1998 International Computer Symposium
Workshop on Artificial Intelligence
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

An Efficient Implementation of Conceptual Structures

Woo-Jeong Bae+), In-Cheol Park++), and Yong-Seok Lee+)

Dept. of Computer Science, Chonbuk Natl. Univ., Chonju, Korea+)
Dept. of Computer Science, Howon Univ. Kunsan, Korea++)
{topaz, yslee}@cbnlps.chonbuk.ac.kr+)
icpark@nlp.cs.howon.ac.kr++)

ABSTRACT

This paper presents an efficient implementation of conceptual structures to build a practical knowledge-based natural language processing system. For efficiency, we represent conceptual structures with trees, which are called conceptual trees in this paper. The motive of the representation is to implement conceptual structures with frames, which are well known as a powerful tool for organizing our knowledge inside computer. In addition, conceptual structures are so extended and restricted that they have the consistency and the simplicity of the representation. For the consistency, we extend set referents and redefine the operation *set join* to manipulate them. For the simplicity, we put some restrictions on conceptual structures, including *preorder expression* - they *put* the head concept first. By experiment, we show that computing time in performing operations for conceptual trees is significantly reduced.

1. INTRODUCTION

Sowa's conceptual graphs are a generalization of other semantic networks. He provides properties useful for representing and handling the meanings of natural language sentences[1]. These properties include canonical graphs, actors, a type hierarchy, type definitions, schemata, and prototypes. In addition, He precisely defines the representation, reasoning operations, semantics in his theory[2]. He argues that conceptual structures can serve as an intermediate language for translating computer-oriented formalisms to and from natural languages[3]. This role is important for knowledge-based natural language processing system because it is impossible to directly generate a logical structure which represent the exact meaning of a natural language sentence[4, 5]. To build a practical system, the intermediate language should be simply represented and efficiently handled inside computer.

Conceptual structures have been represented with graphs, which are called conceptual graphs. However, it is complex to represent and handle graphs inside computer. The complexity makes it difficult to implement a practical system. This example can be shown in the system representing feature structures with trees instead of graphs[6]. So we develop conceptual trees that are conceptual structures represented with trees. This tree representation aims at a natural representation of conceptual structures with frames inside computer. Frames provide a natural way to represent schemata, prototypes, inheritances, and default values. In addition, they make it easier to implement an actor, a type hierarchy, and other operations[7].

It is important to keep the consistency of the representation because it simplifies the handling of conceptual structures inside computer. Here, consistency means that sentences which have same syntactic or semantic structure must have the same formalism. By examples, we show that conceptual graphs cannot keep the consistency. To overcome it, we so extend set referents that an element in a set referent is a conceptual tree. In addition, we redefine the operation *set join,* which unifies two concepts with set referents[1].

Traversing conceptual structures is usually complex. It causes to drop the performance of a system. To simplify it, we use preorder expression in conceptual structures. So they put verbal concepts first and put nominal concepts before adjective concepts[1]. It enables to traverse a conceptual structure in only one direction. We also put some restrictions on conceptual structures to represent and handle them efficiently. This paper show these restrictions make more efficient to implement conceptual structures, keeping the expressive

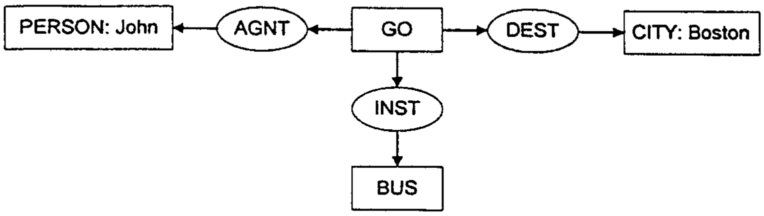

Figure 1 John is going to Boston by bus

power for natural language sentences.

We first describe the tree representation of conceptual structures. Next, we present the notation and the operation *set join* in conceptual trees. We then discuss about implementation of conceptual trees based on frames. Finally, We show that computing time in performing operations for conceptual trees is significantly reduced.

2. TREE REPRESENTATION OF CONCEPTUAL GRAPHS

2.1 Tree Translation of Conceptual Graphs

A conceptual graph is a bipartite graph representing the meanings of sentences with concept nodes and relation nodes. For an instance, the conceptual graph in Figure 1 represents the sentence *John is going to Boston by bus*[3]. The representation of conceptual trees is similar to the linear notation of conceptual graphs with some restrictions. For example, (2-1) and (2-2) shows a conceptual graph and a conceptual tree representing the sentence respectively:

[GO] - (2-1)
(AGNT) → [PERSON: John]
(DEST) → [CITY: Boston]
(INST) → [BUS].

[GO] - (2-2)
(AGNT) - [PERSON: John]
(DEST) - [CITY: Boston]
(INST) - [BUS].

The difference of the notations is trivial. However, conceptual trees are restricted by the preorder expression unlike conceptual graphs. For example, let us consider the conceptual graph in Figure 2. Depending on the choice of the first traversed concept node, we can get two conceptual graphs represented with the linear notation[3]:

[MONEY]← (PTNT)← [EARN] - (2-3)
(AGNT)→ [ELEPHANT: ∀]← (AGNT)← [PERFORM] -
(IN)→ [CIRCUS].

[CIRCUS]← (IN)← [PERFORM] - (2-4)
(AGNT)→ [ELEPHANT: ∀]← (AGNT)← [EARN] -
(PTNT)→[MONEY].

The following sentences can be generated from the above two conceptual graphs respectively [1]:

Every elephant which performs in a circus earns money. (2-5)

Every elephant which earns money performs in circus. (2-6)

Sentences (2-5) and (2-6) have different meanings. This shows us that a conceptual graph can be interpreted differently depending on the order of traverse in the graph. Therefore, conceptual graphs don't keep the uniqueness of the representation because they may have semantic ambiguity.

Because conceptual trees are restricted by the preorder expression, they put first a head concept which is a main concept of sentences, clauses, or phrases. Therefore, the sentences

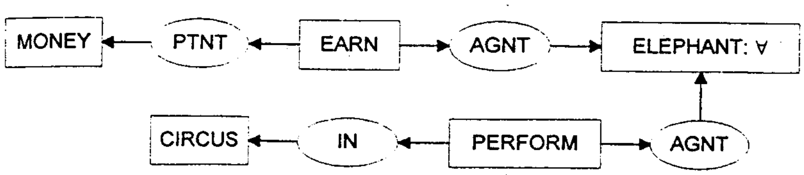

Figure 2 a conceptual graph

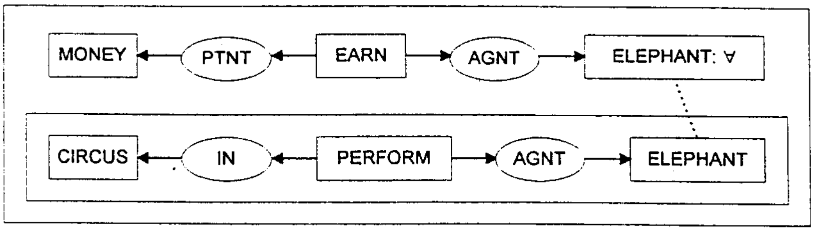

Figure 3 Using nested context

(2-5) and (2-6) can be represented with two different conceptual trees (2-7) and (2-8) respectively:

```
[EARN] -                                          (2-7)
  (AGNT) - [ELEPHANT: ∀] -
              (AGNT) - [PERFORM] - (IN) - [CIRCUS],,
  (PTNT) - [MONEY].

[PERFORM] -                                       (2-8)
  (AGNT) - [ELEPHANT: ∀] -
              (AGNT) - [EARN] - (PTNT) - [MONEY],,
  (IN) - [CIRCUS].
```

The conceptual trees (2-7) and (2-8) don't match fully. Therefore, they are interpreted as different meanings. It means that conceptual trees are able to keep the uniqueness of the representation.

Nested contexts can be used for distinguishing between sentences (2-5) and (2-6) in conceptual graphs[3]. For an instance, Figure 3 shows a conceptual graph using nested context. However, neither usage of nested contexts is natural in the representation of natural language sentences nor it is easy to handle the scope of nested contexts during operations. Use of the preorder expression in conceptual trees is similar to one of nested contexts in conceptual graphs. While conceptual graphs with nested contexts have a higher-order structure, conceptual trees always have the first-order structure, as shown in (2-7) and (2-8).

2.2 Restrictions on Relation Node

We represent a relation node with a label in concept node. The label representation of relation node does not reduce the expressive power for natural language sentences because a relation node represents only conceptual relation between concepts. In addition, it simplifies representation of conceptual trees inside computer and makes a frame representation possible ultimately. However, conceptual trees must have some restrictions to represent a relation node with a label.

Conceptual trees have only monadic and dyadic relations. To represent triadic relation or more, we use set referents. For example, the sentence *A person is between John and Sue* is represented with the conceptual graph (2-9) and the conceptual tree (2-10) respectively:

```
[PERSON] ← (BETW) -                               (2-9)
    1← [PERSON: John]
    2← [PERSON: Sue].

[PERSON] -                                        (2-10)
    (BETW) - [PERSON: {John, Sue}].
```

By representing triadic relation or more with set referents, conceptual trees can keep the consistency of representation. For example, let us consider representation for the sentence *Jane is between two persons*. If we keep the consistency, the conceptual graph and conceptual tree for the sentence must be (2-11) and (2-12) respectively:

```
[PERSON: Jane] ← (BETW) -                         (2-11)
    1← [PERSON]
    2← [PERSON].

[PERSON: Jane] -                                  (2-12)
    (BETW) - [PERSON: {*}@2].
```

However, the conceptual graph (2-11) is absurd because the number of concept node will be increased in proportion to the cardinality. Therefore, it is reasonable that triadic relation or more are represented with set referents. We also don't permit the duplication of relations. Instead, it is represented with set referents.

Conceptual trees don't have a directed arc. Instead, we use the hyphen('-') that only serves as the delimiter. The directed arc is unnecessary because conceptual trees always read in one direction by using the preorder expression. For example, the conceptual tree (2-12) always reads *there is a person Jane who is between two persons*. Conceptual trees are more efficient because they neither represent an arc inside computer nor consider the direction of an arc in

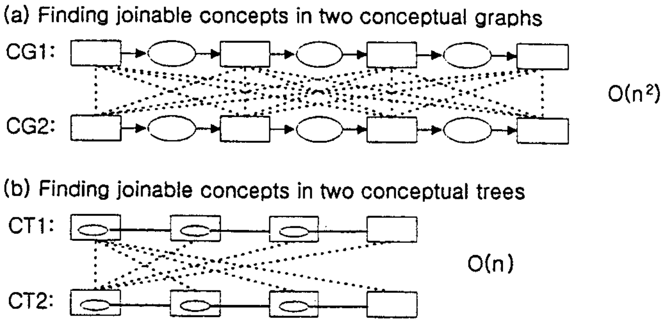

Figure 4 Finding joinable concepts

performing operations.

2.3 Extension of Set Referents

In conceptual graphs, the plural terms are represented with set referents[3]. In [8] and [2], they extended set referents and developed rules of inference for conceptual graphs with set referents. A set referent consists of a scope list, domain, and cardinality. All concepts in a domain must have a same type. For example, the sentence (2-13) can be represented with the conceptual graph (2-14) if all concepts for *John*, *Mary*, and *Tom* have the type PERSON:

At least two of John, Mary, and Tom own a house. (2-13)

```
[OWN: (p, h) o {*}@1] -                              (2-14)
  (STAT) ← [PERSON: ()p{#John,#Mary,#Tom}@2-∞]
  (PTNT) → [HOUSE: (p) h {*}@1-∞].
```

We have so extended set referents that an element in the domain is a conceptual tree. The type of concept with a set referent must be one of common subtypes of types in the domain and its initial type is the universal type T. The conceptual tree (2-15) shows us our extended set referents explicitly:

```
[[OWN: *] -                                          (2-15)
  (STAT) - [T: {[BOY: John], [GIRL: Mary],
                [BOY: Tom]}@2-∞]
  (PTNT) - [HOUSE: *].
```

The motive of extension of set referents is to give the consistency of the representation to conceptual trees. For example, let us consider the conceptual tree (2-10). Though *John* and *Sue* may conform to the different types, we can represent it with the same form:

```
[PERSON] -                                           (2-16)
  (BETW) - [PERSON: {[BOY: John], [GIRL: Sue]}].
```

3. CONCEPTUAL TREES

3.1 Basic Notation

Representation of conceptual trees is similar to frame representation: a relation label has role of a slot in a frame and the value of a slot is an individual marker of another concept node linked to the relation label. A concept node is represented with a frame inside computer and we call it conceptual frame. However, the notation of conceptual trees has the similar form with the linear notation of conceptual graphs. The reason is that conceptual trees are based on the semantics of conceptual graphs. Therefore,

Table 1 BNF representation for conceptual trees

```
conceptual-tree ::= ~concept {- modifier-sequence}* [end-marker]
modifier-sequence ::= modifier | modifier modifier-sequence
modifier ::= property | slot
end-marker ::= . | ,
property ::= (monadic-relation)
slot ::= (dyadic-relation) - conceptual-tree
monadic-relation ::= NOT | PSBL | PAST | ···
dyadic-relation ::= AGNT | DEST | BETW | ···
```

conceptual graphs can be viewed as an integrated formalism based on frames for internal representation and on conceptual graphs for semantics. Table 1 shows us BNF notation of conceptual trees. BNF notation for a concept is similar to one in [2].

Use of the preorder expression does not reduce the expressive power for natural language sentences. We can find that feature structures are represented with trees, which are similar to preorder expression[4, 6, 9]. For instance, Table 2 shows feature structures and conceptual trees for the sentence *John sold the book to Mary*. We can map feature structures to conceptual trees directly and easily. It makes conceptual trees have preorder expression naturally.

To keep the preorder expression during operating, all the operations of conceptual trees are performed on the basis of head concept. This is similar to head-driven unification approach[9]. Figure 4 shows this restriction graphically. As shown in Figure 4, it can decrease the complexity of the operations.

3.2 The Operation Set Join

In this section, we define set join to handle our extended set referents. We assume that If two set referents have different cardinality, then the resulting set referent has more restricted cardinality. For example, set join of the concept [PERSON: {*}@2] for the phrase two persons and the concept [BOY: {*}@5] for the phrase five boys may generate the concept [BOY: {*}@2]. The algorithm of set join follows:

```
Algorithm set join of two concept nodes that
            have a set referent
operator set-join(node1, node2: concept nodes
           which have a set referent)
var
    cnode: concept node;
    ct1, ct2: conceptual trees;
    new-domain: a domain;
    t: a type;
begin
    make a concept node cnode of which type
    is the maximal common subtype of types
    of node1 and node2;
    add scope of node1 to cnode's set referent;
    get more restricted cardinality from node1
    and node2 and add it to cnode;
    if both node1 and node2 are generic set
    then
        return cnode;
    if either node1 or node2 is generic set
    then begin
        if node2 is genetic set then
            swap node1 with node2;
        if type of node1 < type of node2 then
            foreach type t in the domain of node2
            do begin
                if t is not subtype of type of
                node1 then
                    return failed;
            end do
        add the domain of node2 to cnode;
    end
    else begin
        initialize new-domain to a empty set;
        foreach conceptual tree ct1 in the
        domain of node1 do
        begin
            foreach conceptual tree ct2 in the
            domain of node2 do
                if success in join of ct1 and ct2
                then
                begin
                    append the result of maximal
                    join of ct1 and ct2 to
                    new-domain;
                    remove ct2 at the domain of
                    node2;
                    continue the outer foreach
                    statement;
                end
            append ct1 and ct2 to new-domain;
        end
        add new-domain to cnode;
        if new-domain does not match the
        cardinality of cnode then
            return failed;
    end
    return cnode;
end.
```

As shown in the above Algorithm, an failure in set join can occur when the type of the resulting concept is not common subtype of types in its domain. For example, let us consider the following concepts:

[DOG: {*}@2]. (3-1)
[PET: {*}@2]. (3-2)
[ANIMAL: [BEAGLE: Snoopy], [CAT: Tom]]. (3-3)

The join of (3-1) and (3-3) will be failed because CAT is not subtype of DOG, while the join of (3-2) and (3-3) will generate the concept [PET: [BEAGLE: Snoopy], [CAT: Tom]] with success.

As both the domains of two concepts to be

Table 2 John sold the book to Mary

(a) feature structure	(b) a conceptual tree
(S **SUBJ** (NP **NAME** John) **NUM** {3s}) **MAIN-V** sold **TENSE** {PAST} **VOICE** {ACTIVE} **OBJ** (NP **DET** the **HEAD** book **NUM** {3s}) **MODS** (PP **PREP** to **POBJ** (NP **NAME** Mary **NUM** {3s})))	[SELL] - (PAST) (AGNT) - [PERSON: John] (PTNT) - [PERSON: Mary] (OBJ) - [BOOK: #].

joined are not generic domains, the resulting domain is the union set of the domains. If two concepts in the resulting domain are joinable, they will be maximally joined. For example, let us consider the following sentences:

Jane owns a black cat and a white dog. (3-4)

She owns two pets - Tom and Snoopy. (3-5)

The sentences (3-4) and (3-5) can be represented with the conceptual trees (3-6) and (3-7) respectively:

```
[OWN] -                                              (3-6)
   (STAT) - [PERSON: Jane]
   (PTNT) - [T: {[CAT] - (ATTR) - [COLOR: Black],
                 [DOG] - (ATTR) - [COLOR: White]}].

[OWN] -                                              (3-7)
   (STAT) - [GIRL]
   (PTNT) - [PET: {[CAT: Tom],
                   [BEAGLE: Snoopy]}@2].
```

If the concept [CAT] is joinable with the concept [CAT: Tom] and the concept [DOG] with [BEAGEL: Snoopy] simultaneously, then the resulting conceptual tree follows:

```
[OWN] -                                              (3-8)
  (STAT) - [PERSON: Jane]
  (PTNT) - [PET: {[CAT: Tom] -
                    (ATTR) - [COLOR: Black],
                  [BEAGLE: Snoopy] -
                    (ATTR) - [COLOR: White]}@2].
```

If one or both of the above couples of concepts are not joinable, then the resulting domain does not match the cardinality of the resulting concept. This shows us another case that set join is failed.

4. IMPLEMENTATION AND EXPERIMENT

A concept node in conceptual trees is represented with a frame, which is called conceptual frame in this paper. The basic unit of a conceptual frame is an index node, which is a vector table. It uses a hash table to allow fast access of structures. Figure 5 illustrates the memory map of the index node. This map is based on one of HYPERFRAME system, which is a practical frame system[10]. In Figure 5, the field LABEL contains the individual marker for a concept node. The marker plays the role of an identifier for accessing a knowledge base. The fields TYPE and REFERENT correspond the type and referent fields in a concept node respectively. The field COREFERENCE is for representing a coreference link. The field PROPERTY stores monadic relations attached to the concept node, while the field SLOT stores dyadic relations. The value attached a slot is an individual marker for the head concept node in a

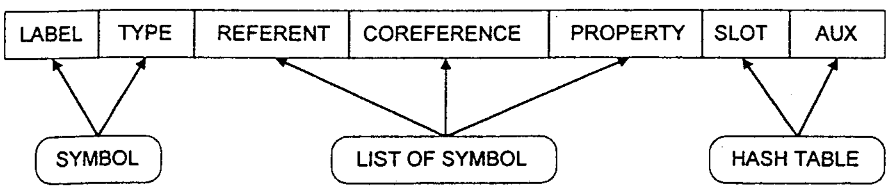

Figure 5 Memory map of a conceptual frame

Table 3 computing time for *unify* operation

	(a) Conceptual Graphs	(b) Restricted Conceptual Graphs	(c) Conceptual Trees
First Result	13.24 sec.	10.43 sec.	7.24 sec.
Second Result	24.66 sec.	18.84 sec.	12.71 sec.

conceptual tree. Finally, the field AUX is a reserved space for storing informations required to resolve various linguistic problems such as ambiguities.

Finally, we present an experiment which shows that conceptual trees significantly can reduce computing time compared with conceptual graphs. Table 3 shows computing times in performing the operation *unify*, which generates a *maximally joined* conceptual tree, appling four formation rules(*copy*, *restrict*, *join*, and *simplify*) to two conceptual trees. For the experiment, we build thirty conceptual graphs and conceptual trees respectively and we perform the operation one hundred times. In the case (a) we have unified pure conceptual graphs. In the case (b) we have unified conceptual graphs restricted by preorder expression. In the case (c), finally, we have unified conceptual trees. In the first experiment, conceptual structures have concept nodes between two and five. In the second, they have only four or five ones.

5. CONCLUSION

Conceptual trees can be regarded as an integration of conceptual structures for semantics and frames for implementation. We give extensions and restrictions to conceptual trees to represent them with frames efficiently and simply. We also show conceptual trees are significantly reduce computing time in their operations.

REFERENCES

[1] John F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, pp. 69-210 and 230-246, 1984.

[2] David A. Gardiner, Bosco S. Tjan, and James R. Slagle, "Extending Conceptual Structures: Representation Issues and Reasoning Operations" in *Conceptual Structures: Current Research and Practice*, Ellis Horwood, 1992.

[3] John F. Sowa, "Conceptual Graphs Summary" in *Conceptual Structures: Current Research and Practice*, Ellis Horwood, 1992.

[4] James Allen, *NATURAL LANGUAGE UNDERSTANDING (2nd ed.)*, The B/C Publishing Company, pp. 161 and 232-233, 1995.

[5] Hiyan Alshawi (ed.), *The Core Language Engine*, The MIT Press, pp. 1-9, 1992.

[6] Masaru Tomita and Kevin Knight, "Pseudo-Unification and Full-Unification", CMU-CMT-88-MEMO, 1987.

[7] George F. Luger and William A. Stubblefield, *ARTIFICIAL INTELLIGENCE and the Design of Expert Systems*, The B/C Publishing Company, pp. 359 - 363, 1989.

[8] Bosco S. Tjan, David A. Gardiner, and James R. Slagle, "Representing and Reasoning with Set Referents and Numerical Quantifiers" in *Conceptual Structures: Current Research and Practice*, Ellis Horwood, 1992.

[9] Derek Proudian and Carl Pollard, "Parsing Head-Driven Phrase Structure Grammar", 23rd Annual Meeting of the Association for Computational Linguistics, 1985.

[10] Eric H. Nyberg, 3rd, *The HyperFrame User's Guide*, Cognitive Research Laboratories Technical Memo, 1990.