自然言語処理 107－2
（1995. 5. 26）

A Pattern-Based Approach for Interactive Clarification of Natural Language Utterances

Hervé BLANCHON, Laurel FAIS, Kyung-Ho LOKEN-KIM & Tsuyoshi MORIMOTO
ATR-ITL
2-2 Hikaridai
Seika-cho, Soraku-gun
Kyoto 619-02 Japan
e-mail: blanchon@itl.atr.co.jp

Abstract

We are going to describe an interactive clarification methodology that has been used to develop two clarification modules for French and English input. A clarification module is made of two parts, a kernel, which is language-independent, and a description of the language-dependent clarification processes.

The clarification methodology works on tree structures. The basic mechanism involved in the recognition of an ambiguity is a pattern matching mechanism. A class of ambiguities is described with one or several pattern beams (sets). Once a pattern beam has been matched, a question is prepared. The question is composed of a number of items equal to the number of patterns in the matched beam. A dialogue item production method is associated with each pattern. These dialogue item production methods are described by a set of basic operators.

The clarification kernel combines a pattern matching module, a beam matching module, the set of basic operators, and a question presentation module. For a given language, a clarification module is described by a clarification session planner, the relevant pattern beams, and the dialogue item production methods associated with each described pattern.

Keywords

Interactive clarification, pattern-matching, beam-matching, ambiguity, clarification framework

パターンを用いた曖昧性の検出と再確認方式

Hervé BLANCHON, Laurel FAIS, Kyung-Ho LOKEN-KIM & Tsuyoshi MORIMOTO
㈱ATR音声翻訳通信研究所

概要：

本論文では、フランス語と英語の話し言葉を対象に、その曖昧性をユーザとインタラクションにより再確認する方法について述べる。曖昧性を検出し、再確認するモジュールは、言語に依存しないカーネル部と、言語に依存した曖昧さの検出および再確認部の2つのモジュールからなる。

曖昧性の検出は基本的に木構造のパターン・マッチングにより実現される。1つの曖昧性に対し、複数の木構造パターン（ビーム）が定義されている。パターンビームがマッチすると、ユーザに対し確認のための質問が行なわれる。1つの質問は、そのビームに定義されたパターン数と同じアイテムで構成される。各質問アイテムをどのような形式とするかは、システムで用意された基本オペレータを用いて各々のパターンごとに定義する。

カーネル部の役割は、パターンマッチング・モジュール、ビームマッチング・モジュール、基本オペレータ、および質問出力モジュールなど結合することにある。

キーワード：
曖昧性の再確認、パターンマッチング、ビームマッチング、曖昧性

Introduction

Natural language (spoken or written) is seen as an attractive modality for interactive computer systems. Recent applications using a natural language interface include multi-modal drawing tools [Caelen 1994 ; Hiyoshi & Shimazu 1994 ; Nishimoto, *et al.* 1994], on-line information retrieval [Haddock 1992 ; Zue, *et al.* 1993 ; Goddeau, *et al.* 1994], oral control systems, and, finally, face to face translation systems [Morimoto, *et al.* 1992 ; Kay, *et al.* 1994].

Nevertheless, natural language input is handled with great difficulty by computers. As natural language is highly ambiguous even in restricted domains, interactive clarification is seen as the solution to produce more robust, fault-tolerant and user-friendly interactive systems.

In this context, the role of an interactive clarification module is to plan interactive sessions enabling the system to recover the information the analysis module has not been able to calculate automatically.

We defined a framework in which two experiments have been conducted involving clarification modules, one for French, at the GETA lab (France), in the context of the LIDIA project [Blanchon 1994b ; Boitet & Blanchon 1995] of dialogue-based machine translation, and one for English, at ATR-ITL (Japan), in the context of interpreting telecommunications [Blanchon & Loken-Kim 1994].

In this paper we will concentrate on the English clarification module. We will first give an overview of the methodology involved. We will then describe the clarification engine, which is the language-independent part of a clarification module. We will next describe the linguistic data which make up the English clarification module. Then we are going to give several examples of the produced clarification dialogues.

1. Overview

1.1. Framework

The framework we propose is based on the manipulation of tree structures. The basic mechanism involved in the recognition of an ambiguity is a pattern matching one. A clarification module is made of two parts, an *engine* (language-independent) and *"linguistic data"* (language-dependent).

A class of ambiguity is described with one or several pattern sets (beams). Once a beam has been matched, a question is prepared. It contains as many items (cf. fig. 20) as the matched beam. An item production method is associated with each pattern. The production of these items is described by a set of basic operators.

The clarification *engine* combines a pattern matching module, a beam matching module, the set of basic operators, and a question presentation module.

Thus, the *linguistic data* of a clarification process is made of pattern beams (`objects`), item production methods (`methods`) and a clarification scheduler (`automaton`) in charge of defining the order in which the beams are matched against the analysis structure.

1.2. Ambiguity Classified

The ambiguities which make up the corpus upon which the clarification mechanism was based were taken from a data base of spontaneous speech collected at ATR-ITL. The conversations, between native speakers of American English, were recorded during an experiment conducted in the Environment for MultiModal Interaction (EMMI) [Loken-Kim, *et al.* 1993], and took place via both telephone and multimedia communication contexts [Fais 1994]. The 17 conversations from the experiment, comprising over 8000 words, were examined by hand, and all detected ambiguities were extracted. Ambiguities due solely to polysemy were disregarded; typical examples of all other types of ambiguity were selected to form the final corpus.

The ambiguities were then classified into several classes. The meta-classes of ambiguity are: syntactic class ambiguities, geometrical ambiguities and décorational ambiguities. For a complete definition of those ambiguity meta-classes cf. [Blanchon 1994a]. Those meta-classes have been refined into several classes as described below.

Syntactic Class
- Noun-Adjective
  - ex: *This is an English speaking agent.*
- Noun-Verb
  - ex: *You can either travel by subway, bus, or taxi.*
- Phrasal-Verb
  - ex: *It is difficult to get out of Kyoto station.*

Geometrical Ambiguity
- Prepositional Attachment
  - ex: *Where can I catch a taxi from Kyoto station?*
- Adverbial Attachment
  - ex: *You can pay for it right on the bus.*
- Conjunction
  - ex: *You can tell him that you are going to the conference center and it should be a 20 minute ride.*

Decorational Ambiguity
- ex: *Good morning conference center.*

The ambiguities are descibed as follows in the clarification module:

Ambiguity	Description
Noun-Adjective	1 type, 2 beams
Noun-Verb	1 type, 1 beam
Phrasal-Verb	not described by beam
Prepositional Attachment	3 types, 5 beams
Adverbial Attachment	2 types; 3 beams
Conjunction	1 type, 1 beam
Decorational Ambiguity	not handled yet

1.3. Implementation

The current implementation is realized in the Common Lisp Object System (CLOS) [Keene 1989], in the Macintosh Common Lisp environment. The only platform-specific module is the dialogue presentation module. Thus, most of the code is portable to any CLOS implemenation.

2. The engine

The clarification engine, which is language independent and is to be reused by each clarification module, consists of:

- a pattern matching mechanism,

- a beam matching mechanism,
- a presentation module, and,
- a set of basic operators.

2.1. The pattern matching mechanism

The patterns are described with a language derived from the one proposed in [Norvig 1992]. Each pattern is described with the syntax described in figure 1.

```
pattern        ::=  ...                          |
                    seg-pattern                  |
                    spl-pattern                  |
                    (pattern . pattern)
spl-pattern    ::=  (?is variable pred args)     |
                    ; check the predicate pred
                    ...
seg-pattern    ::=  ((?+ variable) ...)          |
               ; match a segment of one or more
               ...
variable       ::=  ?character+
```

Figure 1: Extract of the pattern syntax

A pattern describes a family of trees, with constraints on their geometry and labelling.

The pattern matching mechanism is also inspired by [Norvig 1992]'s proposal.

The result of the pattern matching mechanism is a list whose first element (**match-p**) is t if matched, nil if not, and whose second element (**binding**) is a binding list containing the value of each variable defined in the pattern.

2.2. The beam matching mechanism

A family of ambiguities can be defined with several sets of patterns also called beams. Thus, a sentence S, with s solutions S_i, contains the ambiguity described by the beam B made of b patterns P_j if and only if:

- $b < s$
- $\forall i, \exists! j$ / match-p(S_i, P_j)=t
- $\forall j, \exists i$ / match-p(S_i, P_j)=t
- $\forall i, i', \forall j, j'$
  match-p(S_i, P_j)=t
  and match-p($S_{i'}$, $P_{j'}$)=t
  $\Leftrightarrow$ d(binding(S_i, P_j), binding($S_{i'}$, $P_{j'}$))=0

Figure 2: Beam matching definition

with d a distance on bindings defined as follows:

Let $v_{j,k}$, $1<k\leq l$, be the variables used in pattern P_j.

d(binding(S_i, P_j),
binding($S_{i'}$, $P_{j'}$))=0
$\Leftrightarrow$ $\forall k$, $1<k<l$,
coverage($v_{j,k}$)=coverage($v_{j',k}$)
and
coverage($v_{j,l}$)=coverage($v_{j',l}$)
or prefix-p($v_{j,l}$, $v_{j',l}$)
or prefix-p($v_{j',l}$, $v_{j,l}$)

Figure 3: Matching distance definition

The coverage of a variable is the projection of the leaves of the tree this variable represents.

Thus, the distance between two bindings is null if and only if the coverage of each variable, except the last one, is the same in each binding. For the last variable of the patterns, if the coverage is not the same, one coverage has to be a prefix of the other.

In practice, the beam matching is realised by the method **match-beam** (fig. 4). The input parameters for this method are a pattern-beam (self) and a structure called a numbered analysis list (na_list). A numbered analysis list is a list of couples: ((number solutions)+). The clarification allows the user to indirectly select a number.

```
(defmethod match-beam
                          ((self pattern-beam) na_list)
  (let* (  (the_beam_name (beam-name self))
           (the_pattern_list (beam-value self))
           ...
           (the_fill_in_result (fill-the-matrix self na_list))
           (the_fill_in_success (car the_fill_in_result))
           (the_filled_matrix (cadr the_fill_in_result)))
    (if the_fill_in_success
      (let* (  (the_reduced_list ...)
               (the_normalized_list ...)
               (the_named_binding_list ...)
               (the_new_solution_sets ...) )
        (list  t
               the_beam_name
               the_named_binding_list
               the_new_solution_sets))
      '(nil nil nil nil))))
```

Figure 4: The method match-beam

The first step is the filling of a matching matrix **M**:

Solutions \ patterns	P_1	...	P_b
S_1			
...			
S_s			

while verifying the constraints listed in figure 2. If the_fill_in_success is true then the bindings are reduced so as to obtain the smallest coverage for each variable in each binding. The colums are first projected and reduced so as to obtain the following list:

((matched-solutions $Pattern_p$-reduced-binding)+)

The last variable in each reduced binding is then reduced so that a **the_reduced_list** is produced.

Finally a list is constructed (**the_new_solution_sets**) containing one new numbered analysis list per pattern. These new numbered analysis lists will be used as new input to the clarification scheduler to prepare further questions.

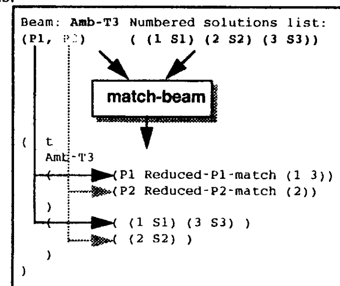


Figure 5: match-beam *input and output*

2.3. The presentation module

For a given clarification module, the clarification planner produces a question tree defined as follows:

Clarif_tree ::= (Clarification_question (Clarif_tree*))

A clarification question is defined as follows:

```
(defclass clarification-question-class ()
  ( (question-language
      :initarg    :question-language
      :accessor   question-language
      :documentation "the language of the question")
    (question-type
       :initarg    :question-type
       :accessor   question-type
       :documentation "type of the ambiguity")
    (question-modality
       :initarg    :question-modality
       :accessor   question-modality
       :documentation "modality of the question")
    (ambiguous-item
         :initarg    :ambiguous-item
         :accessor   ambiguous-item
         :documentation "the ambiguous utterance")
    (question-items-list
         :initarg    :question-items-list
         :accessor   question-items-list
         :documentation "(item solutions-number")))
```

Figure 6: The class clarification-question-class

Once a question has been answered by the selection of the item (choice) ***n***, if necessary, the next ***nth*** question tree has to be presented as shown in figure 7.

```
(defun question-tree-presentation
                              (the_question_tree)
  (if (= 1 (length the_question_tree))
    (concerned-solution (first the_question_tree))
    (let (  (the_choice
                (ask-question (first the_question_tree)))
            (the_other_questions
                (second the_question_tree)))
        (question-tree-presentation
                    (nth   (- the_choice 1)
                           the_other_questions)))))
```

Figure 7: The function question-tree-presentation

The method ***ask-question*** (cf. fig. 7) — specialized on the question-language, the question-type, and the question-modality — proposes the question to the user.

2.4. The operators

Ten families of operators have been defined. These operators are used to describe some manipulation of the bindings in order to produce the dialogue items (cf. fig. 20). Here is an example of some of them:

`Text` produce the text of the linguistic trees given as parameter.

`Coord` produce the coordinating occurrence of the linguistic trees given as parameter.

`But_Coord` produce the text of the linguistic trees given as parameter without the coordinating occurrence.

`Substitute` replace an ambiguous preposition with a non-ambiguous one (in the context) according to several properties: syntactic function or logico-semantic relation.

3. The linguistic data

The linguistic data are used to describe an instance of a clarification module for a given language. The data consist of patterns and pattern beams, an automaton, dialogue item production methods, and dialogue classes

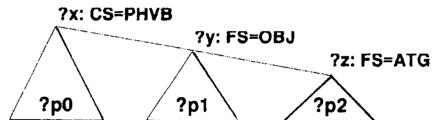
3.1. The patterns & pattern beams

A pattern is made of a pattern-name (the name of the pattern), a pattern-value (the definiton of the pattern) and a pattern-method (the method to be applied to the binding in order to produce a dialogue item). Figures 8 and 9 describe the two **patterns** used in the the definition of the **pattern-beam** called: *phvbprepatt_set_1* (fig. 10).

```
(defvar *phvbprepatt-t1-1*
  (make-instance 'pattern
    :pattern-name '*phvbprepatt-t1-1*
    :pattern-value
      '(  (?is ?x node-prop-equal-p 'CS 'PHVB)
          (?+ ?p0)
          (   (?is ?y node-prop-equal-p 'FS 'OBJ)
              (?+ ?p1)
              (   (?is ?z node-prop-equal-p 'FS 'ATG)
                  (?+ ?p2))))
    :pattern-method #'item-production-method))
```

Figure 8: The variable *phvbprepatt-t1-1*

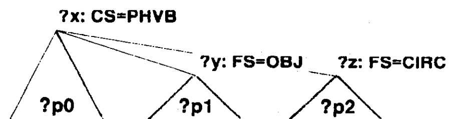
A graphical interpretation of the previous pattern is given below:




```
(defvar *phvbprepatt-t1-2*
  (make-instance 'pattern
    :pattern-name '*phvbprepatt-t1-2*
    :pattern-value
      '(  (?is ?x node-prop-equal-p 'CS 'PHVB)
          (?+ ?p0)
          (   (?is ?y node-prop-equal-p 'FS 'OBJ)
              (?+ ?p1))
          (   (?is ?z node-prop-equal-p 'FS 'CIRC)
              (?+ ?p2)))
    :pattern-method #'item-production-method))
```

Figure 9: The variable *phvbprepatt-t1-2*

A graphical interpretation of the previous pattern is given below:



A pattern beam is made of a beam-name (the name of the beam), and a beam-value. The beam-value is a list of patterns composing the set. The order in which the patterns appear in the list determines the order in which they are matched while filling the matching matrix. This order also determines the order of the question's items; for a recognized beam, pattern $_i$ gives rise to the production of item $_i$ of the dialogue.

The *phvbprepatt_set_1* beam (fig. 10) is one of the two beams used to describe an ambiguity called "verbal phrase prepositional attachment". The other beam is called *phvbprepatt_set_2*. These two beams are used to describe a **beam-stack** called **phvbprepatt_beam_stack** (fig. 11) that will be used by the state of the clarification scheduler in charge to check the presence of a phvbprepatt ambiguity.

```
(defvar *phvbprepatt_set_1*
  (make-instance 'pattern-beam
    :beam-name '*phvbprepatt_set_1*
    :beam-value (list *phvbprepatt-t1-1*
                      *phvbprepatt-t1-2*)))
```

Figure 10: The variable *phvbprepatt_set_1*

A beam-stack (fig. 11) combines all the beams proposed to define a class of ambiguity. A beam-stack is associated with each one of the relevant ambiguity recognition states of the clarification scheduler. When trying to recognize a class of ambiguity, each beam is matched one after another until a beam has been matched or until every beam has been matched without success.

```
(defvar *phvbprepatt_beam_stack*
  (make-instance 'beam-stack
    :beam-stack-name '*phvbprepatt_beam_stack*
    :beam-stack-value (list *phvbprepatt_set_1*
                            *phvbprepatt_set_2*)))
```

Figure 11: The variable *phvbprepatt_beam_stack*

3.2. The clarification scheduler

The clarification scheduler is an automaton made of three kinds of states: an automaton-scheduler, meta-class recognition states, and ambiguity-class recognition states.

The automaton-sheduler (fig. 12) is the entry point of the clarification scheduler of every clarification module. It is defined as a method with one parameter specialized on the_language. The automaton-scheduler shown in figure 12, is the automaton-scheduler for the English clarification scheduler.

```
(defmethod automaton-scheduler
                    ( (the_language (eql 'english))
                      the_sentence
                      the_numbered_analysis_list)
 " in: a list of indexed-solution-sets,
  out: a question"
 (if (= 1 (length the_numbered_analysis_list))
   (list (make-instance 'empty-question
          :concerned-solution
            (first (first the_numbered_analysis_list))
          :concerned-tree
            (second (first the_numbered_analysis_list))))
   (same-cat-p-state the_language
                     the_sentence
                     the_numbered_analysis_list)))
```

Figure 12: The method automaton-scheduler

There are two meta-class recognition states: one to test the presence of an ambiguity of syntactic labelling, and one to test the presence of an ambiguity of geometry. If neither of these ambiguity meta classes have been recognized, the ambiguity is a decorational ambiguity.

There are two kinds of ambiguity-class recognition states: ones using the beam matching mechanism (for the ambiguities described with beams), and ones using a property recognition mechanism (for the ambiguities not described with beams). We will focus on the first kind of states.

The ambiguity class recognition states are described by methods sharing a common skeleton; if the ambiguity the state is to recognize is recognized, a question tree is prepared; if not, the next ambiguity class recognition state is triggered.

In the **phvb-prep-att-state** (fig. 13), if a phvb-prep-att ambiguity is not recognized, the next triggered state is the ***rel-phvb-adv-att-state*** one. If the ambiguity is recognized, a question tree is prepared with the method **prepare-question-tree.**

```
(defmethod phvb-prep-att-state
                    ( (the_language (eql 'english))
                      the_sentence
                      the_na_list)
  (let*
   ( (the_beam_stack *phvbprepatt_beam_stack*)
     (the_beam_match
       (beam-stack-match the_beam_stack
                         the_na_list))
     (matched? (first the_beam_stack_match)))
   (if matched?
    (let
     ( (the_type 'general)
       (the_modality 'textual)
       (the_list_of_triplets (third the_beam_match))
       (the_new_sol_sets (fourth the_beam_match)))
     (prepare-question-tree the_language
                            the_type
                            the_modality
                            the_sentence
                            the_triplets
                            the_new_sol_sets))
    (rel-phvb-adv-att-state the_language
                            the_sentence
                            the_na_list))))
```

Figure 13: The method phvb-prep-att-state

The method **prepare-question-tree** (fig. 14) prepares a question with the third item (*the_list-of_triplets*, fig. 13 & 14) of the result of the beam-match method (cf. fig. 5).

The new indexed solutions sets in the_na_list are used to prepare, if necessary, the next questions. Finally, a question tree is constructed (the_result).

```
(defmethod prepare-question-tree
                    ( (the_language (eql 'english))
                      the_type
                      (the_modality (eql 'textual))
                      the_sentence
                      the_list_of_triplets
                      the_new_solution_sets)
 (let*
  ( (the_first_question   (prepare-question
                              the_language
                              the_type
                              the_modality
                              the_sentence
                              the_list_of_triplets))
    (the_next_questions   (prepare-question-list
                              the_language
                              the_sentence
                              the_new_sol_sets))
    (the_result (list the_first_question
                      the_next_questions)))
  the_result))
```

Figure 14: The method prepare-question-tree

The method **prepare-question-list** (fig. 15) calls the automaton-scheduler for each new indexed solutions set (the_new_sol_sets, cf. fig. 15).

```
(defmethod prepare-question-list
                    (    (the_language (eql 'english))
                          the_sentence
                          the_sol-sets)
  (if (= 1 (length the_sol-sets))
    (list (automaton-scheduler the_language
                               the_sentence
                               (car the_new_sol_sets))
    (cons (automaton-scheduler the_language
                               the_sentence
                               (car the_new_sol_sets))
          (prepare-question-list the_language
                               the_sentence
                               (cdr the_new_sol_sets)))))
```

Figure 15: The method prepare-question-list

The creation process of the question tree is resumed in the following figure.

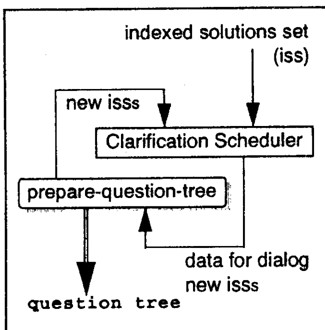


Figure 16: The construction of the `question tree`

3.3. The dialogue item production methods

The dialogue item production methods are described with the method **item-production-method** (fig. 17 & 18) specialized on its pattern-name (the name of the pattern) argument.

Each method produces a string of characters which is an arrangement of a manipulation, with the operators (defined in § 2.4) of the binding associated with some of the variables defined in the pattern the method is associated with.

The method associated with pattern *phvbprepatt-t1-1* (fig. 18) produces the following string:

text(?p0) **(** text(?p1) text(?p2) **)**

```
(defmethod item-production-method
    ((pattern-name (eql '*phvbprepatt-t1-1*)) binding)
  (format nil
          "~A (~A ~A)."
          (apply #'text (cdr (assoc '?p0 binding)))
          (apply #'text (cdr (assoc '?p1 binding)))
          (apply #'text (cdr (assoc '?p2 binding)))))
```

Figure 17: The method item-production-method

The method associated with pattern *phvbprepatt-t1-2* (fig. 19) produces the following string:

text(?p2)**,** text(?p0) text(?p1)

```
(defmethod item-production-method
    ((pattern-name (eql '*phvbprepatt-t1-2*)) binding)
  (format nil
          "~A, ~A ~A."
          (apply #'text (cdr (assoc '?p2 binding)))
          (apply #'text (cdr (assoc '?p0 binding)))
          (apply #'text (cdr (assoc '?p1 binding)))))
```

Figure 18: The method item-production-method

3.4. The dialogue classes

Generic dialogue classes are specialized with dialogue subclasses. For English textual dialogue we have defined two classes: english-general-textual-dialogue-class (fig. 19), and english-polysemy-textual-dialogue-class.

These classes specialize the invitation-string, the prompt-string, the window-title (shown fig. 20), and other slots of the generic-textual-clarif-dialogue-class (cf. fig. 19).

```
(defclass english-general-textual-dialogue-class
                      (generic-textual-clarif-dialogue-class)
  (  (window-length   :initform 400)
     (invitation-string   :initform "The following
        sentence has several possible interpretations.")
     (invitation-string-font   :initform '("geneva" 10))
     (ambiguous-string-font   :initform '("geneva" 10 :bold))
     (prompt-string   :initform "Choose the right one:")
     (prompt-string-font   :initform '("geneva" 10))
     (items-font  :initform '("geneva" 10 :bold)))
  (:default-initargs :window-title "Ambiguity"))
```

Figure 19: The class english-general-textual-dialogue-class

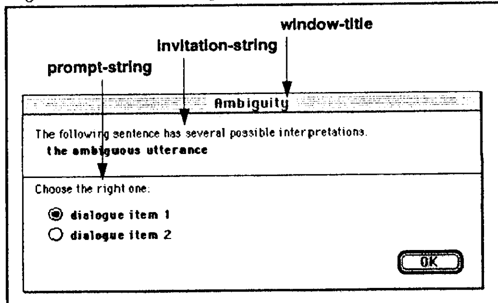


Figure 20: Some dialogues' slots

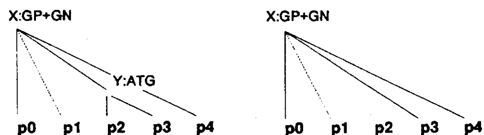
4. Some examples

In this last section, let us give some examples of the produced dialogues according to the ambiguity to be solved.

4.1. Syntactic class ambiguity

In the sentence "This is an English speaking agent." there is ambiguity for the syntactic class of the word "English" which can interpreted as a noun (an agent who speaks English), or as an adjective (an agent who is English).

This ambiguity is detected by the following pattern beam.



The dialogue produced to solve this ambiguity is shown in the following figure.

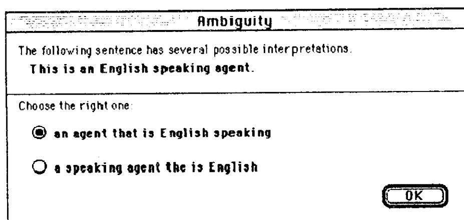


Figure 21: Syntactic class clarification

4.2. Prepositional attachment involving the verb

In the sentence "Where can I catch a taxi from Kyoto station?" there is an ambiguity called phvb-prep-att (prepositional attachment involving the verb).

This ambiguity is detected with the beam shown in fig. 10.

The dialogue produced to solve this ambiguity is shown in the following figure.

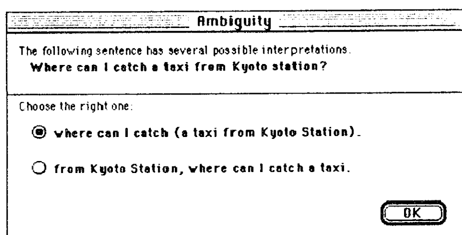
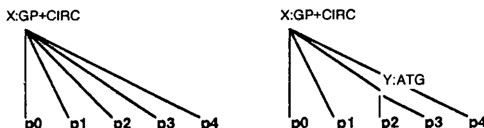


Figure 22: phvb-prep-att type-1 clarification

4.3. Prepositional attachment not involving the verb

In the sentence "You are going to the international conference center." there is an ambiguity called non-phvp-prep-att (prepositional attachment not involving the verb).

This ambiguity is detected by the following beam.



The dialogue produced to solve this ambiguity is shown in the following figure.

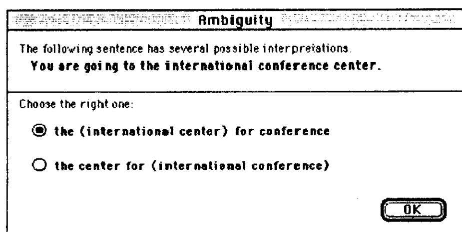


Figure 23: non-phvb-prep-att type-1 clarification

Conclusion

The methodology we described has in our opinion two main advantages: it can be customized and it can be improved incrementally. The ability to customize comes from the clear separation of the linguistic data from the kernel. In this framework, a number of different clarification modules can be produced for different languages and kinds of input. The descripition of the linguistic data can be improved incrementally as the design and the use of a clarification module progress.

The coverage of the first version of an interactive clarification module described here is currently being evaluated. For this evaluation we are constructing a new corpus from the data collected in the latest experiments conducted in the EMMI framework of interpreting multimedia /multimodal and telephone communications, [Park & Loken-Kim 1994] and [Park, *et al.* 1995].

Most of the ambiguities we have found in the evaluation and improvement corpus are already covered by the current module. The most important difference lies in the fact that there are a greater number of ambiguities of coordination. After we evaluate the first version of the clarification module by testing its coverage on the new data, we will improve the module by extending it to include the ambiguities it was not able to handle in the test data. Certainly, we will not be able to claim that the improved module will cover all the ambiguities found in spontaneous English, but it will have broad coverage for application to these particular domains. Similarly, if new ambiguities are located in future data, the module can be incrementally improved to cope with the new ambiguities.

We are also currently investigating the use of weights to let the module learn from the history of the dialogue. If one particular interpretation of an often recurring ambiguity is always chosen, the module will more heavily weight that interpretation, either automatically, or after querying the user. The module will then be tunable.

In future work, we intend to integrate the clarification module into the EMMI context to investigate its usability in a multimedia environment. We will also design experiments with naive users to determine the optimal design of clarification dialogues and interactive sessions.

References

Blanchon H. (1994a). *Pattern-based approach to interactive disambiguation: first definition and implementation.* Rap. ATR-Interpreting Telecommunications Research Laboratories. Technical Report. n° TR-IT-0073. Sept. 8, 1994. 91 p.

Blanchon H. (1994b). *Perspectives of DBMT for monolingual authors on the basis of LIDIA-1, an implemented mock-up.* Proc. Coling-94. Kyoto, Japan. August 5-9, 1994, vol. **1/2** : pp. 115-119.

Blanchon H. & Loken-Kim K. H. (1994). *Towards More Robust, Fault-Tolerant and User-Friendly Software Integrating Natural Language Processing Components.* in Bulletin of the Information Processing Society of Japan. vol. **94**(109) : pp. 17-24.

Boitet C. & Blanchon H. (1995). *Multilingual Dialogue-Based MT for monolingual authors: the LIDIA project and a first mockup.* in Machine Translation. vol. **À paraître** : pp. 21.

Caelen J. (1994). *Multimodal Human-Computer Interaction.* in Fundamentals of Speech Synthesis and Speech Recognition. Keller, E. (ed.). John Wiley & Sons. New York. pp. 339-373.

Fais L. (1994). *Effects of communicative mode on spontaneous English speech.* Rap. Institute of Electronics, Information and Communication Engineers. Technical Report. n° NLC94-22. Oct. 94. 6 p.

Goddeau D., Brill E., Glass J., Pao C., Philips M., Polifroni J., Seneff S. & Zue V. (1994). *GALAXY: a Human-Language Interface to On-Line Travel Information.* Proc. ICSLP 94. Yokohama, Japan. September 18-22, 1994, vol. **2/4** : pp. 707-710.

Haddock N. J. (1992). *Multimodal Database Query.* Proc. Coling-92. Nantes, France. 23-28 juillet 1992, vol. **4/4** : pp. 1274-1278.

Hiyoshi M. & Shimazu H. (1994). *Drawing Pictures with Natural Language and Direct Manipulation.* Proc. Coling-94. Kyoto, Japan. August 5-9, 1994, vol. **2/2** : pp. 722-726.

Kay M., Gawron J. M. & Norvig P. (1994). *Verbmobil: A Translation System for Face-to-Face Dialog. CSLI lecture note no 33.* Center for the Study of Language and Information, Stanford, CA. 235 p.

Keene S. E. (1989). *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS.* Addison-Wesley Publishing Compagny. New York. 266 p.

Loken-Kim K.-H., Yato F., Kurihara K., Fais L. & Furukawa R. (1993). *EMMI-ATR environment for multi-modal interactions.* Rap. ATR-ITL. Technical Report. n° TR-IT-0018. Sept 30, 1993. 28 p.

Morimoto T., Suzuki M., Takezawa T., Kikui G., Nagata M. & Tomokio M. (1992). *A Spoken Language Translation System: SL-TRANS2.* Proc. Coling-92. Nantes, France. 23-28 juillet 1992, vol. **3/4** : pp. 1048-1052.

Nishimoto T., Shida N., Kobayashi T. & Shirai K. (1994). *Multimodal Drawing Tool Using Speech, Mouse and Key-Board.* Proc. ICSLP 94. Yokohama, Japan. September 18-22, 1994, vol. **3/4** : pp. 1287-1290.

Norvig P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp.* Morgan Kaufmann Publishers. San Mateo, California. 945 p.

Park Y.-D. & Loken-Kim K.-H. (1994). *Text Database of Telephone and Multimedia Multimodal Interpretation Experiment.* Rap. ATR-ITL. Technical Report. n° TR-IT-0086. Dec. xx, 1994. XX p.

Park Y.-D., Loken-Kim K.-H., Mizunashi S. & Fais L. (1995). *Transcription of the Collected Dialogue in a Telephone and Multimedia/Multimodal WOZ Experiment.* Rap. ATR-ITL. Technical Report. n° TR-IT-0090. Feb., 1995. 123 p.

Zue V., Seneff S., Polifroni J. & Phillips M. (1993). *PEGASUS: a Spoken Dialogue Interface for On-Line Air Travel Planing.* Proc. ISSD-93 — New Directions in Human and Man-Machine Communication. International Conference Center, Waseda University, Tokyo, Japan. November 10-12, 1993, vol. **1/1** : pp. 157-160.