Language Model and Sentence Structure Manipulations for Natural Language Application Systems

Zenshiro Kawasaki, Keiji Takida, and Masato Tajima
Department of Intellectual Information Systems
Toyama University
3190 Gofuku, Toyama 930-0887, Japan
{kawasaki, takida, and tajima}@ecs.toyama-u.ac.jp

Abstract

This paper presents a language model and its application to sentence structure manipulations for various natural language applications including human-computer communications. Building a working natural language dialog systems requires the integration of solutions to many of the important subproblems of natural language processing. In order to materialize any of these subproblems, handling of natural language expressions plays a central role; natural language manipulation facilities are indispensable for any natural language dialog systems. Concept Compound Manipulation Language (CCML) proposed in this paper is intended to provide a practical means to manipulate sentences by means of formal uniform operations.

1 Introduction

Sentence structure manipulation facilities such as transformation, substitution, translation, etc., are indispensable for developing and maintaining natural language application systems in which language structure operation plays an essential role. For this reason structural manipulability is one of the most important factors to be considered for designing a sentence structure representation scheme, i.e., a language model. The situation can be compared to database management systems; each system is based on a specific data model, and a data manipulation sublanguage designed for the data model is provided to handle the data structure (Date, 1990).

In Concept Coupling Model (CCM) proposed in this paper, the primitive building block is a Concept Frame (CF), which is defined for each phrasal or sentential conceptual unit. The sentence analysis is carried out as a CF instantiation process, in which several CFs are combined to form a Concept Compound (CC), a nested relational structure in which the syntactic and semantic properties of the sentence are encoded. The simplicity and uniformity of the CC representation format lead to a corresponding simplicity and uniformity in the CC structure operation scheme, i.e., CC Manipulation Language (CCML).

Another advantage of the CCM formalism is that it allows inferential facilities to provide flexible human computer interactions for various natural language applications. For this purpose conceptual relationships including synonymous and implicational relations established among CFs are employed. Such knowledge-based operations are under development and will not be discussed in this paper.

In Section 2 we present the basic components of CCM, i.e., the concept frame and the concept compound. Section 3 introduces the CC manipulation language; the major features of each manipulation statement are explained with illustrative examples. Concluding observations are drawn in Section 4.

2 Concept Coupling Model

2.1 Concept Compound and Concept Frame

It is assumed that each linguistic expression such as a sentence or a phrase is mapped onto an abstract data structure called a concept compound (CC) which encodes the syntactic and semantic information corresponding to the linguistic expression in question. The CC is realized as an instance of a data structure called the concept frame (CF) which is defined for each conceptual unit, such as an entity, a property, a relation, or a proposition, and serves as a template for permissible CC structures. CFs are distinguished from one another by the syntactic and semantic properties of the concepts they represent, and are assigned unique identifiers. CFs are classified according to their syntactic categories as sentential, nominal, adjectival, and adverbial. The CCM lexicon is a set of CFs; each entry of the lexicon defines a CF. It should be noted that in this paper inflectional information attached to each CF definition is left out for simplicity.

Zenshiro Kawasaki, Keiji Takida and Masato Tajima (1998) **Language Model and Sentence Structure Manipulations for Natural Language Applications Systems.** In D.M.W. Powers (ed.) *NeMLaP3/CoNLL98 Workshop on Human Computer Conversation,* ACL, pp 281-286.

2.2 Syntax

In this section we define the syntax of the formal description scheme for the CF and the CC, and explain how it is interpreted. A CF is comprised of four types of tokens. The first is the concept identifier which is used to indicate the relation name of the CF structure. The second token is the key-phrase, which establishes the links between the CF and the actual linguistic expressions. The third is a list of attribute values which characterize the syntactic and semantic properties of the CF. Control codes for the CCM processing system may also be included in the list. The last token is the concept pattern which is a syntactic template to be matched to linguistic expressions. The overall structure of the CF is defined as follows:

(1) $C(K, A, P)$,

where C and K are the concept identifier and the key-phrase respectively, A represents a list of attribute values of the concept, and P is the concept pattern which is a sequence of several terms: variables, constants, and the symbol * which represents the key-phrase itself or one of its derivative expressions. The constant term is a word string. The variable term is accompanied by a set of codes which represent the syntactic and semantic properties imposed on a CF to be substituted for it. These codes, each of which is preceded by the symbol +, are classified into three categories: (a) constraints, (b) roles, and (c) instruction codes to be used by the CCM processing system. No reference is made to the sequencing of these codes, i.e., the code names are uniquely defined in the whole CCM code system.

The CF associated with the word *break* in the sense meant by *John broke the box yesterday* is shown in (2):

(2) $break010('break', [sent, dyn, base],$
$\quad '\$1(+nomphrs + hum + subj + agent) \; *$
$\quad\quad \$2(+nomphrs + cncrt + obj + patnt)').$

In this example the identifier and the key-phrase are *break*010 and *break* respectively. The attribute list indicates that the syntactic category of this CF is *sent(ential)* and the semantic feature is *dyn(amic)*. The attribute *base* is a control code for the CCM processing system, which will not be discussed further in this paper. The concept pattern of this CF corresponds to a subcategorization frame of the verb *break*. Besides the symbol * which represents the key-phrase *break* or one of its derivatives, the pattern includes two variable terms ($1 and $2), which are called the immediate constituents of the concept *break*010. The appended attributes to these variables impose conditions on the CFs substituted for them. For example, the first variable should be matched to a CF which is a *nom(inal-)phr(a)s(e)* with the semantic feature *hum(an)*, and the syntactic role *subj(ect)* and the semantic role *agent* are to be assigned to the instance of this variable.

The CC is an instantiated CF and is defined as shown in (3):

(3) $C(H, R, A, [X1, X2, ..., Xn, M])$,

where the concept identifier C is used to indicate the root node of the CC and represents the whole CC structure (3), and H, R, and A are the head, role, and attribute slot respectively. The head slot H is occupied by the identifier of the C's head, i.e., C itself or the identifier of the C's component which determines the essential properties of the C. The role slot R, which is absent in the corresponding CF definition, is filled in by a list of syntactic and semantic role names which are to be assigned to C in the concept coupling process described in Section 2.3. The last slot represents the C's structure, an instance of the concept pattern P of the corresponding CF, and is occupied by the constituent list. The members of the list, $X1$,$X2$,..., and Xn, are the CCs corresponding to the immediate constituents of C. The tail element M of the constituent list, which is absent in non-sentential CCs, has the form $md\_(H, R, A, [M1, ..., Mm])$, where $M1$,...,Mm represent CCs which are associated with optional adverbial modifiers.

By way of example, the concept structure corresponding to the sentence in (4a) is shown in (4b), which is an instance of the CF in (2).

(4a) *John broke the box yesterday.*

```
(4b) break010
       (break010,
        [],
        [sent, dyn, fntcls, past, agr3s],
        [john0060
          (john0060,
           [subj, agent],
           [nomphrs, prop, hum, agr3s, mascln],
           []),
         box00010
          (box00010,
           [obj, patnt],
           [the_, cncrt, nomphrs, agr3s],
           []),
         md_
          ([yeste010],
           [modfr],
           [advphrs, mod],
           [yeste010
             (yeste010,
              [],
              [advphrs, timeAdv],
              []) ]) ]).
```

In (4b) three additional attributes, i.e., *f(i)n(i)t(e-)cl(au)s(e)*, *past*, and *agr(eement-)3(rd-person-)s(ingular)*, which are absent in the CF definition, enter the attribute list of *break*010. Also note that the constituent list of *break*010 contains an optional modifier component with the identifier *md_*, which does not have its counterpart in the corresponding CF definition given in (2).

2.3 Concept Coupling

As sketched in the last section, each linguistic expression such as a sentence or a phrase is mapped onto an instance of a CF. For example, the sentence (4a) is mapped onto the CC given in (4b) which is an instance of the sentential CF defined in (2). In this instantiation process, three other CFs given in (5) are identified and coupled with the CF in (2) to generate the compound given in (4b).

(5a) *john*0060('*john*',
[*nomphrs*, *prop*, *hum*, *agr3s*, *mascln*], ' * ').

(5b) *box*00010('*box*', [*nomphrs*, *cncrt*, *base_n*], ' * ').

(5c) *yeste*010('*yesterday*', [*advphrs*, *timeAdv*], ' * ').

All three CFs in (5) are primitive CFs, i.e., their concept patterns do not contain variable terms and their instances constitute ultimate constituents in the CC structure. For example (5b) defines a CF corresponding to the word *box*. The identifier and the key-phrase are *box*00010 and *box* respectively. The attribute list indicates that the syntactic category, the semantic feature, and the control attribute are *nom(inal-)phr(a)s(e)*, *c(o)ncr(e)t(e)*, and *base(-)n(oun)* respectively. The concept pattern consists of the symbol *, for which *box* or *boxes* is to be substituted.

In the current implementation of concept coupling, a Definite Clause Grammar (DCG) (Pereira and Warren, 1980) rule generator has been developed. The generator converts the run-time dictionary entries, which are retrieved from the base dictionary as the relevant CFs for the input sentence analysis, to the corresponding DCG rules. We shall not, however, go into details here about the algorithm of this rule generation process. The input sentence is then analyzed using the generated DCG rules, and finally the source sentence structure is obtained as a CC, i.e., an instantiated CF. In this way the sentence analysis can be regarded as a process of identifying and combining the CFs which frame the source sentence.

3 Concept Compound Manipulations

The significance of the CC representation format is it's simplicity and uniformity; the relational structure has the fixed argument configuration, and every constituent of the structure has the same data structure. Sentence-to-CC conversion corresponds to sentence analysis, and the obtained CC encodes syntactic and semantic information of the sentence; the CC representation can be used as a model for sentence analysis. Since CC, together with the relevant CFs, contains sufficient information to generate a syntactically and semantically equivalent sentence to the original, the CC representation can also be employed as a model for sentence generation. In this way, the CC representation can be used as a language model for sentence analysis and generation.

Another important feature of the CC representation is that structural transformation relations can easily be established between CCs with different syntactic and semantic properties in tense, voice, modality, and so forth. Accordingly, if a convenient CC structure manipulation tool is available, sentence-to-sentence transformations can be realized through CC-to-CC transformations. The simplicity and uniformity of the CC data structure allows us to devise such a tool. We call the tool Concept Compound Manipulation Language (CCML).

Suppose a set of sentences are collected for a specific natural language application such as second language learning or human computer communication. The sentences are first transformed into the corresponding CCs and stored in a CC-base, a file of stored CCs. The CC-base is then made to be available to retrieval and update operations.

The CCML operations are classified into three categories: (a) Sentence-CC conversion operations, (b) CC internal structure operations, (c) CC-base operations. The sentence-CC conversion operations consists of two operators: the sentence-to-CC conversion which invokes the sentence analysis program and parses the input sentence to obtain the corresponding CC as the output, and the CC-to-sentence conversion which generates a sentence corresponding to the indicated CC. The CC internal structure operations are concerned with operations such as modifying values in a specific slot of a CC, and transforming a CC to its derivative CC structures. The CC-base operations include such operations as creating and destroying CC-bases, and retrieving and updating entries in a CC-base. The current implementation of these facilities are realized in a Prolog environment, in which these operations are provided as Prolog predicates.

In the following sections, the operations mentioned above are explained in terms of their effects on CCs and CC-bases, and are illustrated by means of a series of examples. All examples will be based on a small collection of sample sentences shown in (7), which are assumed to be stored in a file named *sophie.text*.

(7a) *Sophie opened the big envelope apprehensively.*

(7b) *Hilde began to describe her plan.*

(7c) *Sophie saw that the philosopher was right.*

(7d) *A thought had suddenly struck her.*

3.1 Sentence-CC Conversions

Two operations, *$get_cc* and *$get_sent*, are provided to inter-convert sentences and CCs.

$get_cc, $get_sent

The conversion of a sentence to its CC can be realized by the operation *$get_cc* as a process of concept coupling described in Section 2.3. The reverse process, i.e., CC-to-sentence conversion, is carried out by the operation *$get_sent*, which invokes the sentence generator to transform the CC to a corresponding linguistic expression. The formats of these operations are:

(8) *$get_cc(Sent, CC)*.

(9) *$get_sent(CC, Sent)*.

The arguments *Sent* and *CC* represent a sentence or a list of sentences, and a CC or a list of CCs, respectively. For the *$get_cc* operation, the input sentence (list) occupies the *Sent* position and *CC* is a variable in which the resultant CC (list) is obtained. For the *$get_sent* operation, the roles of the arguments are reversed, i.e., *CC* is taken up by an input CC (list) and *Sent* an output sentence (list).

Example:

(10a) Get CC for the sentence *Sophie opened the big envelope apprehensively*.

The query (10a) is translated into a CCML statement as:

(10b) *$get_cc('Sophie opened the big envelope apprehensively', CC)*.

Note that the input sentence must be enclosed in single quotes. The CC of the input sentence is obtained in the second argument *CC*, as shown in (10c):

```
(10c) CC =
open0010(open0010, [], [sent, fntcls, past, agr3s],
    [sophi010(sophi010, [subj],
            [nomphrs, prop, hum, agr3s, femnn], []),
     big00020(envel001, [obj],
            [the_, det_modfd, adj_mod, cncrt,
                                  nomphrs, agr3s],
         [envel001(envel001, [],
                  [cncrt, nomphrs, agr3s, f_n], [])]),
     md_([appre010], [modfr], [advphrs, mod],
            [appre010(appre010, [], [advphrs], [])])]).
```

3.2 CC Internal Structure Operations

Since the CC is designed to represent an abstract sentence structure in a uniform format, well-defined attributive and structural correspondences can be established between CCs of syntactically and semantically related sentences. Transformations between these derivative expressions can therefore be realized by modifying relevant portions of the CC in question.

For manipulating the CC's internal structure, CCML provides four basic operations (*$add, $delete, $substitute, $restructure*) and one comprehensive operation (*$transform*).

$add

This operation is used to add values to a slot. The format is:

(11) *$add(CC, Slot, ValueList, CCNew)*.

For the CC given in the first argument *CC*, the elements in *ValueList* are appended to the slot indicated by the second argument *Slot* to get the modified CC in the last argument *CCNew*.

Example:

(12a) For the CC given in (10c), add the value *perf(e)ct* to the slot *attribute*.

(12b) *$add(CC, attribute, [perfct], CCNew)*.

In (12b) the first argument *CC* is occupied by the CC shown in (10c). The last argument *CCNew* is then instantiated as the CC corresponding to the sentence *Sophie had opened the envelope apprehensively*. Note that *imperf(e)ct* is a default attribute value assumed in (10c).

$delete

In contrast to *add*, this operation removes the indicated values from the specified slot. The format is:

(13) *$delete(CC, Slot, ValueList, CCNew)*.

$substitute

This operation is used to replace a value in a slot with another value. The format is:

(14) *$substitute(CC, Slot, OldValue, NewValue, CCNew)*.

Example:

(15a) For the CC in (10c), replace the attribute value *past* by *pres(e)nt*.

(15b) *$substitute(CC, attribute, past, presnt, CCNew)*.

By this operation *CCNew* is instantiated as a CC corresponding to the sentence *Sophie opens the envelope apprehensively*.

$restructure

This operation changes the listing order of immediate constituents, i.e., the component CCs in the structure slot of the specified CC. The format is:

(16) *$restructure(CC, Order, CCNew)*,

where the first argument *CC* represents the CC to be restructured and the second argument *Order* defines the new ordering of the constituents. The general format for this entry is:

(17) $[p_1, p_2, p_3, .., p_n]$,

where the integer p_i $(i = 1, 2, ..., n)$ represents the old position for the p_i-th constituent of the CC in question, and the current position of p_i in the list (17) indicates the new position for that constituent. For example, $[2, 1]$ means that the constituent CCs in the first and second positions in the old structure are to be swapped. The remaining constituents are not affected by this rearrangement.

$transform

The above mentioned basic operations yield CCs which do not necessarily correspond to actual (grammatical) linguistic expressions. The higher level structural transformation operation, *$transform*, is a comprehensive operation to change a CC into one of its derivative CCs which directly correspond to actual linguistic expressions. Tense, aspect, voice, sentence types (statement, question, etc), and negation are among the currently implemented transformation types. The format is:

(18) *$transform(CC, TransTypeList, CCNew)*.

The second argument *TransTypeList* defines the target transformation types which can be selected from the following codes:

Voice: *act(i)v(e), pas(si)v(e)*.
Negation: *affirm(a)t(i)v(e), neg(a)t(i)v(e)*.
Tense: *pres(e)nt, past, fut(u)r(e)*.
Perfective: *perf(e)ct, imperf(e)ct*.
Progressive: *cont(inuous), n(on-)cont(inuous)*.
Sentence Type: *stat(e)m(e)nt, quest(io)n, dir(e)ct(i)v(e), excl(a)m(a)t(io)n*.

Note that the *$transform* operation does not require explicit indication of the attribute type for each transformation code in the above list. This is possible because the code names are uniquely defined in the whole CCM code system.

Examples:

(19a) Get the interrogative form of the sentence *Hilde began to describe her plan.*

The above query is expressed as a series of CCML statements:

(19b) *$get_cc('Hilde began to describe her plan', CC)*,
$transform(CC, [questn], CCNew),
$get_sent(CCNew, SentNew).

The result is obtained in *SentNew* as:

(19c) *SentNew='Did Hilde begin to describe her plan?'*

Note that the same values are substituted for like-named variables appearing in the same query in all of their occurrences, e.g., *CC* and *CCNew* in (19b).

Another example of the use of *$transform* is given in (20):

(20a) Get the present perfect passive form of the sentence *Sophie opened the big envelope apprehensively.*

(20b) *$get_cc('Sophie opened the big envelope apprehensively', CC)*,
$transform(CC, [presnt, perfct, pasv], CCNew),
$get_sent(CCNew, SentNew).

(20c) *SentNew='The big envelope has been opened by Sophie apprehensively.'*

3.3 CC-base Operations

3.3.1 Storage operations

The CC-base storage operations are: *$create_ccb*, *$activate_ccb*, *$destroy_ccb*, *$save_ccb*, and *$restore_ccb*.

$create_ccb

The general format of *$create_ccb* is:

(21) *$create_ccb(SentenceFileName, CCBaseFileName)*.

The file indicated by the first argument *SentenceFileName* contains a set of sentences (one sentence per line) to be converted to their CC structures. The *$create_ccb* operation invokes the sentence analyzer to transform each sentence in the file into the corresponding CC and store the results in the CC-base indicated by *CCBaseFileName* (one CC per line).

Example:

(22a) Convert all sentences in the text file *sophie.text* shown in (7) into their CCs and store the result in the CC-base named *sophie.ccb*.

(22b) *$create_ccb('sophie.text', 'sophie.ccb')*.

The first line of the file *sophie.ccb* is taken by the CC given in (10c) which corresponds to the first sentence in the file *sophie.text* shown in (7).

$activate_ccb

The format of *$activate_ccb* is:

(23) *$activate_ccb(CCBaseFileName)*.

This operation copies the CC-base indicated by *CCBaseFileName* to the "current" CC-base which can be accessed by retrieval and update operations explained in the following sections. If another CC-base file is subsequently "activated", CCs from this new CC-base file are simply appended to the current CC-base.

Example:

(24a) Activate *sophie.ccb*.

(24b) *$activate('sophie.ccb')*.

$destroy_ccb

There are two formats for *$destroy_ccb*:

(25a) *$destroy_ccb(CCBaseFileName)*.

(25b) *$destroy_ccb*.

CCBaseFileName is taken up by the name of a CC-base file to be removed. If *current* is substituted for *CCBaseFileName* in (25a) or the operation is used in the (25b) format, the current CC-base is removed.

$save_ccb

The formats are:

(26a) *$save_ccb(CCBaseFileName)*.

(26b) *$save_ccb*.

The *$save_ccb* operation is used to store the current CC-base into the file indicated by *CCBaseFileName*. The current CC-base is destroyed by this operation. If *CCBaseFileName* is *current* in (26a) or the operation is used in the (26b) format, the current CC-base is stored temporarily in the system's work space. Note that the *$save_ccb* operation discards already existing CC-base in the work space when it is executed.

$restore_ccb

This operation takes no arguments. The existing current CC-base is destroyed and the saved CC-base in the work space is activated. The format is:

(27) *$restore_ccb*.

3.3.2 Retrieval operations

Retrieval operations are applied to the current CC-base. Relevant CC-bases should therefore be activated prior to issuing retrieval operations.

$retrieve_cc

The *$retrieve_cc* operation searches the current CC-base for CCs which satisfy the specified conditions. Note that if a CC contains component CCs which satisfy the imposed conditions, they are also fetched by this operation; CCs within a CC are all searched for. The general format of *$retrieve_cc* is as fallows:

(28) *$retrieve_cc(SelectionalConditionList*,

RetrievedCCList),
where *SelectionalConditionList* is a list of constraints imposed on CCs to be retrieved. Each element of the list consists of either of the following terms:
(29a) *SlotName = ValueList.*
(29b) *RoleName : ConditionList.*
SlotName is occupied by a slot name of the CC structure, i.e., identifier, head, role, attribute, or structure. *ValueList* is a list of values corresponding to the value category indicated by *SlotName*. The (29b) format is used when conditions are to be imposed on the immediate constituent CC with the role value indicated by *RoleName*. The conditions are entered in *ConditionList*, which is a list of terms of the format (29a). Each element of *SelectionalConditionList* represents an obligatory condition, i.e., all the conditions in the list should be satisfied simultaneously. More general logical connectives such as negation and disjunction are not available in the current implementation. The retrieval result is obtained in the second argument *RetrievedCCList* as a list.
Examples:
(30a) Get all finite subordinate clauses in the file *sophie.text.*
(30b) *$destroy_ccb,*
$create_ccb('sophie.text','sophie.ccb'),
$activate_ccb('sophie.ccb'),
$retrieve_cc([attribute = fntcls, attribute = sub_cls], CCL),
$get_sent(CCL, SL).
(30c) *SL=['That the philosopher was right.'].*
(31a) Assuming that the current CC-base is the one activated in (30), get sentences/clauses whose subjects have the semantic feature *hum(an)*.
(31b) *$retrieve_cc([subject : [attribute = hum]], CCL),*
$get_sent(CCL, SentL).
(31c) *SentL=*
['Sophie opened the big envelope apprehensively.',
'Hilde began to describe her plan.',
'To describe her plan.',
'Sophie saw that the philosopher was right.',
'That the philosopher was right.'].
Note that all the embedded sentences are included in the retrieved CC list. Since the non-overt subject of *describe* in the sentence (7b) is analyzed as *Hilde* in the CC generation process, the infinitival clause *To describe her plan* is also retrieved.

3.3.3 Update operations

CCML provides two update operations, i.e., *$append_cc* and *$delete_cc*. These operations are used to add or delete the specified CCs from the CC-base indicated.
$append_cc
The formats are:
(32a) *$append_cc(CC, CCBFile).*
(32b) *$append_cc(CC).*
The first argument *CC* indicates a CC or a list of CCs to be appended to the CC-base specified in the second argument *CCBFile*. If the named CC-base is *current* or the operation is used in the format (32b), the *$append_cc* operation makes the appended CC(s) indicated in the first argument be directly accessible by retrieval operations.
Example:
(33a) Append the CC for the sentence *Sophie saw that the philosopher was right* to the current CC-base.
(33b) *$get_cc('Sophie saw that the philosopher was right', CC),*
$append_cc(CC).
$delete_cc
(34a) *$delete_cc(CC, CCBFile).*
(34b) *$delete_cc(CC).*
Removal of the indicated CC(s) from the current CC-base is carried out by this operation. The interpretation of the arguments and their uses are the same as those of *$append_cc*.

4 Conclusions

A sentence structure manipulation language CCML based on the language model CCM was proposed. In CCM each sentence is transformed into a CC, a nested relational structure in which the syntactic and semantic properties of the sentence are encoded in a uniform data structure. This uniformity in CC's data structure leads to a corresponding uniformity in the CCML operations. The CCML operations implemented so far cover a wide range of areas in sentence structure manipulations including sentence-CC inter-conversion operations, CC internal structure operations, and CC-base operations. The manipulation language CCML proposed in this paper is expected to be used in various natural language application systems such as second language learning systems and human computer communication systems, in which sentence structure manipulation plays an essential role.

References

C. J. Date. 1990. *An Introduction to Database Systems, Volume I, Fifth Edition.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.

F. Pereira and D. H. D. Warren. 1980. Definite clause grammars for language analysis – A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13:231-278.