be reinterpreted as those in first-order (total)models. It thus enables us to define truth in DRT in logic programs using the predicate 'link' :

database (p) ::-DRS (d), link (d, P) (4.11)

where P is a database. DRs are fragments of P with respect to a given discourse. With this interpretation, we incorporate the following metarule :

demo (P, A) ::-demo (DRS(d), A), link (d,p) (4.12)

The meta-rule in (4.12) connects the provability in DRT with that in logic programming concerning whether query A succeeds on database P. It is a plausible identification in that logic programming queries effect the same interpretation although they assume classical provability. Let us look at how the semantic interface is used to induce DRSs. Given a set of sentences each with its analysis tree indicating the outermost rules for the component phrases, the generator converts them into the corresponding complets DRS.

s(np(every(man), vp(owns, np(a(donkey))) (4.13)

We expect a quasi-logical form such as in (4.13). The DRS construction rules correspond to category symbols and terminal symbols with logical import such as "every", "if", "a", etc. Instead of (4.10), we use the next representation for DRSs.

drs (i, d-r, c1, c2, 1) (4.14)

where "i" is the index of the DRS, "d-r" is the list of discourse referents, "c1" is the list of unprocessed conditions, "c2" is that of processed conditions, and "1" is the link predicate. The distinction between unprocessed conditions and processed conditions is necessary for the purpose of DRS construction because, given a quasi-logical form like (4.13), the construction of a complete DRS proceeds step by step by reducing relevant category and terminal symbols.

construct (drs(D1), P1, P4)

```
::- update (drs(D1'), P1, P2),
    augment ([drs(D2), drs (D3)], P2, P3),
    construct (drs (D3), P3, P4).                    (4.15)
```

(4.15) shows the construct predicate which reduces "every". "P1" to "P4" are the different stages of the database, which is both updated and augmented by this rule. The rule is triggered by the unprocessed condition "s (np(every(A)), B)" in D1. The condition becomes a processed condition in D1' along with "implies (D2, D3)". The two new DRSs get recorded in the database. In D2, a discourse referent is introduced along with a processed condition "member (x, A)", indicating that x has the property denoted by "A". In D3, an unprocessed condition "s (x, B)" is introduced and becomes the trigger of a further application of a construction rule. Since DRT is a theory of discourse semantics/pragmatics, the input to the semantic interface is a set of quasi-logical forms corresponding to a discourse. As a result, the generator for DRT must be defined recursively.

```
generator (DRT, [ ], complete_DRSs, complete_DRSs).

generator (DRT, [q-1f | rest], DRSs1, DRSs3)
  ::- construct (drs (i, d-r, c1 ([q-1f]), c2, 1), DRSs1 DRSs2),
      generator (DRT, rest, DRSs2, DRSs3).           (4.16)
```

In (4.16), the second argument of "generator" is a set of quasi-logical forms. The DRSs for the entire discourse are to be obtained as "complet_DRSs".

To achieve (4.12) more precisely, we must explore its connections with the declarative semantics for logic programming especially to some three-valued semantics as in Fitting (1985). This is, however, beyond the confines of this paper. A detailed account of discourse anaphora is omitted here, and the reader should consult Kamp (1982). An alternative foundation of partial semantics can be outlined on the basis of a partial approach to possible-worlds semantics. This line of research was done in *data logic* of Veltman (1985) and Landman (1986). A logic programming formalization of data logic was given in Akama and Kawamori (1988). As an intriguing theory of discourse semantics, we should also mention *situation semantics* of Barwise and Perry(1983), which reflects a different viewpoint on partiality. The semantic interface can also provide a reasonable way to realize sit

uation semantics in logic programming. We hope to report on such a project elsewhere. We have shown how to interpret two of the recent theories of natural language semantics in logic programming. As you see, the semantic interfaces for these two theories are considerably different. MS needs to have higher-order mechanisms, which requires logic programming to include lambda calculus in logic programming. Thus the role played by the semantic interface is rather marginally restricted to providing a technical apparatus. On the other hand, DRT requires us to reinterpret first-order logic in a partial setting. Namely, we have to simulate the meta-theory for DRT in a slightly different way from the meta-theory for first-order logic. Without a semantic interface, implementation of DRT would be difficult although its basis is first-order logic. These facts imply that the semantic interface gives us various levels of representations of meta-theory for natural language semantics. In this sense, our proposed method would be seen as a fruitful application of meta-programming in logic programming.

5. Conclusion

This paper has introduced an approach called the semantic interface to incorporating the contemporary semantic theories for natural language into logic grammar. The semantic interface offers a new perspective on logic grammars in that they can be seen as generalizations of natural language semantics as well as its syntax. Such a comprehensive approach to natural language understanding seems possible only if we base logic grammars with the proposed semantic interface. There is still much work to be done when one applies the semantic interface to other semantics theories than treated in this paper. But the semantic interface is the key to a unified treatment of syntax and semantics. The semantic interface can also be used to elaborate the idea of semantically constrained parsing developed by Akama and Ishikawa (1989), by specifying the interaction between syntax and semantics via the semantic interface. It is expected that the semantic interface will contribute to confirming the fundamental methodological unity of logic-based natural language understanding systems.

Reference

Abramson, H. (1984) 'Definite Clause Translation Grammars,' *Proc. of IEEE 1st Symposium on Logic Programming*, 233-241.

Abramson, H. (1988) 'Metarules and an Approach to Conjunction in Definite Clause Translation Grammars,' *Proc. of 5th International Conference and Symposium on Logic Programming*, 233-248, MIT Press, Cambridge.

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

Akama, S. (1986) 'Methodology and Verifiability in Montague Grammar,' *Proc. of COLING* '86, August, Bonn, West Germany, 88-90.

Akama, S. and Ishikawa, A. (1989) 'Semantically Constrained Parsing and Logic Programing,' H. Abramson and M. H. Rogers (eds.), *Meta-Programming in Logic Programing*, 157-168, MIT Press, Cambridge.

Akama, S. and Kawamori, M. (1988) 'Data Semantics in Logic Programming Framework,' V. Dahl and P. Saint-Dizzier (eds.), *Natural Language Understanding and Logic Programing*, Ⅱ, 135-151, North-Holland, Amsterdam.

Barwise, J. and Perry, J. (1983) *Situations and Attitudes*, MIT Press, Cambridge.

Bowen, K. A. (1985) 'Meta-level Programming and Knowledge Representation,' *New Generation Computing* 3, 359-383.

Bowen, K. A. and Kowalski, R. A. (1982) 'Amalgamating Language and Metalanguage in Logic Programming,' K. L. Clark and S. A. Tarnlund (eds.), *Logic Programming*, 153-172, Academic Press, New York.

Comerauer, A. (1978) 'Metamorphosis Grammars,' L. Bold (ed.), *Natural language Communication with Computers*, 133-189, Lecture Notes in Computer Science 63, Springer, Berlin.

Dahl, V. (1981) 'Translating Spanish into Logic through Logic,' *Computational Linguistics* 7, 149-164.

Dahl, V. and Saint-Dizier, P. (1987) *Constrained Discontinuous Grammars-a Linguistically Motivated Tool for Processing Language*, Simon Fraser University.

Fitting, M. (1985) 'A Kripke-Kleene Semantics for Logic Programs,' *Journal of Logic Programming* 4, 295-312.

Heim, I. (1982) *The Semantics of Definite and Indefinite Noun Phrases*, Ph. D. thesis, University of Massachusetts, Amherst.

Kamp. H. (1982) 'A Theory of Truth and Semantic Representation,' J. Groenendijk, Th. Janssen and M. Stokhof (eds.) *Truth, Interpretations and Information*, 1-41, Forris, Amsterdam.

Landman, F. (1986) *Toward a Theory of Information*, Foris, Dordrecht.

Montague, R. (1974) *Formal Philosophy*, R. H. Thomason (ed.), Yale University Press, new Haven.

Nadathur, G. (1987) *A Higher-Order Logic as the Basis for Logic Programming*, Ph. D. thesis, University of Pennsylvania.

Pereira, F. (1983) 'Logic for Natural Language Analysis,' SRI Technical Report 275.

Pereira, F. and Shieber, S. M. (1988) Prolog and Natural-Language Analysis, CSLI Lecture Notes 10, Stanford University.

Pereira, F. and Warren, D. H. D. (1980) 'Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks,' *Artificial Intelligence* 13, 231-278.

Sait-Dizier, P. (1986) 'An Approach to Natural Alnguage Semantics in Logic Programming,' *Journal of Logic Programming* 4, 329-356.

Sedogbo, C. (1988) 'SYLOG: A DRT System in Prolog,' V. Dahl and P. Saint-Dizier (eds.), *Natural Language Understanding and Logic Programming*, Ⅱ. 185-201. North-Holland, Amsterdam.

Van Benthem, J. (1986) *Essays in Logical Semantics*, Reidel, Dordrecht.

Veltman, F. (1985) *Logic for Conditionals*, Ph. D. thesis, University of Amsterdam.

Warren, D. H. D. (1982) 'Higher-Order Extensions to Prolog : Are They Needed?,' *Machine Intelligence* 10, 441-454, Ellis Horwood, Chichester.

Warren, D. S. (1983) 'Using λ-Calculus to Represent Meanings in Logic Grammar,' *Proc. of 21st Annual Meeting of the ACL*, 52-56.

Dr. Seiki Akama
Fujitsu, Ltd.
3-9-8 Shin-Yokohama
Yokohama 222
Japan

Prof. Akira Ishikawa
Department of English Language and Studies
Sophia University
7 Kioi-cho, Chiyoda-ku, Tokyo, 102
Japan