ICOT Technical Memorandum: TM-0053

TM-0053

Object-Oriented Parser in the Logic Programming Language ESP

Hideo Miyoshi and Koichi Furukawa

April, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Object-Oriented Parser in the Logic Programming Language ESP

Hideo Miyoshi and Koichi Furukawa

Institute for New Generation Computer Technology(ICOT)

Mitakokusai Building, 21F
1-4-28, Mita, Minato-ku, Tokyo, 108, Japan

Abstract

In this paper we propose an object-oriented parsing mechanism for logic programming languages. DCG, XG, and MG are well-known formalisms in logic programming. Because of the correspondence between their grammatical rules and those of programs (Horn clauses), arguments, and extraconditions, these formal frameworks have powerful description capabilities for implementing grammatical rules. However, in the case of implementing a large-scale system in these formalisms, the number of arguments and extraconditions increases, thus decreasing the readability of the program and making it difficult to modify. In our object-oriented parser, each program component is abstracted as a class, and access between the two classes is performed by a message-passing mechanism. Grammatical categories are also abstracted as classes. Therefore, the intrinsic grammatical features which are implemented as predicate arguments in DCG, are described as instances of category classes. This helps to simplify Horn clause grammar rule description. Being implemented in the logic programming language ESP, the fundamental mechanism of DCG is also applicable to our parser.

1. Introduction and Background

The logic programming language Prolog is based on first-order predicate logic and has features that are especially useful for symbol computation. DCG (Definite Clause Grammar) [PW80], embedded in DECsystem-10 Prolog [PPW78], is a good formalism for analyzing a context-free language, and XG (Extraposition Grammar) [Pe83] and MG (Metamorphosis Grammar) [Co78] are good choice for analyzing a type-0 language. In DCG, each context-free grammar rule is represented as a Prolog Horn clause , and each grammar category in a rule is treated as a predicate in the clause. Therefore, the elements of a context-free grammar are translated, one-to-one, into those of a Prolog program. Context-sensitive information can be handled easily in arguments of Prolog predicates. Furthermore, Prolog predicates can be arbitrarily inserted into grammar rules. These capabilities facilitate the use of the parser in combination with such auxiliary routines as semantic checking. Consequently, several systems for natural language processing applications, (for example a database query system), have been implemented in Prolog [Da82, Mc82, Pe83]. However, when a large-scale natural language processing system is implemented using these formalisms, the number of grammar rules, syntactic and semantic features, and constraints inevitably increase. If all of these features and constraints are embedded in the grammar rule as the arguments of a category or Prolog predicate call, the grammar rule will become too complex. This will decrease the readability of the grammar and make modifications difficult.

On the other hand, object-oriented methodology has recently begun to look very attractive in this regard, and several object-oriented programming systems have been developed, such as SIMULA [ND78], Smalltalk-80 [RG83], and Loops [BS83]. Since the object-oriented notion of

data abstraction provides the function of information hiding and locality, the internal data structure is hidden from the outside modules and only a specifically defined message can be used to access the data [SY81]. These facilities provide the user with methods to develop programs that are highly reliable and understandable, and can be easily modified. So far, several natural language processing systems have been developed using the object-oriented mechanism [Ph83].

To remedy the defects of DCG or XG mentioned above, we propose an object-oriented parsing mechanism to be integrated into the logic programming language, and we have implemented an experimental small-scale parser. The main features of this parser are as follows:

(1) Implementation in ESP [Ch84]

As described in section 2, ESP has two aspects: a logic programming language, and an object-oriented language. Therefore, basic methodologies that have already been established in the logic programming framework, such as DCG, can be used. We can also abstract the program components using the object-oriented mechanism. By doing this, the grammar rules can be simplified.

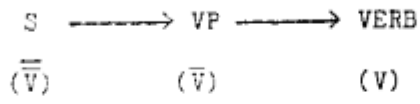
(2) Abstraction of Grammatical Categories

Each grammatical category (nonterminal symbol) is abstracted as a class. A category class has features representing intrinsic grammatical information of the category for example syntactic information. These features are implemented as the slots of a category class. While these features are implemented as the arguments of a category in DCG formalism, an instance which contains those features is embedded in the category as one argument in our program. The number of the arguments in the grammar rule can be reduced using this mechanism.

(3) Using Inheritance Mechanism

In order to define the slots of the grammatical categories which are in X-bar phrasal level [Ja74], the inheritance mechanism is used. HFC(Head Feature Convention) in Generalized Phrase Structure Grammar (GPSG) [GP82] by Dr. Gerald Gazdar is easily implemented by this method.

Example.



Class 'VP' inherits class 'S'.
Class 'VERB' inherits class 'VP'.

An overview of the programming language ESP is given in section 2. In section 3, the object-oriented parsing mechanism and its implementation in ESP are presented.

2. Programming Language ESP

In this section, we give an overview of the programming language ESP. ESP is a logic programming language based on Prolog; it is intended for use on the Sequential Inference Machine (PSI) [NYY83] currently being developed at the ICOT research center. The main features of ESP are as follows:

- Unification, as the basic parameter passing mechanism

- Backtracking, as the basic control structure
- Various built-in predicates
- Object-oriented calling mechanism

2.1 Object

An object in ESP represents an axiom set, and consists of a class object and an instance object. The axiom set to be used in a certain call can be specified by giving an object as the first argument of a call. An object may have slots, each of which has its own name and value.

2.2 Class and Inheritance

An object class, or simply a class, defines the characteristics common to a group of similar objects, i.e., objects that differ only in their slot values. An object belonging to a class is said to be an instance of that class. A class definition consists of nature definitions, slot definitions, and clause definitions. The nature definition defines the inheritance relationship between the classes. ESP provides a multiple inheritance mechanism like that of the Flavor system [WM81]. All class definitions whose names are given in the nature definition are inherited. The slot definition defines the slot used in the class. Clause definitions are used for defining Prolog-like clauses. The syntax of the class definition is shown in Figure 2.1.

```
<class definition> ::=
   "class" <class name>
        [ <macro bank declaration> ]
   "has"
        [ <nature definition> ";" ]
        { <class slot definition> ";" }
 [ "instance"
        { <instance slot definition> ";" }
        { <instance clause definition> ";" } ]
 [ "local"
        { <local clause definition> ";" } ]
   "end" "."
```

Figure 2.1 Syntax of the Class Definition in ESP

In Figure 2.1, "X" indicates a terminal symbol, X. { X } indicates an arbitrary number of appearances of X (including zero) . [X] indicates X or void, i.e., that X is optional. There are two kinds of slots definitions: one is for slots of the class itself, called class slots; the other is for slots of the class instance, called instance slots. This is also true of clause definitions. A local clause is a non-object-oriented local predicate used only in one class.

3. Implementation

In this section, we present the implementation of the object-oriented parser in ESP. The program components are described in 3.1. In 3.2 and 3.3, the examples of a simple program and a parsing result are given.

3.1 Program components

This program consists of the following five components.

(1) Set of Phrase Structure Rule Classes
Basically, the methods of analyzing the left-hand-side category of a context-free grammar rule are described in each phrase structure rule class. The method generates an instance of a category class during parsing. Some classes send messages for dictionary looking up to a dictionary class, or send messages for constraint checking to a constraint class.

(2) Dictionary Class
The dictionary looking up methods are described in this class. The methods also perform a feature instantiation to the instance of a lexical category class.

(3) Set of Category Classes
The slot definitions in each category class describe the intrinsic grammatical features of categories. The structure of a category feature is a two dimensional tree, as in GPSG.

(4) Set of Constraint Classes
The method of checking grammatical constraints is described in each constraint class. At present, HFC (Head Feature Convention) and CAP (Control Agreement Principle) in GPSG are implemented.

(5) Set of Feature Classes
A category class feature set is defined in the slot definitions of each feature class.

The parsing model for this object-oriented parser is shown in Figure 3.1.

3.2 Program Example

An example of a program that analyzes a simple noun phrase based on this object-oriented parsing model is presented here.

(1) Set of Phrase Structure Rule Classes

Figure 3.2 shows the set of phrase structure rule classes. The method in the class 'np_rule' analyzes the following phrase structure:

NP --> DET NOUN

The method calls ':head_of' and ':control' are messages to the constraint classes 'hfc' and 'cap' respectively. The methods in the classes 'det_rule' and 'noun_rule' are used to consult a dictionary of determiners and nouns.

(2) Dictionary Class

Figure 3.3 shows the dictionary class.

```
%-------------------------------------------------
% dictionary class definition
class dict_cls
      :dict(D,these,[these|X],X,DET) :-
         DET!head!agreement!number := plural ,!;
      :dict(D,men,[men|X],X,NOUN) :-
         NOUN!head!agreement!number := plural,
         NOUN!head!gender := male ,!;
end.
```

Figure 3.3 Dictionary Class

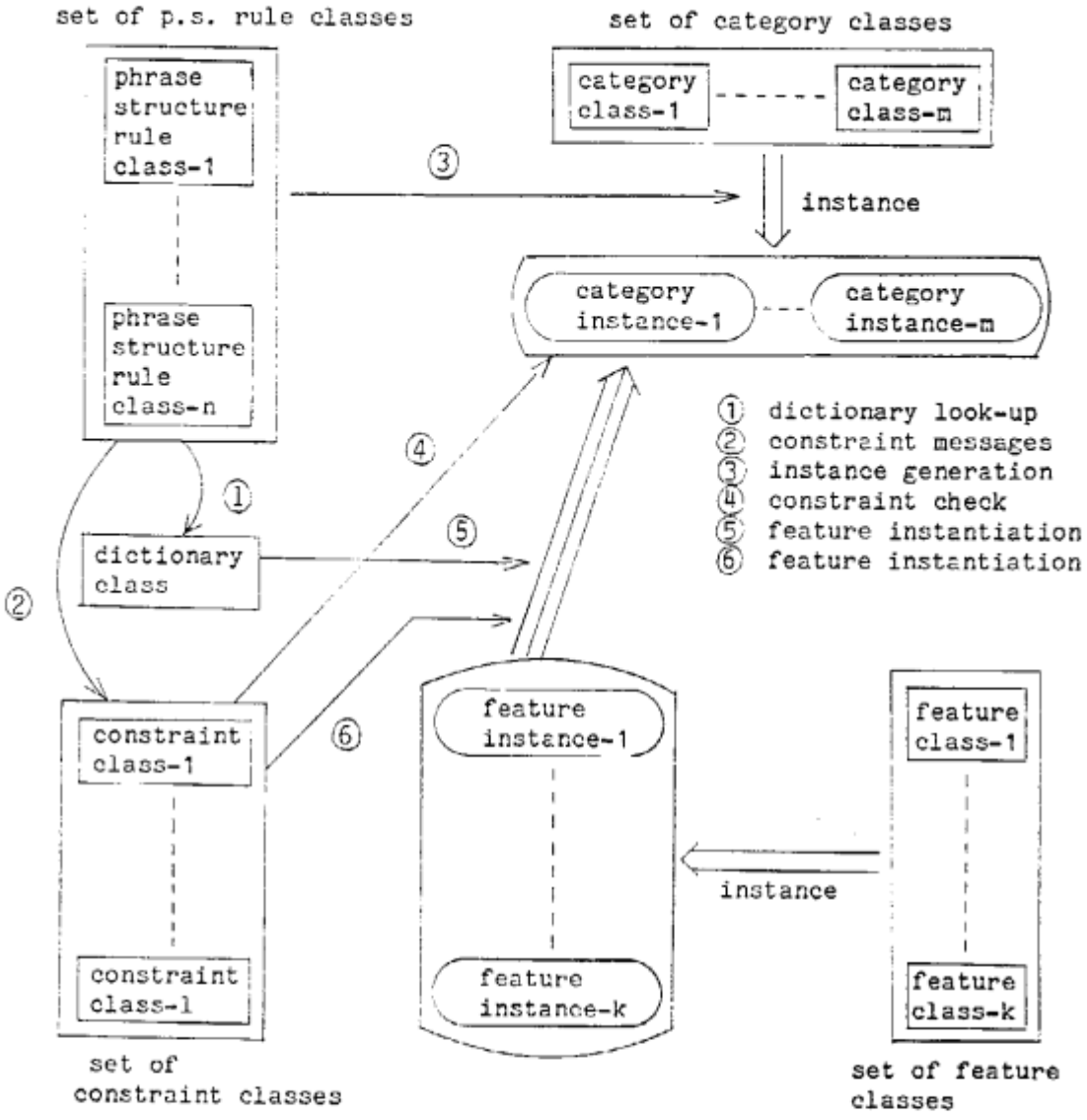


Figure 3.1 Parsing Model of the Object-Oriended Parser

```
%-----------------------------------------------
% np_rule class definition
class np_rule has
      :np(0,np(DETs,NOUNs),X0,X,NP) :- :new(#np,NP),
         :det(#det_rule,DETs,X0,X1,DET),
         :noun(#noun_rule,NOUNs,X1,X,NOUN),
         :head_of(#hfc,NOUN,NP),
         :control(#cap,NOUN,DET) ,!;
end.
%-----------------------------------------------
% det_rule class definition
class det_rule has
      :det(0,det(DETs),X0,X,DET) :- :new(#det,DET),
         :dict(#dict,DETs,X0,X,DET),!;
end.
```

```
%-----------------------------------------------
% noun class definition
class noun_rule has
      :noun(0,noun(NOUNs),X0,X,NOUN) :- :new(#noun,NOUN),
         :dict(#dict,NOUNs,X0,X,NOUN),!;
end.
```

Figure 3.2 Set of Phrase Structure Rule Classes

The class 'dict' has methods to look up the words 'these' and 'men'. The body of the method performs the feature instantiation to the instance of a lexical category classes.

(3) Set of Category Classes and Set of Feature Classes

Figure 3.4 shows the class definition of grammatical category 'np', 'det', and 'noun'.

```
%-----------------------------------------------
% category class
% np class definition
class np has
instance
    attribute
      head is noun_head_features;
end.
%-----------------------------------------------
% category class
% det class definition
class det has
instance
    attribute
      head is det_head_features;
end.
%-----------------------------------------------
% category class
% noun class definition
class noun has
      nature
         np;
end.
```

Figure 3.4 Set of Category Classes

Figure 3.5 shows the class definition of the set of feature classes. The semantics of the class definition 'np' is as follows:

The instance of the class 'np' has a slot 'head', and its slot value is an instance of the feature class 'noun_head_feature' in Figure 3.5. The feature class 'noun_head_feature' has two slots, namely 'gender' and 'agreement'. The slot value of 'agreement' is an instance of the feature class 'noun_head_agreement_features'. The class 'noun_head_agreement_features' has a slot 'number'. By the mechanism mentioned above, the instance of the class 'np' can have a complex feature structure in its slot, as shown in Figure 3.6.

By the same mechanism, the instance of the class 'det' has the following feature structure (Figure 3.7):

By the nature definition 'nature np;' in the class definition of the category 'noun', the class definition of the 'np' is inherited to the class 'noun'. Therefore, an instance of the class 'noun' has the same feature structure as the instance of the class 'np'.

```
%-----------------------------------------------
% feature class
% noun_head_feature class definition
class noun_head_features has
instance
   attribute
      gender,
      agreement is noun_head_agreement_features;
end.
%-----------------------------------------------
% feature class
% noun_head_agreement_features class definition
class noun_head_agreement_features has
instance
   attribute
      number;
end.
%-----------------------------------------------
% feature class
% det_head_features class definition
class det_head_features has
instance
   attribute
      agreement is det_head_agreement_features;
end.
%-----------------------------------------------
% feature class
% det_head_agreement_features class definition
class det_head_agreement_features has
instance
   attribute
      number;
end.
```

Figure 3.5 Set of Feature Classes

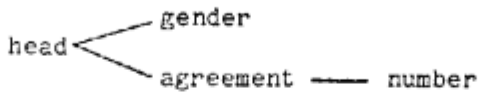


Figure 3.6 Feature Structure of an Instance of a Class 'np'

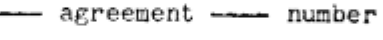


Figure 3.7 Feature Structure of an Instance of a Class 'det'

(4) Set of Constraint Classes

Figure 3.8 shows the constraint classes used in this program. The class 'hfc' is an implementation of Head Feature Convention (HFC), which is the terminology used in GPSG, such that mother and head carry the same feature. The class 'cap' is an implementation of the GPSG Control Agreement Principle (CAP) in GPSG. CAP is a relation holding between sisters in a rule, such that, if A controls B, then the agreement feature of A is the same as that of B. Figure 3.9 shows an illustrative example of

```
%-----------------------------------------------
% HFC class definition
% Head Feature Convention
%   If A is the head of B then HEAD(A) = HEAD(B)
class hfc has
      :head_of(HFC,HEAD,PARENT) :-
         PARENT!head := HEAD!head ,!;
end.
%-----------------------------------------------
% CAP class definition
% Control Agreement Principle
%   If A controls B then AGR(A) = AGR(B)
class cap has
      :control(CAP,CONTROLLER,CONTROLLEE) :-
      agr_equal(CONTROLLER!head!agreement,
                CONTROLLEE!head!agreement) ,!;
local
      agr_equal(A,B) :-
         A!number == B!number ,!;
      agr_equal(A,B) :-
         A!number == void ,!;
      agr_equal(A,B) :-
         B!number == void ,!;
end.
```

Figure 3.8 Set of Constraint Classes

HFC and CAP [GP82].

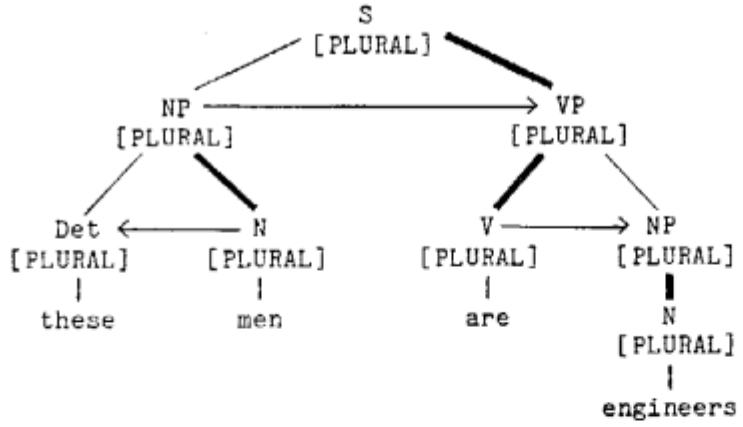


Figure 3.9 An Example of HFC and CAP

In Figure 3.9, an arrow indicates the control relation, and a bold line indicates the HFC.

3.3 Parsing Example

Here, we show the parsing result by the program presented in 3.2. The program parses the input noun phrase 'these men'. The top goal is given by

```
':- :np(#np_rule,N_St,[these,men],[],NP).'
```

and an agreement feature check is performed between 'these' and 'men' by CAP. The term which indicates the phrase structure (Figure 3.10) is instantiated to the second argument of ':np', and the instance of 'np' that has the feature structure shown in Figure 3.11 is instantiated to the fifth

argument of ':np'.

Figure 3.10 Phrase Structure of 'these men'

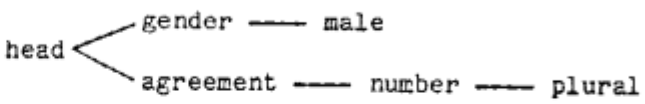


Figure 3.11 Feature Structure of the Instance of 'np'

4. Summary

This work describes an object-oriented parsing mechanism and its implementation in the logic programming language ESP. Each program component, as well as grammatical categories, are abstracted as class objects. Access between program components is performed by a message-passing mechanism. The abstraction of grammatical categories helps to reduce the number of arguments in a rule, and enables the user to update category features without changing the grammar rules. The inheritance mechanism is used to fix the features of a category class. This framework is thought to be suitable for implementing a complex feature system such as GPSG. This object-oriented parser is implemented in the ESP Cross System on DEC-2060.

Acknowledgements

The authors are thankful to H. Yasukawa of ICOT for his helpful discussions. The authors would also like to thank to Dr. G. Gazdar of Univ. of Sussex and Dr. T. Gunji of Toyahashi Univ. of Technology for their helpful comments on GPSG, and Dr. T. Chikayama of ICOT for his useful comments on ESP. The authors are grateful to the Dr. K. Fuchi, Director of the ICOT Research Center, for providing the opportunity to conduct this research.

[References]

[PW80] Pereira, F. and Warren, D.; Definite Clause Grammar for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, 13, 1980, pp231-278.

[PPW78] Pereira, L. and Pereira, F. and Warren, D.; User's Guide to DECsystem-10 Prolog, Dev. de Informatica, LNEC, Lisbon and Dept. of AI, University of Edinburgh, September, 1978.

[Da82] Dahl, V.; On Database Systems Development through Logic, ACM Transactions on Database Systems, Vol.7, No.1, March 1982, pp102-123.

[Mc82] McCord, M. C.; Using Slots and Modifiers in Logic Grammars for Natural Language, Artificial Intelligence, 18, 1982, pp327-367.

[Pe83] Pereira, F.; Logic for Natural Language Analysis, Technical Note 275, SRI International, January, 1983.

[Co78] Colmerauer, A.; Metamorphosis Grammars, Natural Language Communication with Computers, Bole, L.(ed.), Springer-Verlag, 1978.

[ND78] Nygaard, K. and Dahl, O. J.; The Development of the SIMULA Languages, ACM SIGPLAN Notices, Vol.13, No.8, August, 1978, pp245-272.

[RG83] Goldberg, A. and Robson, D.; Smalltalk-80: The Language and its Implementation, Addison-Wesley Publishing Company, 1983.

[BS83] Bobrow, D. G. and Stefik, M.; The LOOPS Manual (Preliminary Version), XEROX PARC Knowledge-based VLSI Design Group Memo, KB-VLST-81-13, 1983.

[SY81] Sado, K. and Yonezawa, A.; A Tutorial on Abstract Data Type Oriented Languages (in Japanese), Journal of Information Processing, Vol.22, No.6, June, 1981.

[Ph83] Phillips, B.; An Object-Oriented Parser for Text Understanding, Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 1983, pp690-692.

[Ch84] Chikayama, T.; ESP Reference Manual, ICOT Research Center Technical Report, No.44, February, 1984.

[Ja74] Jackendoff, R.; Introduction to the X-bar Convention, Indiana University Linguistics Club, October, 1974.

[GP82] Gazdar, G. and Pullum, G. K.; Generalized Phrase Structure Grammar: A Theoretical Synopsis, Indiana University Linguistics Club, August, 1982.

[NYY83] Nishikawa, H., Yokota, M., Yamamoto, A., Taki, K., and Uchida, S.; The Personal Inference Machine(PSI): Its Design Philosophy and Machine Architecture, ICOT Research Center Technical Report No.13, June, 1983.

[WM81] Weinreb, D. and Moon, D; Lisp Machine Manual, 4th ed., Symbolics, Inc., 1981.