ICOT Technical Memorandum: TM-1282

TM-1282

QUIXOTE as a Tool for Natural Language Processing

by
S. Tojo (MRI), H. Tsuda,
H. Yasukawa (Matsushita), K. Yokota
& Y. Morita (Oki)

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~5

Institute for New Generation Computer Technology

$\mathcal{QUIXOTE}$ as a Tool for Natural Language Processing

Satoshi Tojo, Hiroshi Tsuda
Hideki Yasukawa, Kazumasa Yokota, and Yukihiro Morita
Institute for New Generation Computer Technology (ICOT)
4-28 Mita 1, Minato-ku, Tokyo 108, JAPAN

Abstract

We developed a language $\mathcal{QUIXOTE}$ as a tool to deal with various information of natural language processing (NLP). $\mathcal{QUIXOTE}$ is a hybrid language of deductive object-oriented database (DOOD) and constraint logic programming (CLP) language. The new mechanism of $\mathcal{QUIXOTE}$ is a combination of an object-orientation concept such as *object identity* and the concept of *module* that classifies a large knowledge base. In addition, its logical inference system is extended to be able to make restricted abduction. We first apply $\mathcal{QUIXOTE}$ to the sorted feature structure of constraint-based grammar formalisms. Next, we show that $\mathcal{QUIXOTE}$ can contribute to the description of situation-based semantics. We implemented a system to make abductive reasoning to clarify hidden information. Also we resolve the problem of noun phrase reference.

1 Basic Structure of $\mathcal{QUIXOTE}$

This section outlines $\mathcal{QUIXOTE}$[13, 14] as a tool for NLP. $\mathcal{QUIXOTE}$ is classified as both a DOOD and CLP language. It has the following features that are also useful in NLP:

- OO features, such as object identity,
- attribute-value data structure with subsumption constraints,
- efficient information description with modules and inheritance hierarchy, and
- question and answer with assumptions.

$\mathcal{QUIXOTE}$ system is implemented with the KL1 language, a parallel logic programming language for the parallel inference machine, PIM, in ICOT.

Generally, $\mathcal{QUIXOTE}$ programs consist of subsumption relations among basic object terms (1.1), submodule relations among modules (1.3), and rules.

$\mathcal{QUIXOTE}$ rules have the following syntax.

$$\overbrace{m_0 :: H}^{head} \quad \overbrace{\mid HC}^{head\_constraint} \quad \Leftarrow \overbrace{m_1 : B_1, \cdots}^{body} \quad \overbrace{\parallel BC}^{body\_constraint} \quad ;;$$

Here, H and B_i are *object/attribute terms*. HC and BC are *constraints*. m_i, called a *module identifier*, specifies the module in which terms or rules resides. The body, constraint, and module identifier can be omitted.

The above rule resides in module m_0. It implies that, if every B_i holds in a module m_i under constraints BC, then H and constraints HC hold in m_0.

1.1 Object Term

Let Obj,$Bobj$,$Cobj$, and Var be a collection of *object terms*, *basic object terms*, *complex object terms*, and *variables*, respectively. Then, $Obj = Bobj \cup Cobj \cup Var$.

Basic object terms are atomic symbols such as `cat` and `apple`.

Subsumption relation $\sqsubseteq$, is a partial order relation between basic object terms. $\mathtt{a} \sqsubseteq \mathtt{b}$ means that `a` is more specific than `b`, or `a` ISA `b`, for example, $\mathtt{cat} \sqsubseteq \mathtt{animal}$ or $\mathtt{animal} \sqsubseteq \mathtt{creature}$. Special basic object terms $\bot$ and $\top$ satisfy $\forall x \in Bobj, \bot \sqsubseteq x, x \sqsubseteq \top$. $\mathcal{QUIXOTE}$ comprises a complete lattice from $< Bobj, \sqsubseteq >$.

A *complex object term* is a term having the form $o[l_1 = v_1, l_2 = v_2, \cdots]$, where $o \in Bobj$ and $\forall i, l_i \in Bobj, v_i \in Obj$. Each label l_i is called an *intrinsic attribute*. The order of labels is not significant, so `o[l=a,m=b]` and `o[m=b,l=a]`, for example, are identical.

The subsumption relation is extended to treat complex object terms. For example, when $\mathtt{cat} \sqsubseteq \mathtt{animal}$,

$$\mathtt{cat[age = 2, sex = male]} \quad \sqsubseteq \quad \mathtt{cat[age = 2]}$$
$$\mathtt{cat[color = white]} \sqsubseteq \mathtt{animal}.$$

1.2 Attribute Term, Constraints, and Property Inheritance

An *attribute term* is an object term with property specifications represented as:

$$head/[l_1 op_1 v_1, l_2 op_2 v_2 \ldots],$$

where $head \in Obj$ and $\forall i, l_i \in Bobj, op_i \in \{=, \leftarrow, \rightarrow\}, v_i \in Obj$. Each label l_i is called an *extrinsic attribute*.

An attribute term can be transformed into an object term with a set of constraints.

$$\begin{array}{rcl} \mathtt{o/[l = X]} & \leftrightarrow & \mathtt{o} \mid \{\mathtt{o.l} \cong \mathtt{X}\} \\ \mathtt{o/[m \rightarrow U]} & \Leftrightarrow & \mathtt{o} \mid \{\mathtt{o.m} \sqsubseteq \mathtt{U}\} \\ \mathtt{o/[n \leftarrow V]} & \Leftrightarrow & \mathtt{o} \mid \{\mathtt{V} \sqsubseteq \mathtt{o.n}\} \end{array}$$

$O \mid C$ means an object term O with constraint C. `o.l` is called a *dotted term* and specifies the value of the `l` attribute of an object term `o`. *Constraints* is a set of formulas having the form $< term >< op >< term >$, where $< term >$ is an object/dotted term, and $< op > \in \{\cong, \sqsubseteq\}$.

By default, the attributes of an object are inherited by its related objects in terms of $\sqsubseteq$, as follows. This is called the *property inheritance*.

$$\mathtt{o} \sqsubseteq \mathtt{p} \Rightarrow \forall \mathtt{l}, \mathtt{o.l} \sqsubseteq \mathtt{p.l}$$

For example, when `macintosh` $\sqsubseteq$ `apple` and `apple.color` $\cong$ `red`, then `macintosh.color` $\sqsubseteq$ `red` holds.

For complex object terms, however, the values of intrinsic attributes override those of extrinsic ones. For example, even if `apple.color` $\cong$ **`red`** holds, `apple[color=green].color` remains **`green`** and is not subsumed by `red`.

1.3 Module and Object Identity

A QUIXOTE program can be divided into several modules. Each module is identified by an object term called a *module identifier* and consists of a set of rules.

Like many programming languages, QUIXOTE has an inheritance mechanism between modules called *rule inheritance*. Submodule relation $\sqsupseteq_S$ between module identifiers specifies the rule inheritance. For example, when `john` $\sqsupseteq_S$ **`common_knowledge`**, all rules in the module **`common_knowledge`** are inherited by the module `john`. Here, `john` is called a *submodule*, while **`common_knowledge`** is a *supermodule*.

To realize rule inheritance exceptions, each rule can have an inheritance mode `o`,`l`, or `ol`. A rule with `o` *overrides* inherited rules which have the same head. A rule with `l` is a *local* rule, hence is not inherited by the submodules. Inheritance mode `ol` is a combination of `o` and `l`.

The following example describes the knowledge "In Europe, cars usually drive on the right. But cars drive on the left in England."

```
england ⊒S europe, france ⊒S europe
europe :: car/[drive = right];;
england :: (o)car/[drive = left];;
```

Two object terms are *identical* if they currently bind to a literally equal object term except for the order of labels. For example, if `X` binds to `a`, `o[l=X,m=b]` and `o[m=b,l=a]` are identical. In this sense, ground object terms work as object identifiers.

Within a module, the values of the identical attributes of identical objects must be equal. In different modules that are not in the submodule relations, however, identical objects can have distinct values.

For example, the following program becomes inconsistent because `john` has a different `age` value in the module `sit_1993`.

```
sit_1993 :: john/[age=20];;
sit_1993 :: john/[age=30];;
```

However, the following is not inconsistent, when the submodule relation does not hold between `sit_1983` and `sit_1993`.

```
sit_1983 :: john/[age=20];;
sit_1993 :: john/[age=30];;
```

1.4 Answer with Assumptions

A query sentences have the following forms.

$$?- m_1 : G_1, \cdots \parallel C.$$
$$?- m_1 : G_1, \cdots \parallel C;; PG.$$

Here, m_i is a module identifier, G_i an object/attribute term, C a constraint, and PG a program. In response to a query, QUIXOTE returns answer substitutions with a set of constraints among dotted terms called *assumptions*.

Except for constraints among dotted terms, QUIXOTE works like a conventional CLP language [4]. However, dotted term constraints in the body constraints are accumulated as assumptions if they are not satisfied by the head constraints. Assumptions can be seen as lacking information in the DB. Deriving assumptions is a kind of abduction.

The second type of query, which is often used in hypothetical reasoning applications, adds the additional

programs before the inference. For details of the procedural semantics of QUIXOTE, see [8].

Consider the following example. It indicates that there is a book, and that the shipping fee is 500 yen if it is hard-cover, or 300 yen if soft-cover.

```
sit::book;;
sit::ship[fee=500] <= book/[cover->hard];;
sit::ship[fee=300] <= book/[cover->soft];;
```

The first clause tells us only of the existence of an object, **book**, and nothing about its properties. The second clause means that if book exists in sit and the cover property is subsumed by **hard**, ship[fee=500] holds in sit. How about asking a query ?-sit:ship[fee=Yen], which asks the shipping fee of the book, to this program? QUIXOTE returns the following two independent answers. [1]

```
Yen=500  if  sit:book.cover=<hard
Yen=300  if  sit:book.cover=<soft
```

Each answer makes an assumption about the cover property of book which comes from the body of the second or third clauses. Neither constraints are satisfied by the head constraint, which is empty in this example, so they are accumulated as assumptions.

2 NLP in QUIXOTE

2.1 Representation of Feature Structure

Consider the sorted (typed) feature structure[3], upon which the latest framework of HPSG[9] is constructed. It is the feature structure whose nodes are labeled with sort symbols. The inheritance mechanism between supersorts and subsorts enables the efficient representation of lexicon or grammar.

For example (1) is a simple HPSG-like sorted feature structure representing the word "run." **word**, **vp**, **run**, **phrase**, or **np** specifies the sort of each structure.

$$\begin{bmatrix} \textbf{word} \\ \texttt{CAT} : [\textbf{vp}] \\ \texttt{PH} : [\textbf{run}] \\ \texttt{SUBCAT} : \begin{bmatrix} \textbf{phrase} \\ \texttt{CAT} : [\textbf{np}] \end{bmatrix} \end{bmatrix} \quad (1)$$

Attribute terms in QUIXOTE are capable of partially describing information. The basic object term and subsumption relation $< Bobj, \sqsubseteq >$ in QUIXOTE naturally comprises types and their inheritance hierarchies. To treat the inheritance and information partiality, it is natural to describe a sorted feature structure with an attribute term whose head is a basic object term representing the sort. The property inheritance mechanism (1.2) corresponds to the inheritance between sorted feature structures. (1) is represented by the following subsumption relations and two attribute terms.

```
Y=<word, Z=<phrase
Y/[cat->vp,ph->run,subcat=Z]
Z/[cat->np]
```

In comparison with related KR languages, PST (Partially Specified Term) in CIL[6] and ψ-terms in LOGIN[1] are closely related to the feature structures. However, attribute terms in QUIXOTE are more powerful because CIL does not have an inheritance mechanism and LOGIN cannot handle constraints.

2.2 QUIXOTE and Situated Inference

We first define a *situated inference* rule as follows:

$$s_0 \models \sigma_0 \Leftarrow s_1 \models \sigma_1, s_2 \models \sigma_2, \cdots, s_n \models \sigma_n \| C. \quad (2)$$

This sample rule can be interpreted as follows: if s_1 supports σ_1, s_2 supports σ_2, and so on, thus we can infer that s_0 supports σ_0, under constraint C. In QUIXOTE, situated inference rules are rephrased as follows[11]:

situation theory		QUIXOTE
situation	$\Leftrightarrow$	module
infon	$\Leftrightarrow$	object term
role	$\Leftrightarrow$	label
supporting ($\models$)	$\Leftrightarrow$	membership (:)

The most important correspondence between them is the description of a rule, viz. (2), and a QUIXOTE rule (3).

$$m :: \sigma <= m_1 : \tau_1, m_2 : \tau_2, \ldots, m_n : \tau_n \| C. \quad (3)$$

2.3 Abductive Reasoning in QUIXOTE

Let us consider the following famous sentences ([10] p. 15; also in [2] p. 105), that are utterances made by two people, A and B, as an example of conditional reasoning:

A: "If Bizet and Verdi are compatriots, they are Italian." (4)

B: "If Bizet and Verdi are compatriots, they are French." (5)

[1] In the syntax of QUIXOTE system, $\sqsubseteq$ is =<.

The interesting point of sentences (4) and (5) is that the conclusions are different despite application of the same conditional clauses. This is because there is hidden partial knowledge in the two people and this is not mentioned explicitly. The objective of the program introduced below is to determine the nationalities of Bizet and Verdi when they are asked, with inferences on implicitly mentioned information.

The natural language expressions that correspond to (4) and (5) are directly translated into *Quixote* as follows:

```
hypothesis_a ::
    bizet/[nationality = italy] <=
    compatriots[per1=bizet,per2=verdi] ;;
hypothesis_b ::
    verdi/[nationality = france] <=
    compatriots[per1=bizet,per2=verdi] ;;
```

The first rule says that *being compatriots of two persons named Bizet and Verdi* implies that *Bizet's nationality being Italian* in a *hypothesis of person A*. The second rule can be interpreted in a similar way. We need other rules to terminate inference chaining, viz. the definition of `compatriots`, `bizet`, and `verdi`.

```
world :: compatriots[per1=X,per2=Y] <=
  X/[nationality=N1], Y/[nationality=N2]
  || {N1=<nation,N2=<nation,N1==N2} ;;
world :: bizet;;
world :: verdi;;
```

These definitions are valid in every module, viz. in the most general module `world`. We can write the module hierarchy as follows: [2]

```
hypothesis_a >- world ;;
hypothesis_b >- world ;;
```

We need to define the subsumption relation beforehand as follows:

```
nation >= italy ;;
nation >= france ;;
```

The above are all of the rules. Please note that what person *A* knows and what person *B* knows is not mentioned anywhere.

Now, we would like to introduce how *Quixote* works. First, let us ask the nationality of Bizet:

```
?-hypothesis_a:bizet/[nationality=N].
```

The result of inference is as follows:

```
** Answer 1 **
 IF hypothesis_a:verdi.nationality
    == bizet.nationality
    hypothesis_a:verdi.nationality
    =< nation
 THEN
    N == italy
** Answer 2 **
    N == Unbound
```

The system returned two answers. The latter is easy, for it is similar to conventional Prolog: we can infer nothing of the nationality of Bizet because there is no direct reference to it in the chaining of inference rules. The feature of *Quixote* is the ability to find the former answer. In the subsumption mapping of objects, the system can find a minimal model to satisfy the query, and the system returns the model with conditions, that is from `if` to `then`.

2.4 Treatment of Noun Phrase Reference

In this section, we would like to focus upon the problem of noun phrase reference. This problem appears in various forms: opaqueness (*de re* and *de dicto*), anaphora scope, metaphor and metonymy, confusion of roles and attributes-values, and so on. Here, we will analyze the following sentences.

Hitchcock saw himself in that movie. (6)

Hitchcock saw a unicorn in that movie. (7)

(6) has several possible interpretations: one is that a person named Hitchcock saw a movie and he was playing the role of someone. The other is that Hitchcock saw someone was playing the role of Hitchcock (in a biographical movie, for example).[3] (7) refers to an object that does not exist in this world.

Let us consider the following program:

```
real::see[agt=hitchcock,obj=X] <=
  movie:cast[name=hitchcock,
             title=hitchcock_life,act=X];;
real::hitchcock;;
movie::cast[name=X,title=M,act=X] <= M:X;;
movie::cast[name=X,title=M,act=Y]
  <= M:Y/[actor=X],real:X;;
hitchcock_life::
  man[place=bus_stop]/[actor=hitchcock];;
hitchcock_life::
  hitchcock/[actor=orson_welles];;
```

[2] In *Quixote* syntax, $\sqsupseteq_S$ is >-.

[3] There is a possibility that Hitchcock is playing the role of Hitchcock himself.

The first three lines of the program above say that some agent called `hitchcock` is watching some actor `X`, in the movie called `hitchcock_life` (life of Hitchcock). The fourth line represents that, in the `real` world, there is a person called `hitchcock`. The fifth to seventh lines claim that in the situation of some movie called `M`, it is probable that:

- the name `X` can refer to the very role in the movie, or
- the name `X` can refer to the role `Y` that is played by `X` in the movie.

Finally in the eighth to eleventh lines, two roles, that of a `man` at a bus stop and that of `hitchcock`, are required in the movie `hitchcock_life`.

Here, let us ask what Hitchcock saw in his real life. We acquire the following answers.

```
?-real:see[obj=X,agt=hitchcock].
** Answer 1 **
    X == man[place=bus_stop]
** Answer 2 **
    X == hitchcock
```

In the case of (7), there is no object that corresponds to the unicorn in the real world, so only one interpretation can be derived by a similar program. This means that the unicorn necessarily exists only in a hypothetical world.

3 Conclusion

We introduced the *QUIXOTE* language as a tool to deal with complicated natural language phenomena. Compared with related works such as F-logic [5] (as a DOOD language), CLP languages [4, 12], KR languages such as LOGIN[1] and CIL[6], situated inference system PROSIT [7], the new mechanism of *QUIXOTE* is summarized as follows. First, an object-orientation concept such as *object identity* is introduced into the logic programming as the fundamental philosophy. Secondly, the concept of *module* enables us local definition in a large knowledge-base. Thirdly, its logical inference system is extended to be able to restricted abduction.

We first applied the sorted feature structure of *QUIXOTE* to constraint-based grammar formalisms, and then we showed that *object identity* and *module* in *QUIXOTE* could contribute to the description of situation-based semantics.

References

[1] H. Aït-Kaci and R. Nasr. LOGIN: A Logic Programming Language with Built-In Inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[2] J. Barwise. *The Situation in Logic*. CSLI Lecture Notes 17, 1989.

[3] B. Carpenter. *The Logic of Typed Feature Structure*. Cambridge University Press, 1992.

[4] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proceedings of the 14th ACM POPL*, pages 111–119, Munich, 1987.

[5] M. Kifer. Logical Foundation of Object-Oriented and Frame-Based Language. *Technical Report 90/14*, SUNY at Stony Brook, June 1990.

[6] K. Mukai and H. Yasukawa. Complex Indeterminates in Prolog and its Application to Discourse Models. *New Generation Computing*, 3(4):441–466, 1985.

[7] H. Nakashima, S. Peters, and H. Schutze. Communication and Inference through Situations. In *Proc. of IJCAI '91*, pages 76–81, 1991.

[8] T. Nishioka, R. Ojima, H. Tsuda, and K. Yokota. Procedural Semantics of a DOOD Programming Language *QUIXOTE*. In *SIG-DBS No.94* of Inf. Proc. Soc. Japan., pages 1–10, 1993. (in Japanese).

[9] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, 1993. (to appear).

[10] W. V. Quine. *Methods of Logic (revised edition)*. New York: Holt, Rinehart, and Winston, 1959.

[11] S. Tojo and H. Yasukawa. Situated Inference of Temporal Information. In *Proc. of FGCS92*, pages 395–404, 1992.

[12] H. Tsuda, K. Hasida, and H. Sirai. JPSG Parser on Constraint Logic Programming. In *Proc. of 4th ACL European Chapter*, pages 95–102, 1989.

[13] H. Yasukawa, H. Tsuda, and K. Yokota. Objects, Properties, and Modules in *QUIXOTE*. In *Proc. of FGCS '92*, pages 257–268, 1992.

[14] K. Yokota and H. Yasukawa. Towards an Integrated Knowledge-Base Management System. In *Proc. of FGCS '92*, pages 89–112, 1992.