TEMPORAL KNOWLEDGE USING CONCEPTUAL GRAPH

A thesis Presented to the Department of Computer
Science in Partial Fulfillment
of the Requirement for the Degree
Master of Science
in Computer Science

Tanakarn Saithong

September, 1998

TEMPORAL KNOWLEDGE USING CONCEPTUAL GRAPH

A thesis presented to the department of Computer Science in partial fulfillment of the requirement for the degree Master of Science in Computer Science

Tanakarn Saithong

September, 1998

Master Thesis Title : **Temporal Knowledge Using Conceptual Graph**

By : **Mr. Tanakarn Saithong**

The Department of Computer Science, Faculty of Science and Technology, Assumption University had this final report of the 12 credit course, SC 7000 : Master Thesis, submitted in partial fulfillment of the requirement for the degree of Master of Science in Computer Science.

Approval Committee :

(Dr. Tang Van To) Thesis Advisor	(Asst. Prof. Dr. Pratit Santiprabhob) Committee Member
(Dr.Thitipong Tanprasert) Committee Member	(Asst. Prof. Dr. Somnuk Keretho) Representative of Ministry of University Affairs

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks and deep sense of gratitude to his advisor Dr. Tang Van To for his constant guidance and moral support throughout the period of this thesis work.

The author wishes to convey special thanks to his committee members Dr. Pratit Santiprabhob and Dr. Thitipong Tanprasert for their constructive criticisms and useful suggestions, and thanks to Dr. Somnuk Keretho, an external committee member from Ministry of University Affairs, for his constructive suggestions.

Sincere thanks are also for all faculty members, staffs and secretaries of the Department of Computer Science for their help and kindness.

Many thanks to friends in ABAC for their encouragement throughout the study at ABAC.

Finally, no word can express the contribution of the author's family. Their continued love and encouragement throughout the study was the backbone behind the success of the author in study.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Temporal knowledge is the knowledge about time of the events and the temporal relationships between events. The temporal knowledge engine allow us to infer the temporal knowledge and temporal relationships.

In this study, a CG model for representing temporal knowledge is studied, a mapping from CG to RDBMS tables also is derived. A simple program for input, edit, display and query on the temporal knowledge is written in Visual Basic, and an inference engine in Prolog is also developed. These two programs are interacted through text files. The engine allows us to reason the possible time of event based on the temporal knowledge available in the knowledge base, it also can deliver the topological order of events.

CHAPTER 1

INTRODUCTION

1.1 STATEMENT OF THE PROBLEM

For many years, one of the research topics of artificial intelligence has been developed for knowledge representations. In order to make knowledge suitable for processing by computers, many knowledge representation method have been developed. Among them are production rules, frames, scripts, predicate transition networks, and conceptual graphs.

Conceptual Graph (CG) proposed by SOWA (1984) is a knowledge representation language base on linguistics and it attends to incorporate concepts in natural and formal languages. This knowledge representation scheme which has gained more attention recently is a general framework model for representing knowledge, and it can be used as a main model of future integrated knowledge systems [FARGUES et al.(1986)].

CG model generalizes many ideas contained in preceding works on natural language semantics and knowledge representation [JACKMAN and PAVELIN (1988)]. A system of logic for representing natural language semantics, the mapping from natural language to CG is shorter, simpler, and more direct than the mapping to calculus predicate [SOWA (1991)].

In many situations, it may be difficult to map data in precise form. In order to represent the time of events, and attempt to mix the temporal knowledge and natural language processing to CG. Temporal knowledge specifies the time of the event. It allows us to infer the order of events that happened. Another feature is to formulate CG programs to be usable as a programming language, as predicate logic does. It will be very useful to a wide range of application fields if temporal knowledge can also be used as a programming language.

Therefore, in this study I propose to study the CG, and the possibility to apply to temporal logic. The study emphasizes on the structure of CGs, operations on CGs, how to map CG to RDBMS, how to map temporal knowledge to CGs and how to inference about the temporal knowledge on CGs. And if it is possible how to make the Q/A systems using CGs. My main research try to find a common approach. It can be used to map from temporal knowledge to RDBMS tables, which is easy to query and implement.

1.2 OBJECTIVES AND SCOPE OF THE STUDY

The main objectives of this study are :

1. Study the foundation of CG Graph and its application.
2. Study a temporal knowledge such as temporal logic and temporal relations.
3. Mapping the sentence with the temporal knowledge to conceptual graph.
4. Develop the temporal knowledge inference engine.
5. Finally, a program allows to infer about temporal relationship occurring time of events is developed.

CHAPTER 2

CONCEPTUAL GRAPH AND TEMPORAL KNOWLEDGE

In this chapter, definition and representation of CG are reviewed in Section 2.1, the process of mapping sentences to CGs is discussed in Section 2.2 and Section 2.3. Time interval, temporal object, temporal relation and temporal logic of temporal knowledge are discussed in Section 2.4.

2.1 CONCEPTUAL GRAPH

Conceptual Graphs (CG) are graph-based notations with a fundamental formal basis that can be used for knowledge representation and computer based reasoning. The formalism of CG reviewed here is based on SOWA (1984).

2.1.1 Definition

CG is formally defined as a *finite, connected, bipartite graph*. The two kinds of nodes are *concepts* and *conceptual relations* (CR). Every CR has one or more arcs, each of which must be linked to some concepts. A single concept by itself may form a CG [SOWA (1984)].

It should be noted that the different between concept types and concept instances as:

a) Concept types represent classes or types of entities, attributes, states and events.

b) Concept instances (individual concept) or references is an instantiation of a concept type.

b) Concept instances (individual concept) or references is an instantiation of a concept type.

2.1.2 Representation of Conceptual Graph

There are two forms of display for CGs [SOWA (1984)] :

a) Graphical form : In this form a concept node is represented by a rectangle and CR is represented by an oval.

Figure 2.1 : Graphical Form of Display.

b) Linear form : In this form a concept node is represented by [ ] and CR by ( ).

[PERSON : John] ← (Agnt) ← [PLAY] → (Obj) → [TENNIS]

Figure 2.2 : Linear Form of Display.

As mentioned in 2.1.1 definition, a conceptual node itself may form a CG, in this case a CG becomes a compound graph.

2.1.3 Atomic Conceptual Graph

Atomic Conceptual Graph (ACG) [ELLIS (1991)] is a CG that contains no logical connective and no quantifier other than the implicit existential quantifiers (Fig 2.3). An ACG containing only individual references in all the concept nodes is termed as *a ground ACG* (Fig 2.4).

[PERSON] ← (Agnt) ← [SIT] → (Loc) → [PLACE]

Figure 2.3 : An Atomic Conceptual Graph.

[PERSON : Bill] ← (Agnt) ← [SIT : #543] → (Loc) → [PLACE : Shop]

Figure 2.4 : A ground ACG, All Concept Nodes Contain Individual References.

2.1.4 Compound Graph

A CG is called a *Compound graph* [SOWA (1984)], if any or all of the following conditions hold :

a) It contains contexts of depth higher than 0,

b) It contains atomic CG connected by coreference links. Coreference link is used to connect identical concepts. It is shown using a dotted line in the graphic form.

Example 2.1 A Compound graph for "A person would go to ABAC, if he is a student of ABAC."

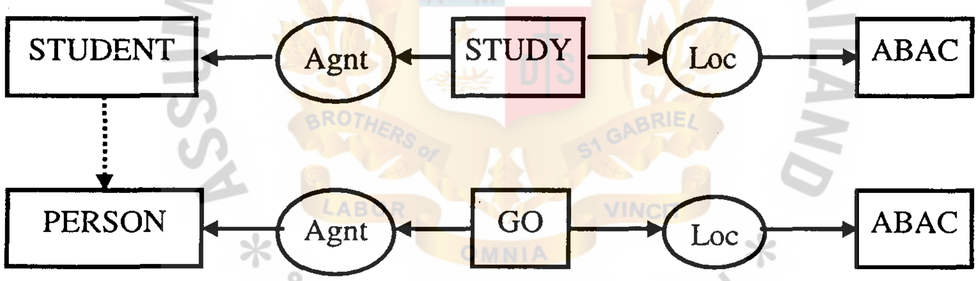


Figure 2.5 : A Compound Graph for "A person would go to ABAC, if he is a student of ABAC".

2.2 MAPPING SENTENCE TO CONCEPTUAL GRAPH

2.2.1 Types of References

There are extended references. They correspond to generalized determiners in natural languages : existential, individual, named individual, unique existential, definite reference, set, generic set, etc. The use of these references, which provides more flexibility in natural language representation but without semantic interpretation [SOWA (1993)]. The typically extended references is show in table 2.1

Type of reference	Example	English reading
Universal	[CAT : ∀]	every cat
Singular	[CAT : @ 1]	exactly one cat
Generic set	[CAT : {*}]	Cats
Counted set	[CAT : {*} @ 3]	three cats
Set of individuals	[CAT : {Yoyo, Meaw}]	Yoyo and Meaw
List of individuals	[CAT : <Yoyo, Meaw>]	Yoyo and then Meaw
Question	[CAT: ?]	which cat ?
Plural question	[CAT: {*} ?]	which cats ?
Measure	[Interval: @ 5 sec]	interval of 5 seconds

Table 2.1 : Types of References.

2.2.2 Mapping Aspects

We shall illustrate several aspects of the system. Each feature is accompanied with several examples [Sait and Jame (1993)].

2.2.2.1 **Subject-Verb Agreement** : The translator obeys all basic English grammar rules, including the subject and verb agreement rule.

a) John lives in Bangkok.

[PERSON : John] ← [LIVE] → [CITY : Bangkok]

b) There exist John and Jane living in Bangkok.

[PERSON: ()P {John, Jane} @2] ← [LIVE] → [CITY : Bangkok]

2.2.2.2 **Tenses** : The default tense of the translator is Present (Progressive) Tense. However, the system is capable of handling other English language tenses as well.

a) John was speaking.

(PAST PROGRESSIVE) → { [PERSON: John] → [SPEAK] }

b) Bob lived in the Bangkok.

(PAST PROGRESSIVE) → { [PERSON:Bob] ← [LIVE] → [CITY: Bangkok] }

c) John is speaking.

(PRESENT PROGRESSIVE) → { [PERSON: John] → [SPEAK] }

d) John is talking to Bob.

(PRESENT PROGRESSIVE) → { [PERSON : John] → [TALK] → [PERSON : Bob] }

2.2.2.3 **Cardinality Information** : The examples below show the importance of this field in natural language translation :

a) There exists a person.

[PERSON : ()P {*} @1]

b) There exist 1 to 5 persons.

[PERSON : ()P {*} @1-5]

c) There exist at least 3 persons.

[PERSON : ()P {*} @3-∞]

d) There exist 2 to 3 persons among John, Jane, and Jill.

[PERSON : ()P {John, Jane, Jill} @2-3]

e) The year 1997 consists of exactly 365 days.

[YEAR : 1997] ← [CONSIST] → [DAY : ()D {*} @365]

2.2.2.4 **Cardinality is not always needed** : There are some cases, however, where cardinality information can be omitted, resulting in more natural translations.

For example : There exist persons John and JANE.

[PERSON : ()P {John, Jane} @2]

2.2.2.5 **Simple Disjunctions** : The Conceptual Structure notation is capable of handling disjunctions as well, again through the cardinality information.

For example : There exists persons John or Jane.

[PERSON : ()P {John, Jane} @1]

2.2.3 Hierarchical Links [Sait and Jame (1993)]

2.2.3.1 **Link between individual concepts**. There are two kind of links between individuals.

a) Relation corresponding to individual role.

For example : There is a person who walks.

[PERSON:*X] ← [AGT] ← [WALK:*Y]

b) Cross-reference link between two individuals, to express the same object.

For example : There is a person named Mary, she is a teacher.

[PERSON : "Mary", *X] [TEACHER:*X]

2.2.3.2 Link from a prototype to an individual concept.

For example : Every elephant is grey.

[ELEPHANT:∀] → [COLOR] → [COLOR : gray]

2.2.4 Canonical Formations of Conceptual Graphs

Canonical graphs is a CG that specifies the constraints on the pattern of concepts and relations that may be linked to a concept and relation type.

There are four canonical formation rules for deriving a CG w from CG u and v (where u and v may be the same graph) [SOWA (1984)]. Assume that u and v are two CGs in Fig 2.6.

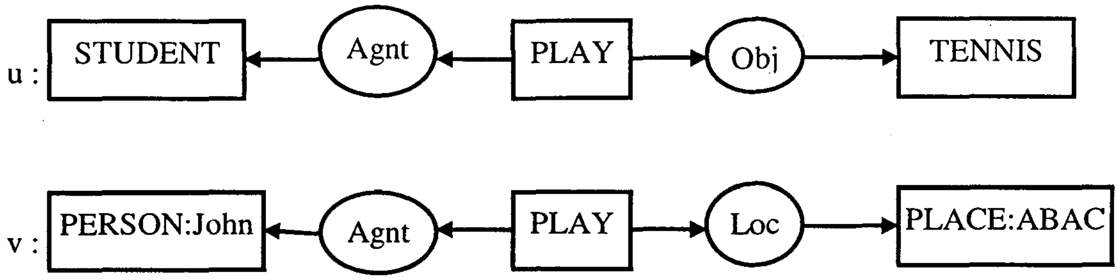


Figure 2.6 : Two CGs "A student plays tennis" and "The person John plays at ABAC".

2.2.4.1 **Copy(u, w)** : The copy rule allows us to form a new graph, w, that is the exact copy of u.

2.2.4.2 Restrict(u:c, w:c) : Let c be a concept of u that is not nested inside any context and with a reference that is either a constant or an existential quantifier. Then w is u with c restricted either by type or by reference : restriction by type replaces the type label of c with some subtype; and restriction by reference replaces an existential quantifier with a constant. From Fig 2.6 we can replace PERSON with STUDENT, and represented in Fig 2.7, called restriction by type.

Figure 2.7 : The Graphs Obtained After Restriction.

2.2.4.3 Join(u:c, v:d, w:c) : The join rule lets us combine two graphs into a single graph. Let c be a concept of u and d a concept of v, where neither c nor d is nested inside a context and both c and d have identical type and reference fields. Then w is the graph obtained by deleting d and linking to c all arcs of CRs that had been linked to d. From Fig 2.6 we can join CG u and CG in Fig 2.7 by delete concept STUDENT in u and represented in Fig 2.8.

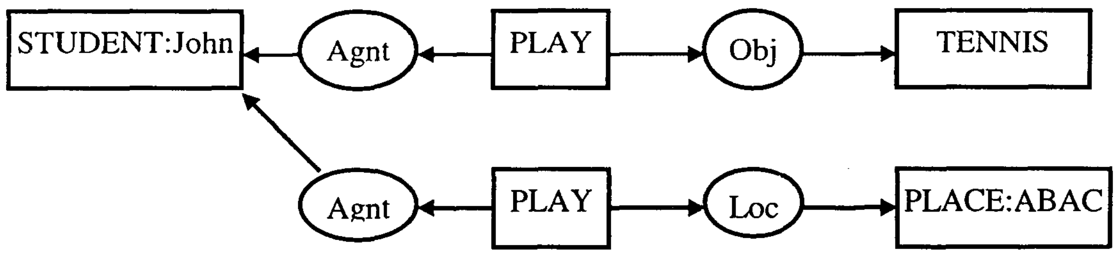


Figure 2.8 : The Graph Obtained After Join.

2.2.4.4 **Simplify(u:r, u:s, w:r)** : w is u where one of a pair of duplicate CRs in u has been deleted. Two CRs r and s in the graph u are said to be duplicates if they are of exactly the same type, and for each i, the i-th arc of r is attached to the same concept as the i-th arc of s. Duplicate relations often occur as the result of a join operation, as in Fig 2.8 we can delete relation [STUDENT : "John"] ← (Agnt) ← [PLAY] and represented in Fig 2.9.

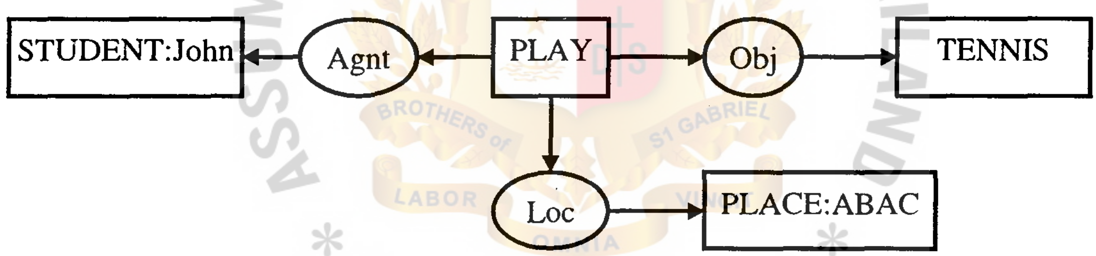


Figure 2.9 : The Graphs Obtained After Simplification.

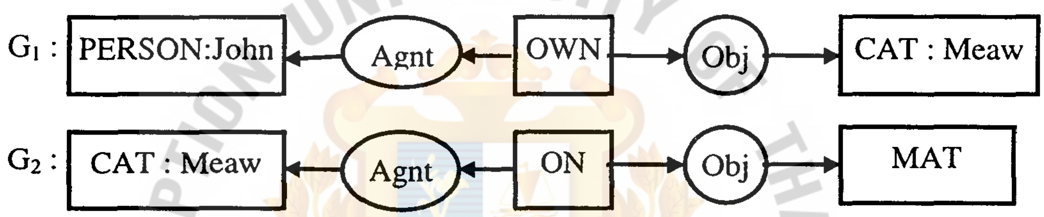
2.2.5 Maximal Join

Maximal join(G_1:C_1, G_2:C_2, G_3) is an important and useful operation defined on CG[JACKMAN and PAVELIN (1988)], which can be considered for the composition of CGs. It has been defined as a join of two graphs followed by a sequence of restrictions, joins and simplifications that explain in 2.2.4 Canonical Formations of Conceptual Graphs, so that as much matching and merging of the original graph as possible is performed.

The algorithm for maximal joining the graph G_1 and G_2 is the following steps.

1) List all concepts contained in G_1 and G_2.
2) Search for pairs of concept C_1 and C_2 which can be element of the projection graph.
3) Generate lists of relation which are connect to C_1 and C_2 in the corresponding graph.
4) Generate the resulting G_3 which consists of the projection graph link to subgraph that connect with C_1 and C_2 in G_1 and G_2.

__Example 2.2__ Given G_1 and G_2 are two CGs, as in below figures.



From the algorithm that described above, we have

Step 1 : G_1 has the concept [PERSON : John] and [CAT : Meaw]

G_2 has the concept [CAT : Meaw] and [MAT]

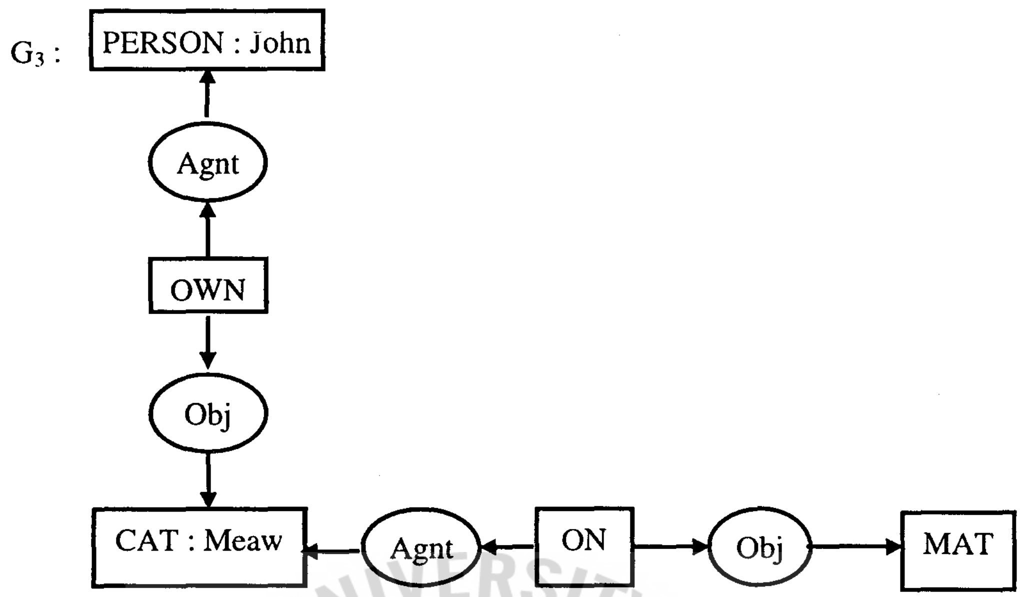
Step 2 : Search the pairs of concept :

C_1 = [CAT : Meaw]

C_2 = [CAT : Meaw]

Step 3 : Generate the lists of relation.

Step 4 : Generate the resulting graph G_3 :



2.2.6 Type Hierarchy

Type Hierarchy is a partial ordering on the set of types, indicated by the symbol ≤. If s and t are types and t ≤ s, then t is said to be a **subtype** of s and s is said to be a **supertype** of t [Sait and Jame (1993)]. A type may have more than one supertype and more than one subtype.

If s, t, and u are types, with t ≤ s and t ≤ u, then t is said to be a **common subtype** of s and u. Similarly, if s ≤ v and u ≤ v then v is a **common supertype** of s and u.

The **universal type**, indicated by T, is a supertype of all types. The **absurd type**, indicated by ⊥, is a subtype of all types.

2.3 MAPPING SENTENCE TO CONCEPTUAL GRAPH WITH THE HELP OF PARSING TECHNIQUES

This requires a knowledge of the structure of the sentence, the roles of individual words and how the words modify each other. The process of determining the syntactical structure of a sentence is known as parsing.

Parsing is the process of analyzing a sentence by taking it apart word-by-word and determining its structure from its component parts and subparts. The structure of a sentence can be represented with a syntactic tree. The parsing process is basically the inverse of the sentence generation process since it involves finding a grammatical sentence structure from an input string. When given an input string, the lexical parts must first be identified by type, and then the role they play in a sentence must be determined. These parts can then be combined successively into larger units until a complete structure has been completed.

To determine the meaning of a word, a parser must have access to a lexicon. When the parser selects a word from the input stream it locates the word in the lexicon and obtains the word's possible function and other features, including semantic information. This information is then used in building a parse tree.

Example 2.3 Parse tree for a sentence "John plays tennis." is given in Fig 2.10.

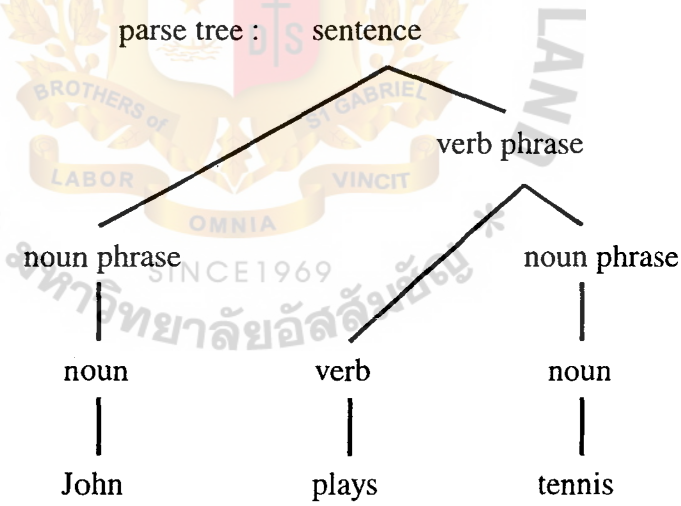


Figure 2.10 : A Parse Tree for "John plays tennis.".

Once a parse tree is obtained, we apply the rule that is given in 2.2 to map the sentence into CG.

2.4 TEMPORAL KNOWLEDGE

An approach that can be used to model temporal information. CG can be extended to include the notion of time coordinate systems in the form of agent perspective and temporal localizations.

Any discourse contains sentences which result from speech acts performed by the narrator (speaker or writer) and received by another agent (hearer or reader). A conversation between agents is conducted in what we call a *conversational context*, the *place* and *time* where agents interact, as well as the *world knowledge* that is used.

This approach distinguishes two levels for representing the information contained in a discourse :

a) Conceptual level which describes the relationships (temporal relations) associating temporal entities (objects, situations, localizations, perspectives).

b) Linguistic level which contains the linguistic information needed for uttering the discourse sentences.

2.4.1 Time Intervals

All temporal structures are associated with a time interval [Bernard (1993)]. An absolute *time reference* (TR, called "time axis") composed of a set of elements called "time points". An elementary time interval is a continuous sub-set of TR.

The elementary time interval is specified by a list of parameters :

a) Begin Time (BT)

b) End Time (ET) (lower and upper bounds of the time interval on TR)

c) Time Scale (TS) (unit used to measure the begin-time and end-time on TR)

d) Duration Time (DT)

e) Duration Scale (DS)

A *multiple time interval* is composed of a set of elementary time intervals, which may be contiguous or not.

2.4.2 Temporal Object

A temporal object is a concept that is characterized by a time interval such as "day", "week", "month" or "year" [Bernard (1993)]. A temporal object can be decomposed into other temporal object. For instance "day" can be decomposed into "morning", "afternoon", "evening" and "noon".

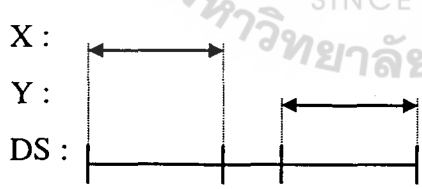
We represent a temporal object with a rectangle. The rectangle represents the corresponding time interval. On the top of the rectangle we indicate BT and ET.

2.4.3 Temporal Relation

Temporal relations is the relation between two temporal objects. A fundamental set of thirteen temporal relations [Allen (1983)] are shown in the following:

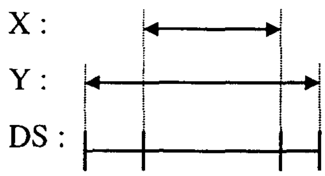
1) BEFORE (X, Y) - time interval X is before Y, and they do not overlap.
- ET(X) < BT(Y).

It can be seen in the following diagram :



2) DURING (X,Y) - time interval X is fully contained within Y, although they may coincide on their end points.
- BT(X) > BT(Y).
- ET(X) < ET(Y).

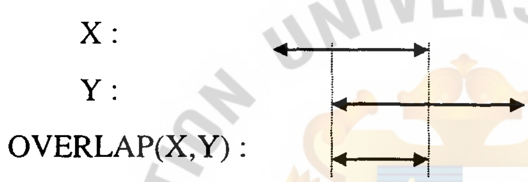
It can be seen in the following diagram :



3) OVERLAP (X,Y) - time interval X start before Y, and they overlap.

- BT(X) < BT(Y).
- ET(X) < ET(Y).
- ET(X) - BT(Y) = OVERLAP(X,Y).

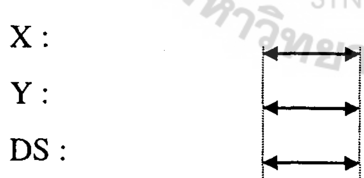
It can be seen in the following diagram :



4) EQUAL (X,Y) - time interval X is equal Y.

- BT(X) = BT(Y).
- ET(X) = ET(Y).

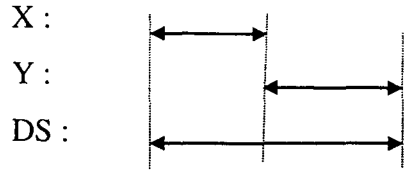
It can be seen in the following diagram :



5) MEET (X,Y) - time interval X start before Y.

- ET(X) = BT(Y).

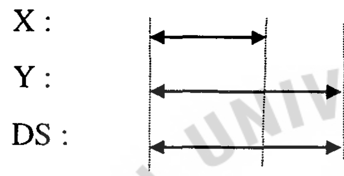
It can be seen in the following diagram :



6) START (X,Y) - time interval X start at the same time of Y.

- BT(X) = BT(Y).

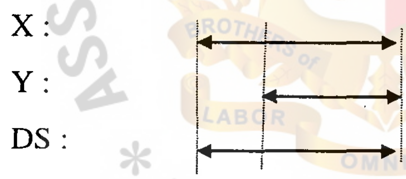
It can be seen in the following diagram :



7) FINISH (X,Y) - time interval X finish at the same time of Y.

- ET(X) = ET(Y).

It can be seen in the following diagram :



The remaining six temporal relations are inverse of FINISH, MEET, DURING, BEFORE, OVERLAP and START.

8) FINISH^{-1} (Y,X) is the inverse of FINISH (X,Y).

9) MEET^{-1} (Y,X) is the inverse of MEET (X,Y).

10) DURING^{-1} (Y,X) is the inverse of DURING (X,Y).

11) START^{-1} (Y,X) is the inverse of START (X,Y).

12) OVERLAP^{-1} (Y,X) is the inverse of OVERLAP (X,Y).

13) AFTER (Y,X) is the inverse of BEFORE (X,Y).

A temporal relation associating two temporal objects is represented by a circle which is related to the two objects by arrow. The circle contains the temporal relation type (such as BEFORE, AFTER, DURING). We consider eight primitive relations called “AFTER”, “BEFORE” , “DURING”, “EQUAL”, “FINISH”, “OVERLAP”, “MEET” and “START” between two intervals.

Example 2.4 Temporal objects for a sentence “John played tennis on December 10,1997 from 9:00 to 9:30 after he had read a cartoon-book” are given in the Fig 2.11.

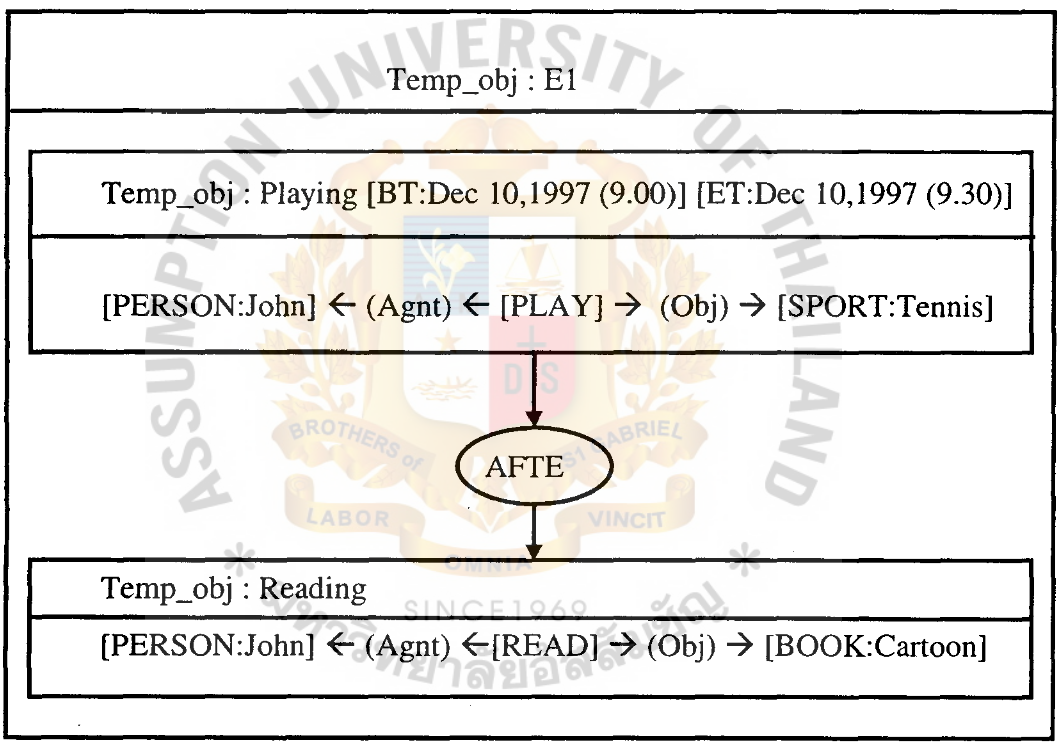


Figure 2.11 : Temporal Objects for “John played tennis on December 10,1997 from 9:00 to 9:30 after he had read a cartoon-book”.

2.4.4 Temporal Logic

All systems of temporal logic that we shall consider are extensions of minimal temporal logic and are obtained by impose further constraints on relation of temporal precedence [Turner (1984)]. The concept of "branching time" is obtained by impose two constraints R:

R1 : $(\forall t\ \forall s\ \forall r)\ ((R(t, s)\ \&\ R(s, r)) \rightarrow R(t, r))$

R2 : $(\forall t\ \forall s\ \forall r)\ ((R(t, r)\ \&\ R(s, r)) \rightarrow R(t, s) \vee (t = s) \vee R(s, t))$

Condition, R2, referred to as backwards linear.

R3 : $(\forall t\ \forall s)\ ((R(s, t) \vee s) = (t \vee R(t, s)))$

Obviously, R3 is equivalent to the conjunction of R2, and forwards linear given as R4:

R4 : $(\forall t\ \forall s\ \forall r)\ (R(r, t)\ \&\ R(r, s)) \rightarrow R(s, t) \vee (s = t) \vee R(t, s))$

R5 : $(\forall s)\ (\exists t)\ (R(t, s))$

R6 : $(\forall s)\ (\exists t)\ (R(s, t))$

R5 guarantees that time has no beginning, and R6 that time has no end.

Each of temporal relations is represented by a predicate logic, and they are governed by a set of axioms of which the following are representative :

1) Before(A,B), Before(B,C) → Before(A,C)
2) During(A,B), During(B,C) → During(A,C)
3) Equal(A,B), Equal(B,C) → Equal(A,C)
4) Finish(A,B), Finish(B,C) → Finish(A,C)
5) Start(A,B), Start(B,C) → Start(A,C)
6) After(A,B) → Before(B,A)
7) Equal(A,B) → Equal(B,A)
8) Finish(A,B) → Finish(B,A)
9) Start(A,B) → Start(B,A)
10) Before(A,C), During(B,C) → Before(A,B)
11) Before(C,A), During(B,C) → Before(B,A)

12) Before(A,B), Equal(B,C) → Before(A,C)

13) Before(A,B), Equal(A,C) → Before(C,B)

14) Before(A,B), Finish(A,C) → Before(C,B)

15) Before(B,C), Meet(A,B) → Before(A,C)

16) Before(C,A), Meet(A,B) → Before(C,B)

17) Before(A,B), Overlap(B,C) → Before(A,C)

18) Before(A,B), Overlap(C,A) → Before(C,B)

19) Before(A,B), Start(B,C) → Before(A,C)

20) During(A,B), Equal(A,C) → During(C,B)

21) During(A,B), Equal(B,C) → During(A,C)

22) During(C,B), Meet(A,B) → Before(A,C)

23) During(C,A), Meet(A,B) → Before(C,B)

24) Equal(A,B), Finish(B,C) → Finish(A,C)

25) Equal(A,C), Meet(A,B) → Meet(C,B)

26) Equal(B,C), Meet(A,B) → Meet(A,C)

27) Equal(A,B), Overlap(B,C) → Overlap(A,C)

28) Equal(A,C), Overlap(B,C) → Overlap(B,A)

29) Equal(A,B), Start(B,C) → Start(A,C)

30) Finish(A,C), Meet(A,B) → Meet(C,B)

31) Finish(A,B), Start(A,B) → Equal(A,B)

32) Meet(A,B), Start(B,C) → Meet(A,C)

33) Meet(A,B), Overlap(C,A) → Before(C,B)

34) Meet(A,B), Overlap(B,C) → Before(A,C)

CHAPTER 3

IMPLEMENTATION ISSUES

This chapter presents the framework suggested by author. Mapping sentences to CGs, mapping sentences with temporal knowledge to CGs, mapping CGs to tables of relation database, mapping tables of relation database to predicates are discussed in Section 3.1 to 3.4. The development of temporal knowledge inference engine is given in Section 3.5.

3.1 MAPPING SENTENCE TO CONCEPTUAL GRAPH

The basic principle in mapping sentence to CG is that content words are map to concept nodes, and function words like prepositions and conjunctions are map to relation nodes. Following are some finer distinctions :

a) Ordinary nouns, verbs, adverbs, and adjectives are mapped to concept node. For example [PERSON : John], [PLAY], and [TENNIS].

b) Proper names map to the reference field of a concept whose type field specifies the type. For example [PERSON : John], [CITY : Bangkok], and [PLACE : ABAC].

c) The symbol # with optional qualifiers is used in the reference field for contextually defined references. For example [PERSON : #John], [CITY : #Bangkok], and [PLACE : #ABAC].

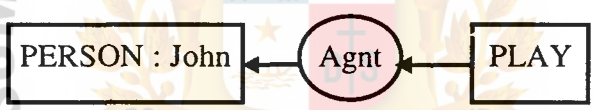
d) Plural nouns are represented by the plural reference {*} followed by an optional count.

e) CR between two concept nodes is connected to concept nodes by the arrows.

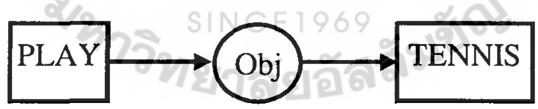
Concept designs a type and a reference, which is either a *specified individual* or an *unspecified individual* or generic of the type. For example [PERSON : "John"] is specified individual, [BUS :*] or [BUS] is unspecified individual.

Example 3.1 Mapping the sentence "John plays tennis." to CG.

1. John is noun, and is a name of person, thus we the concept node PERSON and specify by John. It is [PERSON : John].
2. Play is a verb, it is the concept node [PLAY].
3. Tennis is a noun, it is the concept node [TENNIS].
4. CR PLAY to PERSON with the relation agent, for abbreviate Agnt.



5. CR PLAY to TENNIS with the relation object, for abbreviate Obj.



Combining 4 and 5 we get the complete CG for "John plays tennis.".

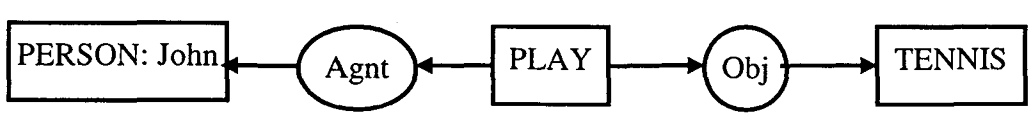


Figure 3.1 : CG for "John plays tennis.".

3.2 MAPPING SENTENCE WITH TEMPORAL KNOWLEDGE TO CONCEPTUAL GRAPH

The mapping a sentence with temporal knowledge to CG can be done through following steps :

1) Decompose the sentence into subclauses.
2) Change each subclauses to CG and map to temporal object.
3) Relate each temporal objects with the temporal relation type (such as BEFORE, AFTER, DURING).

In order to explain the mapping from temporal knowledge to CG, the example are given below :

Example 3.2 The sentence "On July 15,1998, John started sleeping at 4pm during the time Mary watched TV from 3pm to 5pm, after that they had dinner together.".

Step 1 : The sentence can be decomposed into :

- Subclause 1 "John started sleeping.".
- Subclause 2 "Mary watched TV.".
- Subclause 3 "John and Mary had dinner.".

Step 2 : - Change "Subclause 1" to CG and map to temporal object "Sleeping",

- change "Subclause 2" to CG and map to temporal object "Watching", and
- change "Subclause 3" to CG and map to temporal object "Having_dinner".

Step 3 : Relate temporal object "Sleeping" and "Watching" with temporal relation DURING. Temporal object "Sleeping" represent BT by [BT : July 15,1998 (16.00)]. Temporal object "Watching" represent BT by [BT : July 15,1998 (15.00)] and ET by [ET : July 15,1998 (17.00)].

Step 4 : Relate temporal object "Watching" and "Having_dinner" with temporal relation BEFORE.

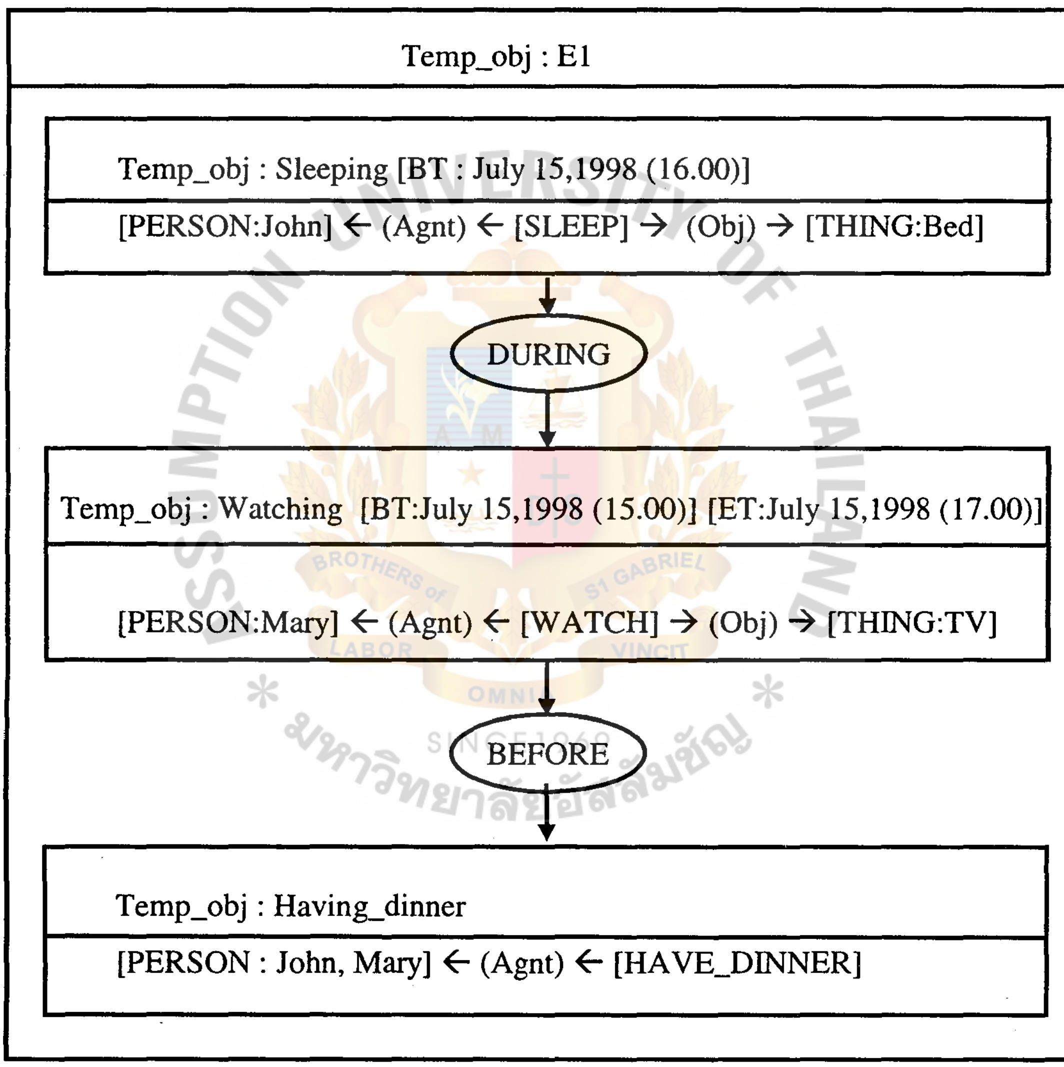


Figure 3.2 : CG of "On July 15,1998, John started sleeping at 4pm during the time Mary watched TV from 3pm to 5pm, after that they had dinner together.".

3.3 MAPPING CONCEPTUAL GRAPH TO TABLES OF RELATION DATABASE

All graph-storing tables in our system can be shown in the following :

1) The conceptual table, the domains common to all uses are :

- TYPE	The concept from and to position
- INDIVIDUAL	The reference of concept

2) The action table, the common domains are :

- FROM	The reference of the concept in the from position
- ACTION	The concept refer to action name
- TO	The reference of the concept in the to position
- EVENT	The event name

3) The temporal relation table, the common domains are :

- TYPE	The temporal relation : after, before, and during
- EVENT_1	The event name
- EVENT_2	The event name

4) The time table, the common domains are :

- EVENT	The event name
- BT	Begin time of event
- ET	End time of event
- DT	Duration Time of event

When we mention the FROM and TO positions, these refer to the concepts attached to a relation; the FROM position is occupied by a concept which points to the relation, and the TO position is occupied by a concept which the relation points to.

Example 3.3 Consider the CG given in Fig 3.2 that presented "On July 15,1998, John started sleeping at 4pm during the time Mary watched TV from 3pm to 5pm, after that they had dinner together." can be mapped to RDB as Fig 3.3.

CONCEPTUAL TABLE :

TYPE	INDIVIDUAL
Person	John
Person	Mary
Thing	Bed
Thing	TV

ACTION TABLE:

FROM	ACTION	TO	EVENT
John	Sleep	Bed	Sleeping
Mary	Watch	TV	Watching
John, Mary	. Have_dinner	-	Having_dinner

TEMPORAL RELATION TABLE :

TYPE	EVENT_1	EVENT_2
During	Sleeping	Watching
Before	Watching	Having_dinner

TIME TABLE :

EVENT	BT	ET	DT
Watching	July 15,1998 (15.00)	July 15,1998 (17.00)	-
Sleeping	July 15,1998 (16.00)	-	-

Figure 3.3 : RDB for "On July 15,1998, John started sleeping at 4pm during the time Mary watched TV from 3pm to 5pm, after that they had dinner together."

Example 3.4 Mapping the CG given in Fig 2.11 that represented "John played tennis on December 10,1997 from 9:00 to 9:30 after he had read a cartoon-book" to RDB as Fig 3.4.

CONCEPTUAL TABLE :

TYPE	INDIVIDUAL
Person	John
Sport	Tennis
Book	Cartoon-Book

ACTION TABLE:

FROM	ACTION	TO	EVENT
John	Play	Tennis	Playing
John	Read	Cartoon-Book	Reading

TEMPORAL RELATION TABLE:

TYPE	EVENT_1	EVENT_2
After	Playing	Reading

TIME TABLE :

EVENT	BT	ET	DT
Playing	Dec 10,1997 (9.00)	Dec 10,1997 (9.30)	-

Figure 3.4 : RDB for "John played tennis on December 10,1997 from 9:00 to 9:30 after he had read a cartoon-book".

3.4 MAPPING TABLES OF RELATION DATABASE TO PREDICATES

From tables of relation database that explain in 3.4 can map to following predicates:

1) The conceptual table is mapping to the predicate form as

"concept(TYPE, INDIVIDUAL)"

2) The action table is mapping to the predicate form as

"action(FROM, ACTION, TO, EVENT)"

3) The temporal relation table is mapping to the predicate form as

"TYPE(EVENT_1,EVENT_2)"

4) The time table is mapping to the predicate form as

"event(EVENT, BT, ET, DT)"

<u>**Example 3.5**</u> Mapping RDB given in Fig 3.3 to predicates as follow:

concept(person, john)

concept(person, mary)

concept(thing, bed)

concept(thing, tv)

action(john, sleep, bed, sleeping)

action(mary, watch, tv, watching)

action(john, have_dinner, -, having_dinner)

action(mary, have_dinner, -, having_dinner)

during(sleeping, watching)

before(watching, having_dinner)

event(watching,bt(1998,07,15,15,00), et(1998,07,15,17,00), DT)

event(sleeping,bt(1998,07,15,16,00), ET, DT)

Example 3.6 Mapping RDB given in Fig 3.4 to predicates as follow:

concept(person, john)

concept(sport, tennis)

concept(book, cartoon)

action(john, play, tennis, playing)

action(john, read, cartoon, reading)

after(playing, reading)

event(playing, bt(1997,12,10,09,00), et(1997,12,10,09,30), DT)

3.5 TEMPORAL KNOWLEDGE INFERENCE ENGINE

At a general level, the sequence of events take place when new information is added to the application. This makes the temporal knowledge changing. The result of an evaluation of either an adding or query is known or unknown. If any assertion contains unknown information then this is request to add more necessary new information.

When new information are added to the application, new temporal model that follow from the new information are not generated until query time.

3.5.1 Algorithm for Finding Time

Prototype : *time(Ev1)*

1) If we have the knowledge base as : "event(Ev1, bt(Year, Month, Day, Hour, Minute), et(Year,Month,Day,Hour,Minute), dt(Year,Month,Day,Hour,Minute))".

   If we know only two arguments, a multi-way reasoning allows us to infer the another. For example

   a) BT and ET are known, DT can be computed as : DT = ET – BT.

      It can be represented in the pseudocode rule as :

      event(Ev1, bt(...), et(...), D) → D is et(...) - bt(...)

   b) BT and DT are known, ET can be computed as : ET = BT + DT.

      It can be represented in the pseudocode rule as :

event(Ev1, bt(…), E, dt(…)) → E is bt(…) + dt(…)

It can be written in the following prolog program :-

```
time(Ev1) :- event(Ev1,bt(Yr1,Mo1,Da1,Hr1,Mn1), et(Yr,Mo,Da,Hr,Mn),
                    dt(Yr2,Mo2,Da2,Hr2,Mn2)),
             nonvar(Yr1), nonvar(Yr2), var(Yr),
             Mn is Mn1 + Mn2, Hr is Hr1 + Hr2, Da is Da1 + Da2,
             Mo is Mo1 + Mo2, Yr is Yr1 + Yr2,
             check(Yr,Mo,Da,Hr,Mn),
             display("et(Yr,Mo,Da,Hr,Mn)").
```

Predicate "check" checks the legality of time, for example the value of Hr should be in the range of 0 to 24.

c) ET and DT are known, BT can be computed as : BT = ET - DT.

It can be represented in the pseudocode rule as :

event(Ev1, B, et(…), dt(…)) → B is et(…) - dt(…)

d) If all BT, ET, DT are known, it is necessary to check whether these values are compatible, it means that ET = BT + DT should be satisfied.

e) For an event, the ending time should be larger than the begin time, therefore event(Ev1, bt(…), E, D) → E ≥ bt(…)

This rule can be used to infer the ET of an event when it BT is known.

f) For an event, the begin time should be smaller than the end time, therefore event(Ev1, B, et(…), D) → B ≤ et(…)

This rule can be used to infer the BT of an event when it ET is known.

All of these rules can be applied and resolved the conflicts to find a best reasonable interval for a time variable. For example if we know that

ET > (1990,10,20,12,00)

ET > (1992,1,12,15,30)

ET > (1989,5,20,01,00)

ET < (1998,10,20,12,00) and

ET < (1998,5,10,12,00).

Then the suitable range for ET is from (1992,1,12,15,30) to (1998,5,10,12,00).

Based on (a), (b), and (c), the time of event is known explicitly, if we know any two variables of BT, ET or DT.

2) Temporal knowledge of an event can be derived from temporal knowledge of other events and temporal relationships, as given in table 3.1. In this table a marked data is a known data.

Relation	Event 1		Event 2		Inference Rule
	BT	ET	BT	ET	
1. BEFORE	✓		✓		$BT1 < \underline{ET1} < BT2$
	✓			✓	$BT1 < \underline{ET1} < \underline{BT2} < ET2$
	✓				$BT1 < \underline{ET1} < \underline{BT2} < \underline{ET2}$
		✓		✓	$ET1 < \underline{BT2} < ET2$
		✓			$ET1 < \underline{BT2} < \underline{ET2}$
			✓		$\underline{BT1} < \underline{ET1} < BT2$
				✓	$\underline{BT1} < \underline{ET1} < \underline{BT2} < ET2$
2. DURING	✓				$\underline{BT2} < BT1$
		✓			$ET1 < \underline{ET2}$
			✓		$BT2 < \underline{BT1}$
				✓	$\underline{ET1} < ET2$
	✓			✓	$BT1 < \underline{ET1} < ET2$
		✓	✓		$BT2 < \underline{BT1} < ET1$

Table 3.1 : Inference Rules. The underlined variables are derived variables. (Continued on the next page.)

Relation	Event 1		Event 2		Inference Rule
	BT	ET	BT	ET	
3. EQUAL	✓				BT1 = BT2
			✓		BT1 = BT2
		✓			ET1 = ET2
				✓	ET1 = ET2
4. FINISH		✓			ET1 = ET2
				✓	ET1 = ET2
5. MEET		✓			ET1 = BT2
			✓		ET1 = BT2
6. OVERLAP	✓				BT1 < BT2
		✓			BT2 < ET1 < ET2
			✓		BT1 < BT2 < ET1
				✓	ET1 < ET2
	✓	✓			BT1 < BT2 < ET1
			✓	✓	BT2 < ET1 < ET2
7. START	✓				BT1 = BT2
			✓		BT1 = BT2

Table 3.1 (Continued).

3.5.2 Algorithm for Finding Before Relation

Prototype : *before(Ev1,Ev2)*. This predicate is true if event 1 occurred before event 2. Before predicate can be defined on the comparison between ET of one event with the BT of other event as :

If ET1 < BT2 → Before(Ev1,Ev2)

The before relation is a transitive relation, therefore

Before(A,B), Before(B,C) → Before(A,C)

Other approaches, the relation before can be derive from other relations

After(A,B) → Before(B,A)

Before(A,C), During(B,C) → Before(A,B)

Before(C,A), During(B,C) → Before(B,A)

Before(A,B), Equal(B,C) → Before(A,C)

Before(A,B), Equal(A,C) → Before(C,B)

Before(A,B), Finish(A,C) → Before(C,B)

Before(B,C), Meet(A,B) → Before(A,C)

Before(C,A), Meet(A,B) → Before(C,B)

Before(A,B), Overlap(B,C) → Before(A,C)

Before(A,B), Overlap(C,A) → Before(C,B)

Before(A,B), Start(B,C) → Before(A,C)

During(C,B), Meet(A,B) → Before(A,C)

During(C,A), Meet(A,B) → Before(C,B)

Meet(A,B), Overlap(C,A) → Before(C,B)

Meet(A,B), Overlap(B,C) → Before(A,C)

3.5.3 Algorithm for Finding During Relation

Prototype : *during(Ev1,Ev2)*. This predicate is true if event 1 occurred during the time of event 2. The during relation can be derived on the comparison of BT and ET of two events, or form the transitive property of during as :

If BT1 > BT2 and ET1 < ET2 → During(Ev1,Ev2)

During(A,B), During(B,C) → During(A,C)

During(A,B), Equal(A,C) → During(C,B)

During(A,B), Equal(B,C) → During(A,C)

3.5.4 Algorithm for Finding Equal Relation

Prototype : *equal(Ev1,Ev2)*. This predicate is true if event 1 occurred at the same time of event 2. The equal relation can be derived on the comparison of BT and ET of two events, or form the transitive property of equal, or backward property as :

If BT1 = BT2 and ET1 = ET2 → Equal(Ev1,Ev2)

Equal(A,B), Equal(B,C) → Equal(A,C)

Equal(A,B) → Equal(B,A)

Finish(A,B), Start(A,B) → Equal(A,B)

3.5.5 Algorithm for Finding Finish Relation

Prototype : *finish(Ev1,Ev2)*. This predicate is true if event 1 finished at the same time of event 2. The finish relation can be derived on the comparison of BT and ET of two events, or form the transitive property of finish, or backward property as :

If ET1 = ET2 → Finish(Ev1,Ev2)

Finish(A,B), Finish(B,C) → Finish(A,C)

Finish(A,B) → Finish(B,A)

Equal(A,B), Finish(B,C) → Finish(A,C)

3.5.6 Algorithm for Finding Meet Relation

Prototype : *meet(Ev1,Ev2)*. This predicate is true if ending time of event 1 and beginning time of event 2 are equal. The meet relation can be derived on the comparison of BT and ET of two events as :

If ET1 = BT2 → Meet(Ev1,Ev2)

Equal(A,C), Meet(A,B) → Meet(C,B)

Equal(B,C), Meet(A,B) → Meet(A,C)

Finish(A,C), Meet(A,B) → Meet(C,B)

Meet(A,B), Start(B,C) → Meet(A,C)

3.5.7 Algorithm for Finding Overlap Relation

Prototype : *overlap(Ev1,Ev2)*. This predicate is true if event 1 occurred before event 2 and they overlap. The overlap relation can be derived on the comparison of BT and ET of two events as :

If BT1 < BT2 and BT2 < ET1 < ET2 → Overlap(Ev1,Ev2)

Equal(A,B), Overlap(B,C) → Overlap(A,C)

Equal(A,C), Overlap(B,C) → Overlap(B,A)

3.5.8 Algorithm for Finding Start Relation

Prototype : *start(Ev1,Ev2)*. This predicate is true if event 1 started at the same time of event 2. The start relation can be derived on the comparison of BT and ET of two events, or form the transitive property of start, or backward property as :

If BT1 = BT2 → Start(Ev1,Ev2)

Start(A,B), Start(B,C) → Start(A,C)

Start(A,B) → Start(B,A)

Equal(A,B), Start(B,C) → Start(A,C)

3.5.9 Algorithm for Finding the Topology Order

Prototype : *topology(Order)*. The topology order of events can be obtained through the following step :

1) Find all before relations (before(Ev1,Ev2)) and store them.
2) Count the member of incoming edges for each event. The number of incoming edges is the number of events occurring before the specified event.
3) Find event whose the number of incoming edge equal to 0 and put it to order.
4) Put the first list to the result.
5) Find the relation before(X,Y) and put Y to the order of result.
6) Repeat step (5) until no relation that match before(X,Y).
7) Repeat (4) to (6) with the next list until empty list.

CHAPTER 4

TEMPORAL KNOWLEDGE PROGRAM

This chapter presents the features of the temporal knowledge process program that has been developed. The process how to input, how to infer and how to query the temporal knowledge will be given in detail, through some examples.

4.1 TEMPORAL KNOWLEDGE PROCESS

Temporal knowledge process have the following steps :

1. Input knowledge into CG.
2. Find topological order, temporal relation and time by Prolog program.
3. Display temporal relation and CG by Visual Basic program.

4.2 HOW TO INPUT THE KNOWLEDGE

We can input the knowledge into the application 2 ways :

4.2.1 Input Obtained from Temporal Knowledge Program (Written in Visual Basic)

The menu of the CG program in Visual Basic program is given in Fig 4.1.

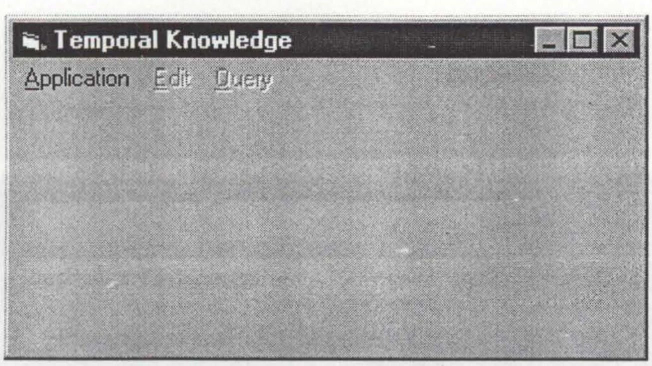


Figure 4.1 : Temporal Knowledge's Menu.

The functions of menu items are explained as follow :

a) Application Menu

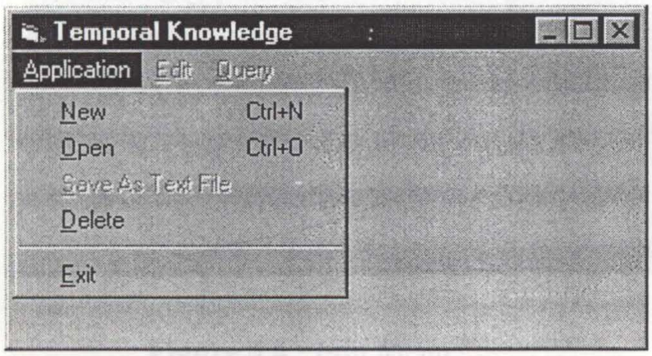


Figure 4.2 : Application Menu.

- New : create new application.
- Open : open the existing application. The existing application will be displayed as in Fig 4.3.
- Save As Text File : save application in text file format.
- Delete : delete the application.
- Exit : exit the program.

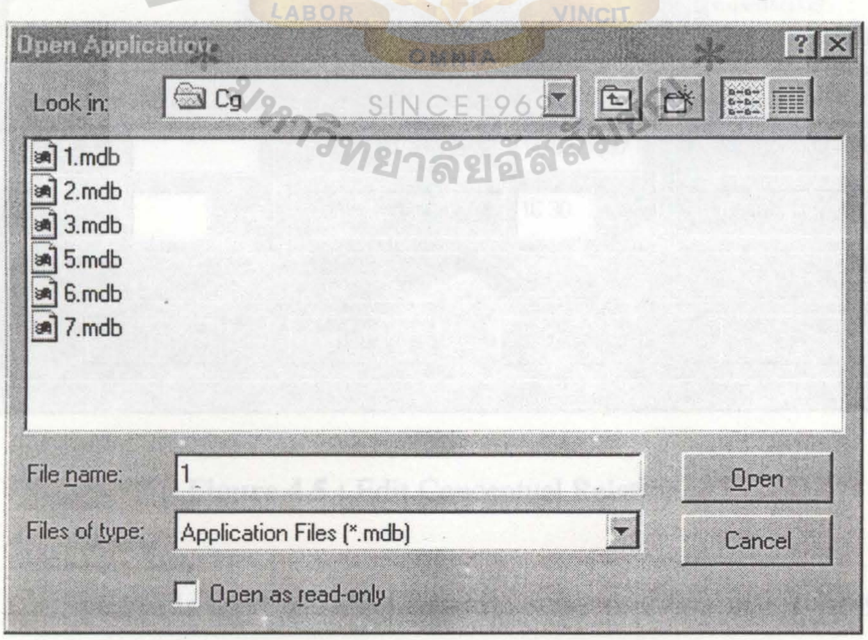


Figure 4.3 : Open Application Dialog Box.

b) Edit Menu

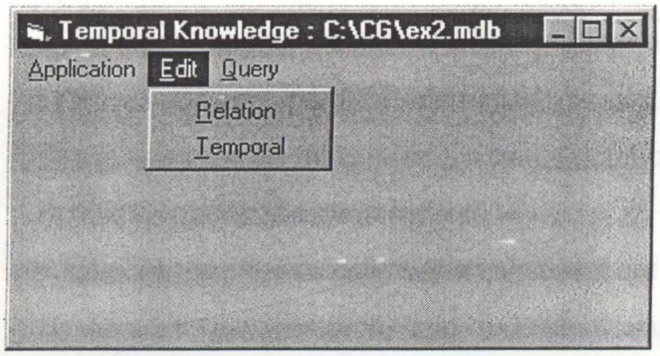


Figure 4.4 : Edit Menu.

- <u>Edit Conceptual Relation</u>

When you selected this command, it will show in Fig 4.5.

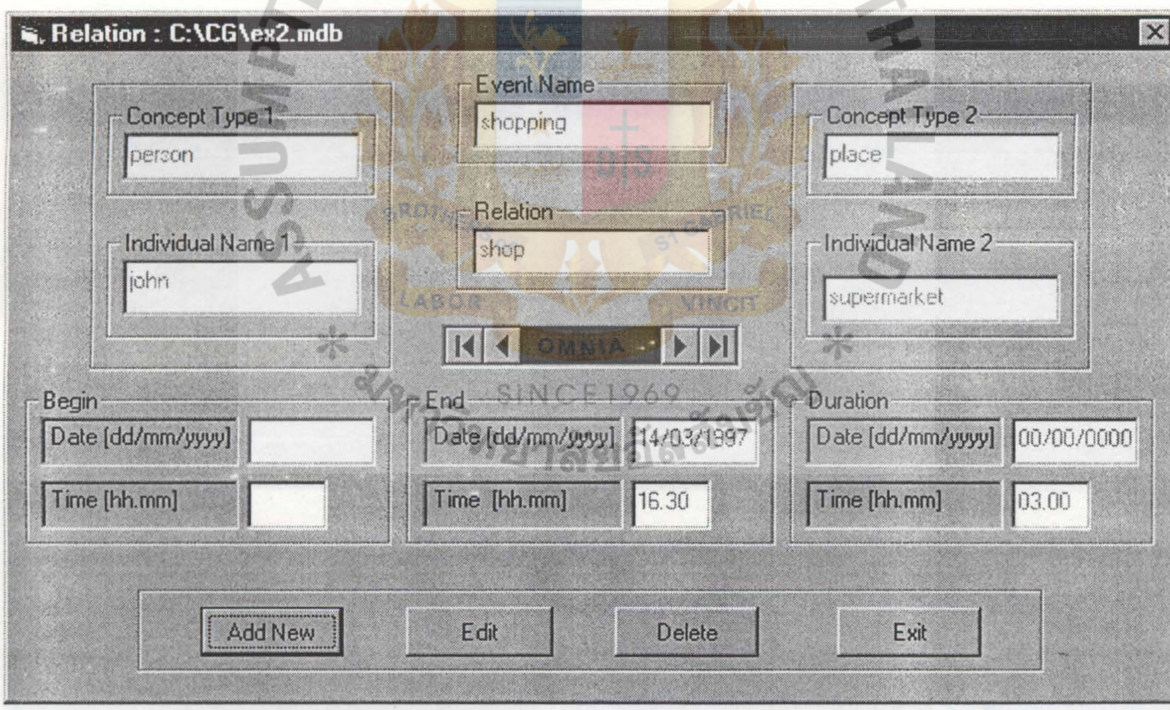


Figure 4.5 : Edit Conceptual Relation.

Mouse is used to select relation. After that you select the following command button.

↳ Add New Button : for adding the new relation to the application. After that, it will show the blank form for input the event_name, concept_type_1, individual_name_1, relation, concept_type_2, individual_name_2, begin_date, begin_time, end_date, end_time, duration_date and duration_time as shown in Fig 4.6.

↳ Edit Button : for modify any attributes about concept such as concept type, relation, and individual name. The old information is displayed, and we can correct it.

↳ Delete Button : for deleting the relation from application. It will show detail about concept type, relation, and individual name.

↳ Exit Button : for leaving the relation submenu, and go back to the main menu.

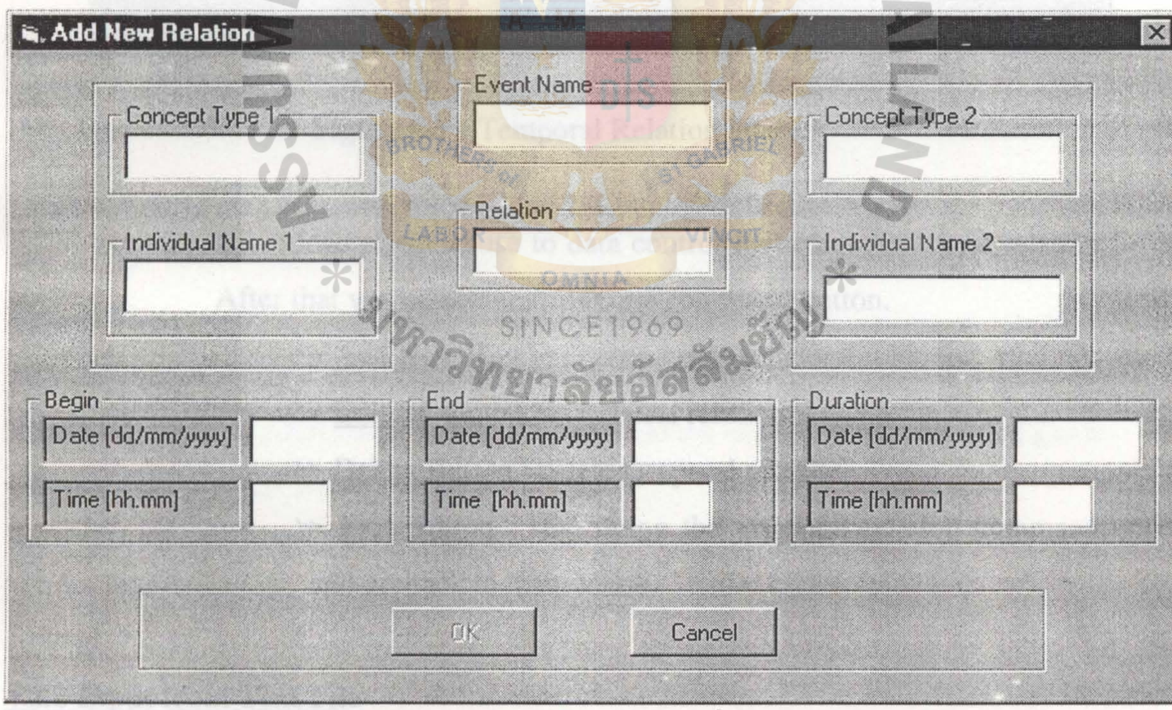


Figure 4.6 : Add New Relation Form.

- Temporal Relation

When this submenu is selected, it will show the temporal relationship between Conceptual relations, for example as in Fig 4.7.

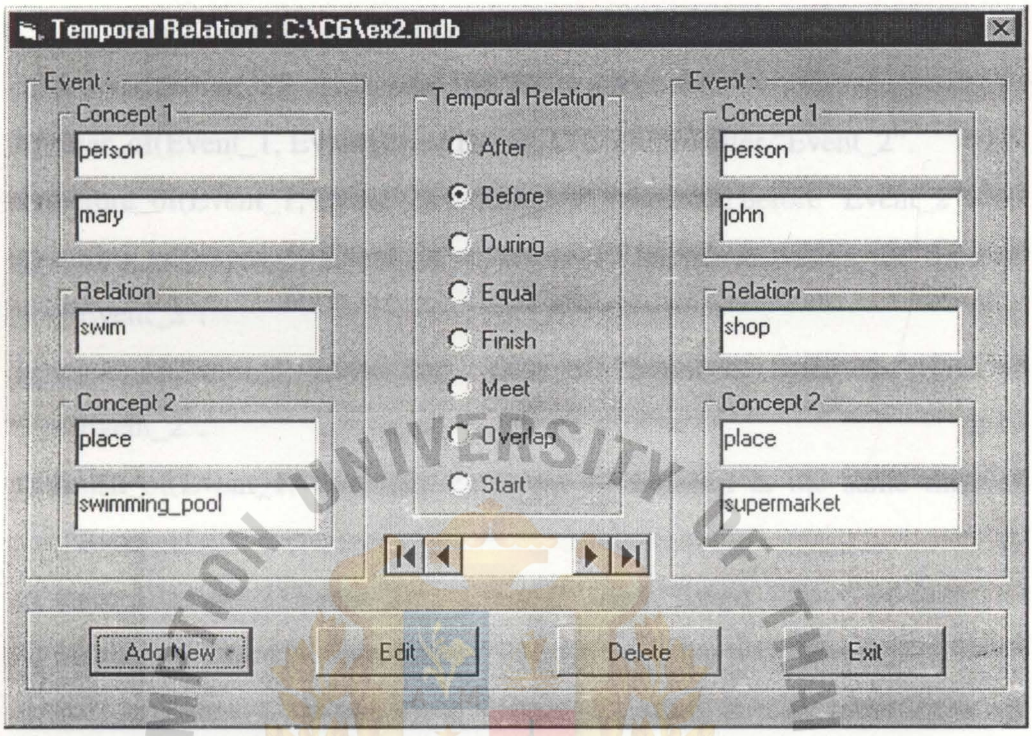


Figure 4.7 : Temporal Relation Form.

You click mouse to data control for select temporal relation. After that you select the following command button.

↪ Add New Button : append new temporal relation.

↪ Edit Button : edit temporal relation.

↪ Delete Button : delete temporal relation.

↪ Exit Button : stop doing the temporal relation command, and go back to main menu.

4.2.2 Input from Text File

The temporal knowledge can be specified in the following forms.

a) event(Event_Name, bt(Year, Month, Day, Hour, Minute), et(Year, Month, Day, Hour,Minute), dt(Year,Month,Day,Hour,Minute)).

The explicit value with specified explicitly. The unknown value will be given as a variable. For example if we know BT and DT of "have_lunch", it can be written as

"event(have_lunch, bt(1998,5,20,12,00), A, dt(0,0,0,1,30))", where A is a variable of ET, this means that ET is unknown.

b) after_of(Event_1, Event_2). : "Event_1" occurred after "Event_2".

c) before_of(Event_1, Event_2). : "Event_1" happened before "Event_2".

d) during_of(Event_1, Event_2). : "Event_1" happened during the time of "Event_2".

e) equal_of(Event_1, Event_2). : time of "Event_1" equal the time of "Event_2".

f) finish_of(Event_1, Event_2). : "Event_1" finished at the same time of "Event_2".

g) meet_of(Event_1, Event_2). : "Event_2" meet "Event_1".

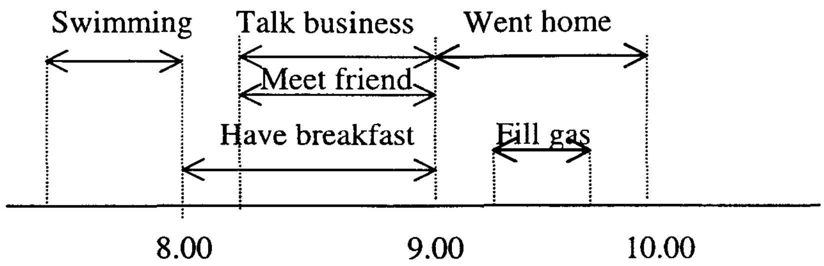
h) overlap_of(Event_1, Event_2). : "Event_1" overlap with "Event_2".

i) start_of(Event_1, Event_2). : "Event_1" start at the same time of "Event_2".

Example 4.1 Suppose that we have the following story :

"On March 14,1997, Mary went swimming in the early morning at YMCA club and she finished swimming at 8.00. After that she had breakfast at the club. While she were having breakfast, she met her friend, John. They had a talk about business until 9.00. After that Mary went home. On the way to home, she filled gas at the petrol station. She arrived home at 10.00.

It can be shown in the following diagram :



We can translate to table as :

CONCEPTUAL TABLE :

TYPE	INDIVIDUAL
Person	John
Person	Mary
Place	Petrol Station
Place	YMCA
Place	Home

ACTION TABLE :

FROM	ACTION	TO	EVENT
Mary	Swim	YMCA	Swimming
Mary	Have_breakfast	YMCA.	Breakfast
Mary	Meet_friend	John	Meeting
Mary	Talk_business	John	Talking
Mary	Go	Home	Going_home
Mary	Fill_gas	Petrol_station	Filling

TEMPORAL RELATION TABLE:

TYPE	EVENT_1	EVENT_2
Meet	Swimming	Breakfast
Finish	Breakfast	Meeting
Finish	Breakfast	Talking
During	Talking	Breakfast
During	Meeting	Breakfast
Meet	Breakfast	Going_home
During	Filling	Going_home

TIME TABLE :

EVENT	BT	ET	DT
Swimming	-	Mar 14,1997 (08.00)	-
Breakfast	-	Mar 14,1997 (09.00)	-
Meeting	-	Mar 14,1997 (09.00)	-
Talking	-	Mar 14,1997 (09.00)	-
Going_home	-	Mar 14,1997 (10.00)	-

Finally, it can be converted to the following facts :

event(swimming, A, et(1997,03,14,08,00), C).

event(breakfast, A, et(1997,03,14,09,00), C).

event(meeting, A, et(1997,03,14,09,00), C).

event(talking, A, et(1997,03,14,09,00), C).

event(going_home, A, et(1997,03,14,10,00), C).

event(filling, A, B, C).

meet_of(swimming, breakfast).

finish_of(breakfast, meeting).

finish_of(breakfast, talking).

during_of(talking, breakfast).

during_of(meeting, breakfast).

meet_of(breakfast, going_home).

during_of(filling, going_home).

concept(person, mary).

concept(place, ymca).

concept(person, john).

concept(place, home).

concept(place, petrol_station).

action([mary],[swim],[ymca],swimming).

action([mary],[have_breakfast],[ymca],breakfast).

action([mary],[meet_friend],[john],meeting).

action([mary],[talk_business],[john],talking).

action([mary],[go],[home],going_home).

action([mary],[fill_gas],[petrol_station],filling).

4.3 HOW TO QUERY THE TEMPORAL KNOWLEDGE

4.3.1 Query from Prolog Program

Main Menu

A - Display All Events
B - Display All Concept Types
C - Display All Before Relation
D - Display All During Relation
E - Display All Equal Relation
F - Display All Finish Relation
G - Display All Meet Relation
H - Display All Overlap Relation
I - Display All Start Relation
J - Find Events After Given Event
K - Find Events Before Given Event
L - Find Time & Concept of an Event
M - Find Topological Order
0 - Exit
Input choice ....

Figure 4.8 : Prolog Program's Menu.

For the temporal knowledge given in Example 4.1, we can have the following queries:

A) Display all Events

Event = [filling, going_home, talking, meeting, breakfast, swimming]

B) Display all Concept Types.

Concept Type = [person, place]

place : [petrol_station, home, ymca]

person : [john, mary]

C) Display All Before Relations

[before(breakfast, going_home), before(swimming, breakfast), before(swimming, going_home), before(meeting, going_home), before(talking, going_home), before(breakfast, filling), before(swimming, filling), before(meeting, filling), before(talking, filling), before(swimming, meeting), before(swimming, talking)]

D) Display All During Relations

[during(filling, going_home),
during(meeting, breakfast),
during(talking, breakfast)]

E) Display All Equal Relations

[]

F) Display All Finish Relations

[finish(talking, breakfast), finish(meeting, breakfast), finish(breakfast, talking), finish(breakfast, meeting)]

G) Display All Meet Relations

[meet(talking, going_home), meet(meeting, going_home), meet(breakfast, going_home), meet(swimming, breakfast)]

H) Display All Overlap Relations

[]

I) Display All Start Relations

[]

J) Find Events After a Given Event

Input event name : meeting.

Event after event “meeting” is [going_home, filling]

K) Find Events before a given Event

Input event name : meeting.

Event before meeting is [swimming]

L) Find Time & Concept of Event

Input event name : meeting.

[mary] [meet_friend] [john]

Begin time before ==> 1997/3/14 9:0

after ==> 1997/3/14 8:0

End time equal ==> 1997/3/14 9:0

M) Find Topological Orders

Topological = [[swimming,breakfast,going_home],
[swimming,breakfast,filling],
[swimming,meeting,going_home],
[swimming,meeting,filling],
[swimming,talking,going_home],
[swimming,talking,filling]]

4.3.2 Query from Visual Basic Program

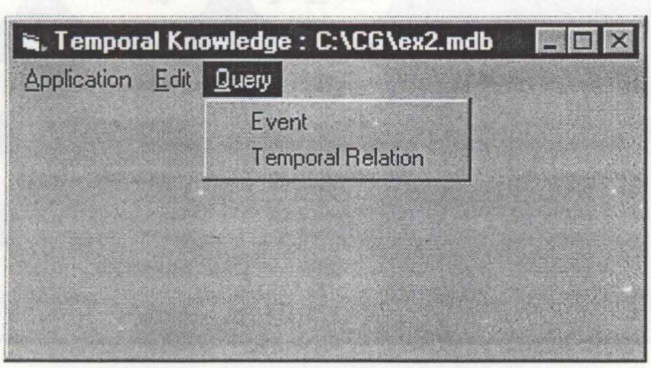


Figure 4.9 : Query Menu.

a) Event submenu

When you select this menu item, it will show the form as in Fig 4.10.

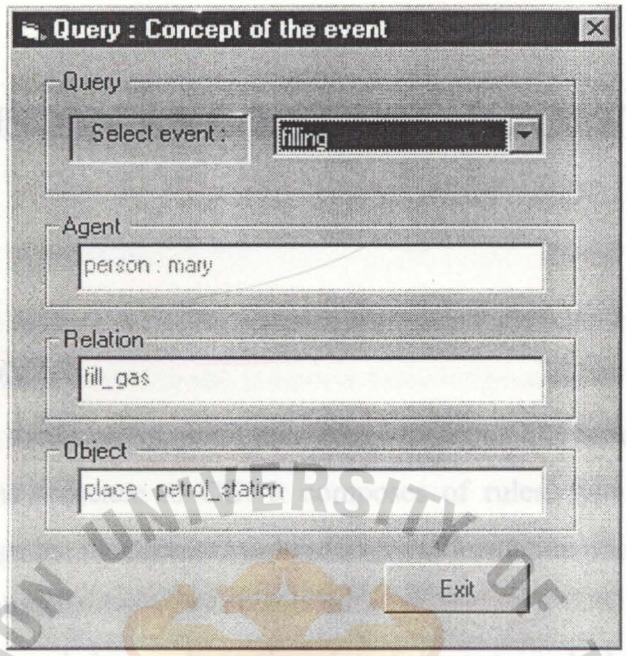


Figure 4.10 : Query Event Item.

b) Temporal Relation

From topology graph, you can find the CG of events by clicking at it's node.

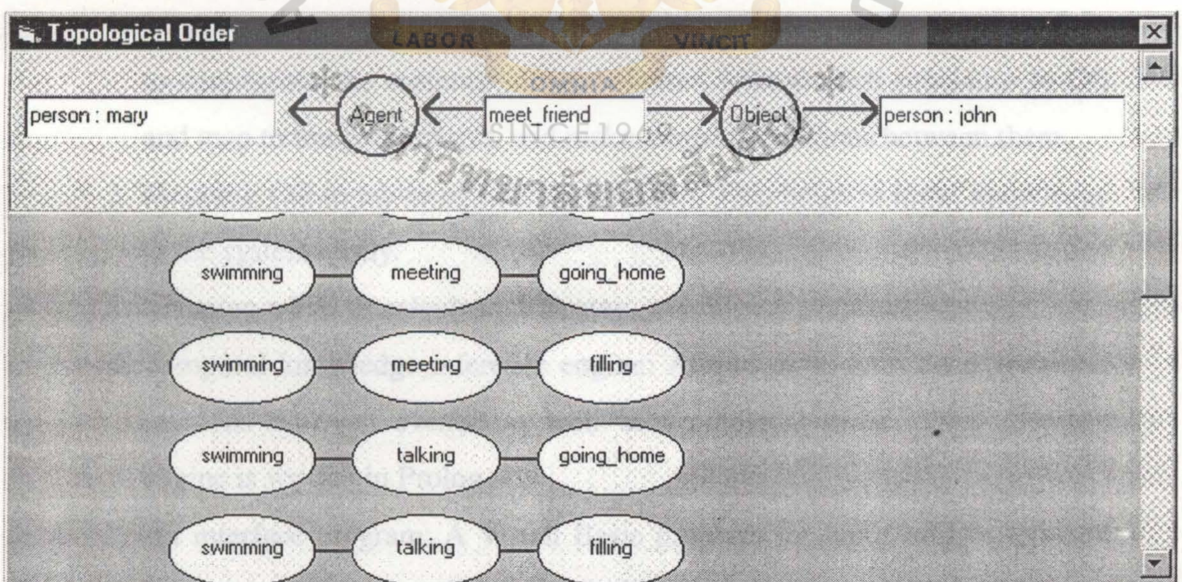


Figure 4.11 : Topology Graph.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION

The following topics on temporal knowledge based on CG have been studied:

1. Mapping the sentence to CG: It composes of rules, type of references, mapping aspects, hierarchical links, canonical formation, maximal join and type hierarchy.
2. Temporal knowledge such as time intervals, temporal objects, temporal relations, and temporal logic: Temporal relation is relation between two temporal objects. Each temporal object can be associated with a time interval.

After that the following topics have been proposed and developed:

1. Mapping the sentence with the temporal knowledge to CG: It shows how to decompose the sentence into subclauses, change each subclause to CG and map to temporal object, and make temporal relations between them.
2. Mapping CG to tables of relation database: This helps to input knowledge to the system easily.
3. Mapping tables to calculus predicates.
4. Temporal knowledge inference engine: Allows us to infer time, temporal relations between events, as well as topological order. This inference engine is written in Prolog.
5. An interface program: A Visual Basic program for input, edit, delete and display temporal knowledge for facilitating the interaction between user and system. This program also displays temporal relations and CGs in graphical mode.

A mapping from sentence to calculus predicate has been studied, the process includes three steps, mapping a sentence to CG, then from CG to tables and finally to predicates. Which the final form, it allows us to make the inference and easily to check the consistency of the system as well as to guarantee constraints, which are set by user.

5.2 RECOMMENDATION

In addition, the demonstration system of temporal knowledge has been developed. It is useful for querying the temporal knowledge such as time of event, temporal relation between events and the topological orders between events. The approaches followed in this study could be adapted to construct a larger temporal knowledge system. Future enhancement to the temporal knowledge system can be developed and listed as follows :

1. It may be difficult to map data in precise form automatically. For example mapping the sentence to CG form automatically can be done by using parsing technique to analyze the structure of the sentence and then map them to CG. The mapping form sentence to CG could be done automatically if an extensive knowledge base is used.
2. In practice, the number of temporal relations can be reduced to three main temporal relations with the following modifications:

   2.1 BEFORE(X,Y) if ET(X) ≤ BT(Y). In this case the meet relation can be considered as a special case of before where ET(X) = BT(Y).

   2.2 DURING(X,Y) if BT(Y) ≤ BT(X) ≤ ET(X) ≤ ET(Y) then EQUAL(X,Y) is a special case of during where BT(Y) = BT(X) ≤ ET(X) = ET(Y).

   2.3 OVERLAP(X,Y) if BT(X) ≤ BT(Y) ≤ ET(X) ≤ ET(Y) then FINISH(X,Y) is a special case of overlap where ET(X) = ET(Y); and START(X,Y) is also a special case of overlap where BT(X) = ET(Y).

Other six inverse relations can be represented by arranging the order of parameters, for example, AFTER(X,Y) ≡ BEFORE(X,Y). Such extension helps to reduce the number of rules for inferring the temporal knowledge enumerously.

REFERENCES

Allen J. F. 1983, "Maintaining Knowledge about Temporal Intervals", "Communications of the ACM", vol. 26 n.11.

Bernard Moulin., 1993, "The Representation of Linguistic Information in An Approach Used for Modeling Temporal Knowledge in Discourses", "Lecture Notes in Artificial Intelligence : Conceptual Graphs for Knowledge Representation", Vol. 699, pp182-204, Germany : Springer-Verlag.

Ellis G., 1991, "Compiled Hierarchical Retrieval", "Proc. of the Sixth Annual Workshop on Conceptual Structures", July, pp187-207.

Fargues, J., M. Landau, A. Duguord, and L. Catach, 1986, "Conceptual Graphs for semantics and knowledge processing", IBM J. Res. Development, Vol. 30, No. 1.

Heike Petermann, 1996, "Natural Language Text Processing and the Maximal Join Operator", "Lecture Notes in Artificial Intelligence : Conceptual Structures : Knowledge Representation as Interlingua", Vol. 1115, pp100-114, Germany : Springer-Verlag.

Jackman, M. K., and C. Pavelin, 1988, "Conceptual Graphs in G. Ringland & D. Duce, eds", "Approaches to knowledge representation", Wiley, New York.

Luger, George F., 1992, "Artificial Intelligence : Structures and Strategies for Complex Problem Solving", 2nd edition. California : Benjamin/Cummings publishing.

Rich, Elaine., 1991, "Artificial Intelligence", 2nd edition. Singapore : McGraw-HILL.

Sait Dogru, and James R. Slagle, 1993, "A System that Translates Conceptual Structures into English", "Lecture Notes in Artificial Intelligence : Conceptual Structures : Theory and Implementation", Vol. 754, pp 283-292, Germany : Springer-Verlag.

Schalkoff, and Robert J., 1990, "Artificial Intelligence : An Engineering Approach", New York : McGraw-HILL.

Sowa, J. F., 1984, "Conceptual Structures : Information Processing in Mind and Machine", Addison Wesley Publishing Company, Inc., Massachusetts.

Sowa, J. F., 1991, "Conceptual Analysis as a Basis for Knowledge Acquisition", "The cognition of experts : Psychological research and empirical AI", Springer-Verlag, Berlin.

Sowa, J. F., 1993, "Relating Diagrams to Logic", "Lecture Notes in Artificial Intelligence : Conceptual Graphs for Knowledge Representation", Vol. 699, pp1-35, Germany : Springer-Verlag.

Turner, Raymond, 1984, "Logics for Artificial Intelligence", Ellis Horwood, Chichester.

APPENDIX A

RULE FOR REASONING TIME

Time(X, T1, T2, unknown) → Time(X, T1, T2, T2-T1)

Time(X, T1, unknown, T3) → Time(X, T1, T1+T3, T3)

Time(X, unknown, T2, T3) → Time(X, T2-T3, T2, T3)

% Before(Event_1, Event_2) %

Before(X, Y) → After(Y, X)

Before(X, Y) ∧ Time(X, _, T2, _) ∧ Time(Y, unknown, _, _) → Time(Y, T2, _, _)

Before(X, Y) ∧ Time(Y, T3, unknown, _) → Time(Y, T3, T3, _)

Before(X, Y) ∧ Time(X, _, T2, _) ∧ Time(Y, unknown, unknown, _) →
    Time(Y, T2, T2, _)

Before(X, Y) ∧ Time(X, unknown, T2, _) → Time(X, T2, T2, _)

Before(X, Y) ∧ Time(X, T1, unknown, _) ∧ Time(Y, unknown, _, _) →
    Time(X, T1, T1, _)

Before(X, Y) ∧ Time(X, _, unknown, _) ∧ Time(Y, T2, _, _) → Time(X, _, T2, _)

% Overlap(Event_1, Event_2) %

Overlap(X, Y) ∧ Time(X, _, unknown, _) ∧ Time(Y, _, T2, _) → Time(X, _, T2, _)

Overlap(X, Y) ∧ Time(X, unknown, _, _) ∧ Time(Y, T2, _, _) → Time(X, T2, _, _)

Overlap(X, Y) ∧ Time(X, T2, _, _) ∧ Time(Y, unknown, _, _) → Time(Y, T2, _, _)

Overlap(X, Y) ∧ Time(X, _, T2, _) ∧ Time(Y, _, unknown, _) → Time(Y, _, T2, _)

% During(Event_1, Event_2) %

During(X, Y) ∧ Time(X, T2, _, _) ∧ Time(Y, unknown, _, _) → Time(Y, T2, _, _)

During(X, Y) ∧ Time(X, _, T2, _) ∧ Time(Y, _, unknown, _) → Time(Y, _, T2, _)

During(X, Y) ∧ Time(X, _, unknown, _) ∧ Time(Y, _, T2, _) → Time(X, _, T2, _)

During(X, Y) ∧ Time(X, unknown, _, _) ∧ Time(Y, T2, _, _) → Time(X, T2, _, _)

APPENDIX B

LISTING OF PROLOG PROGRAM

```
%------------------------------------------------------------------------------------------
before(X,Y) :- after_of(Y,X).
before(X,Y) :- before_of(X,Y).
before(X,Y) :- before_of(X,Z), before(Z,Y).
before(A,B) :- during(B,C), before(A,C).
before(B,A) :- during(B,C), before(C,A).
before(A,C) :- equal(B,C), before_of(A,B).
before(C,B) :- equal(A,C), before_of(A,B).
before(C,B) :- finish(A,C), before_of(A,B).
before(A,C) :- start(B,C), before_of(A,B).
before(A,C) :- overlap(B,C), before(A,B).
before(C,B) :- overlap(C,A), before(A,B).
before(A,C) :- meet(A,B), meet_of(B,C).
before(A,C) :- meet(A,B), before_of(B,C).
before(C,B) :- meet(A,B), before_of(C,A).
before(A,C) :- during(C,B), meet(A,B).
before(C,B) :- during(C,A), meet(A,B).
before(A,C) :- meet(A,B), overlap(B,C).
before(C,B) :- meet(A,B), overlap(C,A).
before(X,Y) :- meet(X,Y).
during(X,Y) :- during_of(X,Y).
during(X,Y) :- during_of(X,Z), during(Z,Y).
during(C,B) :- equal(A,C), during_of(A,B).
during(A,C) :- equal(B,C), during_of(A,B).
```

```
equal(X,Y) :- equal_of(X,Y).
equal(Y,X) :- equal_of(X,Y).
equal(X,Y) :- equal_of(X,Z), equal(Z,Y), not(X=Y).
equal(X,Y) :- finish_of(X,Y), start_of(X,Y).
finish(A,B) :- finish_of(A,B).
finish(A,B) :- finish_of(B,A).
finish(A,B) :- finish_of(A,C), finish(C,B), not(A=B).
finish(A,B) :- equal_of(A,B).
meet(A,B) :- meet_of(A,B).
meet(C,B) :- equal(A,C), meet_of(A,B).
meet(A,C) :- equal(B,C), meet_of(A,B).
meet(C,B) :- finish(A,C), meet_of(A,B).
meet(A,C) :- start(B,C), meet_of(A,B).
overlap(A,B) :- overlap_of(A,B).
overlap(A,C) :- equal(A,B), overlap_of(B,C).
overlap(B,A) :- equal(A,C), overlap_of(B,C).
start(A,B) :- start_of(A,B).
start(B,A) :- start_of(A,B).
start(A,B) :- start_of(A,C), start(C,B), not(A=B).
start(A,B) :- equal_of(A,B).
check(A,B,Mn,R) :- A < B, Mn is A, R is 0.
check(A,B,Mn,R) :- A >= B, Mn is A - B, R is 1.
less(A,B,C,D,E, F,G,H,I,J) :- A < F, !.
less(A,B,C,D,E, F,G,H,I,J) :- A = F, B < G, !.
less(A,B,C,D,E, F,G,H,I,J) :- A = F, B = G, C < H, !.
less(A,B,C,D,E, F,G,H,I,J) :- A = F, B = G, C = H, D < I, !.
less(A,B,C,D,E, F,G,H,I,J) :- A = F, B = G, C = H, D = I, E < J, !.
more(A,B,C,D,E, F,G,H,I,J) :- A > F, !.
more(A,B,C,D,E, F,G,H,I,J) :- A = F, B > G, !.
more(A,B,C,D,E, F,G,H,I,J) :- A = F, B = G, C > H, !.
more(A,B,C,D,E, F,G,H,I,J) :- A = F, B = G, C = H, D > I, !.
```

```
more(A,B,C,D,E, F,G,H,I,J) :- A = F, B = G, C = H, D = I, E > J, !.

show(Relation,Yr,Mo,Da,Hr,Mn) :- Yr > -1, Yr < 9999, nonvar(Yr), write(Relation),
        write('==> '), write(Yr), write('/'), write(Mo), write('/'), write(Da),
        write('  '), write(Hr), write(':'), write(Mn), nl.
not(P) :- P, !, fail.
not(_).
append([],L,L).
append([H|T1],L2,[H|T]) :- append(T1,L2,T).
delete(X,[Y|T],R) :- X = Y, delete(X,T,R).
delete(X,[Y|T],[Y|R]) :- not(X=Y), delete(X,T,R).
delete(X,[],[]).
delete1(be(X,_),[be(X,_)|T],R) :- delete1(be(X,_),T,R).
delete1(be(X,_),[be(A,B)|T],[be(A,B)|R]) :- not(X = A), delete1(be(X,_),T,R).
delete1(be(X,Y),[],[]).
cut_duplicate([X|Y],L) :- member(X,Y), cut_duplicate(Y,L).
cut_duplicate([X|Y],[X|L]) :- not(member(X,Y)), cut_duplicate(Y,L).
cut_duplicate([],[]).
subset1([X|Y],R) :- subset2(X,Y,Y,R).
subset1([],[]).
subset2(X,[Y|Z],L,R) :- subset(X,Y), subset1(L,R).
subset2(X,[Y|Z],L,R) :- not(subset(X,Y)), subset2(X,Z,L,R).
subset2(X,[],L,[X|R]) :- subset1(L,R).
subset([],L).
subset([X|Y],Z) :- member(X,Z), subset(Y,Z).
member(X,[X|_]).
member(X,[Y|Z]) :- member(X,Z).
setof(X,G,L) :- assert(ans([])), G, once(retract(ans(L))), assert(ans([X|L])), fail.
setof(_,_,L) :- retract(ans(L)).
once(P) :- P, !.
```

```
%---------------------------------------------------------------------------------------------------
% find best time for equal
best_time2([t(S2,Yr2,Mo2,Da2,Hr2,Mn2)|L]) :-      S2 = 2, show(' equal '
        ,Yr2,Mo2,Da2,Hr2,Mn2), nl, best_time2([]).
best_time2([t(S2,Yr2,Mo2,Da2,Hr2,Mn2)|L]) :-      not(S2 = 2), best_time2(L).
best_time2([]).
% find best time for less than
best_time1([t(S2,Yr2,Mo2,Da2,Hr2,Mn2)|L],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- S2 = 1,
        less(Yr2,Mo2,Da2,Hr2,Mn2, Yr1,Mo1,Da1,Hr1,Mn1),
        best_time1(L,t(Yr2,Mo2,Da2,Hr2,Mn2)).
best_time1([t(S2,Yr2,Mo2,Da2,Hr2,Mn2)|L],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- S2 = 1,
        not(less(Yr2,Mo2,Da2,Hr2,Mn2, Yr1,Mo1,Da1,Hr1,Mn1)),
        best_time1(L,t(Yr1,Mo1,Da1,Hr1,Mn1)).
best_time1([t(S2,_,_,_,_,_)|L],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- S2 = 3,
        best_time1(L,t(Yr1,Mo1,Da1,Hr1,Mn1)).
best_time1([t(S2,_,_,_,_,_)|L],_) :- S2 = 2.
best_time1([],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- show(' before ',Yr1,Mo1,Da1,Hr1,Mn1),
nl.
% find best time for more than
best_time3([t(S2,Yr2,Mo2,Da2,Hr2,Mn2)|L],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- S2 = 3,
        more(Yr2,Mo2,Da2,Hr2,Mn2, Yr1,Mo1,Da1,Hr1,Mn1),
        best_time3(L,t(Yr2,Mo2,Da2,Hr2,Mn2)).
best_time3([t(S2,Yr2,Mo2,Da2,Hr2,Mn2)|L],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- S2 = 3,
        not(more(Yr2,Mo2,Da2,Hr2,Mn2, Yr1,Mo1,Da1,Hr1,Mn1)),
        best_time3(L,t(Yr1,Mo1,Da1,Hr1,Mn1)).
best_time3([t(S2,_,_,_,_,_)|L],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- S2 = 1,
        best_time3(L,t(Yr1,Mo1,Da1,Hr1,Mn1)).
best_time3([t(S2,_,_,_,_,_)|L],_) :- S2 = 2.
best_time3([],t(Yr1,Mo1,Da1,Hr1,Mn1)) :- show(' after ',Yr1,Mo1,Da1,Hr1,Mn1),
nl.
```

```
%--------------------------------------------------------------------------------------------
% know begin time , end time , find duration time
time(X) :- event(X,bt(Yr1,Mo1,Da1,Hr1,Mn1),et(Yr2,Mo2,Da2,Hr2,Mn2),dt
        (Yr,Mo,Da,Hr,Mn)), nonvar(Yr1), nonvar(Yr2), var(Yr),
        Yr3 is Yr2 - 1 - Yr1,  Mo3 is Mo2 + 11 - Mo1,
        Da3 is Da2 + 29 - Da1,        Hr3 is Hr2 + 23 - Hr1,
        Mn3 is Mn2 + 60 - Mn1,        check(Mn3,60,Mn,R),
        check(Hr3+R,24,Hr,S),         check(Da3+S,30,Da,T),
        check(Mo3+T,12,Mo,U),         Yr is Yr3 + U,
        show('Begin time ',Yr1,Mo1,Da1,Hr1,Mn1),
        show('End time  ',Yr2,Mo2,Da2,Hr2,Mn2),
        show('Duration  ',Yr,Mo,Da,Hr,Mn),!.
% know begin time , duration time , find end time
time(X) :- event(X,bt(Yr1,Mo1,Da1,Hr1,Mn1),et(Yr,Mo,Da,Hr,Mn),dt
        (Yr2,Mo2,Da2,Hr2,Mn2)), nonvar(Yr1), nonvar(Yr2), var(Yr),
        Mn3 is Mn1 + Mn2,  check(Mn3,60,Mn,R),
        Hr3 is Hr1 + Hr2 + R,         check(Hr3,24,Hr,S),
        Da3 is Da1 + Da2 + S,         check(Da3,30,Da,T),
        Mo3 is Mo1 + Mo2 + T,         check(Mo3,12,Mo,U),
        Yr is Yr1 + Yr2 + U,
        show('Begin time ',Yr1,Mo1,Da1,Hr1,Mn1),
        show('End time  ',Yr,Mo,Da,Hr,Mn),
        show('Duration  ',Yr2,Mo2,Da2,Hr2,Mn2),!.
% know end time , duration time , find begin time
time(X) :- event(X,bt(Yr,Mo,Da,Hr,Mn),et(Yr2,Mo2,Da2,Hr2,Mn2),dt
        (Yr1,Mo1,Da1,Hr1,Mn1)), nonvar(Yr1), nonvar(Yr2), var(Yr),
        Yr3 is Yr2 - 1 - Yr1,  Mo3 is Mo2 + 11 - Mo1,
        Da3 is Da2 + 29 - Da1,        Hr3 is Hr2 + 23 - Hr1,
        Mn3 is Mn2 + 60 - Mn1,        check(Mn3,60,Mn,R),
        check(Hr3+R,24,Hr,S),         check(Da3+S,30,Da,T),
        check(Mo3+T,12,Mo,U),         Yr is Yr3 + U,
```

```
        show('Begin time ',Yr,Mo,Da,Hr,Mn),
        show('End time  ',Yr2,Mo2,Da2,Hr2,Mn2),
        show('Duration  ',Yr1,Mo1,Da1,Hr1,Mn1),!.
% know end time, find begin time
time(X) :- event(X,bt(Yr1,Mo1,Da1,Hr1,Mn1),et(Yr2,Mo2,Da2,Hr2,Mn2),_),
        var(Yr1), nonvar(Yr2),  setof(t(S,Yr,Mo,Da,Hr,Mn),
        time1(X,bt(S,Yr,Mo,Da,Hr,Mn)), L),
        write(' Begin time '),   best_time2(L),
        best_time1(L,t(Yr2,Mo2,Da2,Hr2,Mn2)),
        best_time3(L,t(0,0,0,0,0)),
        write(' End time '), show(' equal ',Yr2,Mo2,Da2,Hr2,Mn2),nl,!.
% know begin time, find end time
time(X) :- event(X,bt(Yr1,Mo1,Da1,Hr1,Mn1),et(Yr2,Mo2,Da2,Hr2,Mn2),_),
        nonvar(Yr1), var(Yr2), setof(t(S,Yr,Mo,Da,Hr,Mn),
        time1(X,et(S,Yr,Mo,Da,Hr,Mn)), L),
        write(' Begin time '), show(' equal ',Yr1,Mo1,Da1,Hr1,Mn1),nl,
        write(' End time '), best_time2(L),
        best_time1(L,t(9999,99,99,99,99)),
        best_time3(L,t(Yr1,Mo1,Da1,Hr1,Mn1)),!.
% find begin time, end time
time(X) :- event(X,bt(Yr1,Mo1,Da1,Hr1,Mn1),et(Yr2,Mo2,Da2,Hr2,Mn2),_),
        var(Yr1), var(Yr2), setof(t(S,Yr,Mo,Da,Hr,Mn),
        time1(X,bt(S,Yr,Mo,Da,Hr,Mn)), L),
        setof(t(S,Yr,Mo,Da,Hr,Mn), time1(X,et(S,Yr,Mo,Da,Hr,Mn)), M),
        write(' Begin time '), best_time2(L),
        best_time1(L,t(9999,99,99,99,99)),
        best_time3(L,t(0,0,0,0,0)),
        write(' End time '), best_time2(M),
        best_time1(M,t(9999,99,99,99,99)),
        best_time3(M,t(0,0,0,0,0)),!.
```

```
%-----------------------------------------------------------------------------------------------
% know end time                    find begin time
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- event(X,bt(Yr3,_,_,_,_),et
        (Yr4,Mo4,Da4,Hr4,Mn4),_), nonvar(Yr4), var(Yr3),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% after
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- before(Y,X),
        event(Y,bt(Yr3,Mo3,Da3,Hr3,Mn3),et(Yr4,_,_,_,_),_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr3), var(Yr4), var(Yr5),
        S1 is 3, Yr1 is Yr3, Mo1 is Mo3, Da1 is Da3, Hr1 is Hr3, Mn1 is Mn3.
% after
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- before(Y,X),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 3, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% before
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- before(X,Y),
        event(Y,bt(Yr3,_,_,_,_),et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,bt(Yr5,_,_,_,_),_,_), var(Yr3), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% before
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- before(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% during
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- during(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 3, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
```

```
% during
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- during(X,Y),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% during inverse
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- during(Y,X),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% during inverse
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- during(Y,X),
        event(Y,bt(Yr3,_,_,_,_),et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,bt(Yr5,_,_,_,_),_,_), var(Yr3), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% equal
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- equal(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 2, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% meet
time1(Y,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- meet(X,Y),
        event(X,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(Y,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 2, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% overlap
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- overlap(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
```

```
% overlap
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- overlap(X,Y),
        event(Y,bt(Yr3,_,_,_,_),et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,bt(Yr5,_,_,_,_),_,_), var(Yr3), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% overlap inverse
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- overlap(Y,X),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 3, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% overlap inverse
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- overlap(Y,X),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 1, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
% start
time1(X,bt(S1,Yr1,Mo1,Da1,Hr1,Mn1)) :- start(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,bt(Yr5,_,_,_,_),_,_), nonvar(Yr4), var(Yr5),
        S1 is 2, Yr1 is Yr4, Mo1 is Mo4, Da1 is Da4, Hr1 is Hr4, Mn1 is Mn4.
%------------------------------------------------------------------------------------------------
% know begin time                find end time
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :-
        event(X,bt(Yr4,Mo4,Da4,Hr4,Mn4),et(Yr6,_,_,_,_),_),
        nonvar(Yr4), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% after
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- before(Y,X),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
```

```
% after
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- before(Y,X),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),et(Yr5,_,_,_,_),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr5), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% before
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- before(X,Y),
        event(Y,bt(Yr3,_,_,_,_),et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), var(Yr3), nonvar(Yr4), var(Yr6),
        S2 is 1, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% before
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- before(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 1, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% during
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- during(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% during
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- during(X,Y),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 1, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% during inverse
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- during(Y,X),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
```

```
% during inverse
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- during(Y,X),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),et(Yr5,_,_,_,_),_),
        event(X,_,et(Yr6,_,_,_,_),_), var(Yr5), nonvar(Yr4), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% equal
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- equal(X,Y),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 2, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% finish
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- finish(X,Y),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 2, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% meet
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- meet(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 2, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% overlap
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- overlap(X,Y),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),_,_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% overlap
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- overlap(X,Y),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 1, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
```

```
% overlap inverse
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- overlap(Y,X),
        event(Y,bt(Yr4,Mo4,Da4,Hr4,Mn4),et(Yr5,_,_,_,_),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr5), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% overlap inverse
time1(X,et(S2,Yr2,Mo2,Da2,Hr2,Mn2)) :- overlap(Y,X),
        event(Y,_,et(Yr4,Mo4,Da4,Hr4,Mn4),_),
        event(X,_,et(Yr6,_,_,_,_),_), nonvar(Yr4), var(Yr6),
        S2 is 3, Yr2 is Yr4, Mo2 is Mo4, Da2 is Da4, Hr2 is Hr4, Mn2 is Mn4.
% -------------------------------------------------------------------------------------------
all_before(List) :- setof(be(X,Y),before(X,Y),L), cut_duplicate(L,List).
all_vertex(Vertex) :- setof(X,event(X,_,_,_),N), cut_duplicate(N,Vertex).
find(R) :- all_before(List), all_vertex(Vertex), set(Vertex,Weigh), count_in
(Weigh,List,W_in),
        seek(W_in,S_vertex), order(S_vertex,R2),
        list(R2,R3), cut_duplicate(R3,R4), subset1(R4,R).
set([A|B],[c(A,0)|R]) :- set(B,R).
set([],[]).
list([A|B],S) :- list1(A,B,S).
list([],[]).
list1([A|B],C,[A|S]) :- list1(B,C,S).
list1([],C,S) :- list(C,S).
list1([],[],[]).
cut1([A|B],S) :- cut2(A), cut1(B,S).
cut1([A|B],[A|S]) :- not(cut2(A)), cut1(B,S).
cut1([],[]).
cut2([A|[]]).
order([L|R],[M|T]) :- setof(S,(order1(L,V), append([L],V,S)),M), order(R,T).
order([],[]).
order1(X,[Y|Z]) :- before(X,Y), order1(Y,Z).
```

```
order1(_,[]).
seek([c(A,0)|B],[A|R]) :- seek(B,R).
seek([c(A,Num)|B],R) :- not(Num = 0), seek(B,R).
seek([],[]).
count_in([c(A,Num)|B],List,[c(A,New)|T]) :- count1_in(c(A,Num),List,New),
count_in(B,List,T).
count_in([],_,[]).
count1_in(c(A,Num),[be(_,A)|L],V) :- count1_in(c(A,Num),L,S), V is S + 1.
count1_in(c(A,Num),[be(_,D)|L],V) :- not(A = D), count1_in(c(A,Num),L,S), V = S.
count1_in(c(A,Num),[],0).
choose([X|List]) :- nl, write(X), write(' : '), setof(Y,concept(X,Y),L2),
                    cut_duplicate(L2,List1), write(List1), choose(List).
choose([]).
main :- write(' Input data file name "File name". : '), read(Name), reconsult(Name),
        find(T), tell('c:\cg\order.out'), write(T), told, menu.
menu :- nl, nl, write('    Main Menu '), nl,
        write('  a - Display all Events '), nl,
        write('  b - Display all Concept Types '), nl,
        write('  c - Display all before relation '), nl,
        write('  d - Display all during relation '), nl,
        write('  e - Display all equal relation '), nl,
        write('  f - Display all finish relation '), nl,
        write('  g - Display all meet relation '), nl,
        write('  h - Display all overlap relation '), nl,
        write('  i - Display all start relation '), nl,
        write('  j - Find Events after given Event '), nl,
        write('  k - Find Events before given Event '), nl,
        write('  l - Find Time & Concept of an Event '), nl,
        write('  m - Find Topological Order'), nl,
        write('  0 - Exit'), nl, nl,
        write('       Input choice .......... '), read(X), nl,
```

```
        menu(X).
menu(a) :- all_vertex(Event), write(' Event = '), write(Event), nl, menu.
menu(b) :- setof(X,concept(X,Y),L1), cut_duplicate(L1,List1), write(List1),choose
(List1), menu.
menu(c) :- setof(before(X,Y),before(X,Y),L), cut_duplicate(L,R), write(R), menu.
menu(d) :- setof(during(X,Y),during(X,Y),L), cut_duplicate(L,R), write(R), menu.
menu(e) :- setof(equal(X,Y),equal(X,Y),L), cut_duplicate(L,R), write(R), menu.
menu(f) :- setof(finish(X,Y),finish(X,Y),L), cut_duplicate(L,R), write(R), menu.
menu(g) :- setof(meet(X,Y),meet(X,Y),L), cut_duplicate(L,R), write(R), menu.
menu(h) :- setof(overlap(X,Y),overlap(X,Y),L), cut_duplicate(L,R), write(R), menu.
menu(i) :- setof(start(X,Y),start(X,Y),L), cut_duplicate(L,R), write(R), menu.
menu(j) :- write(' Input event name : '), read(Q), nl, setof(T,before(Q,T),L),
        cut_duplicate(L,List), write('Event after '), write(Q), write(' is '), write(List),
menu.
menu(k) :- write(' Input event name : '), read(Q), nl, setof(T,before(T,Q),L),
        cut_duplicate(L,List), write('Event before '), write(Q), write(' is '), write(List),
menu.
menu(l) :- write(' Input event name : '), read(Q), nl, action(A,B,C,Q), write(A),
        write('  '), write(B), write('  '), write(C), nl, time(Q), menu.
menu(m) :- find(R), write('Order = '), write(R), menu.
menu(0).
```

APPENDIX C

LISTING OF VISUAL BASIC PROGRAM

```
% --------------------------------------- Form Menu ------------------------------------------ %
Dim ArrayNum As Integer
Private Sub Form_Load()
        Dim Error_ans As Integer
        ArrayNum = 0
End Sub

Private Sub UpdateMenu()
        mnu_App_Array(0).Visible = True
        ArrayNum = ArrayNum + 1
        For ii = 1 To ArrayNum
                If mnu_App_Array(ii).Caption = Filename Then
                        ArrayNum = ArrayNum - 1
                        Exit Sub
                End If
        Next ii
        If ArrayNum >= 5 Then
                For ii = 1 To ArrayNum
                        mnu_App_Array(ii).Caption = mnu_App_Array(ii +
                        1).Caption
                Next ii
                ArrayNum = ArrayNum - 1
        End If
        mnu_App_Array(ArrayNum).Caption = Filename
```

```
        mnu_App_Array(ArrayNum).Visible = True
End Sub

Private Sub mnu_App_Array_Click(Index As Integer)
        If Index >= 0 Then
                Filename = mnu_App_Array(Index).Caption
                FormRelation.Show 1
        End If
End Sub

Private Sub mnu_App_item_Click(Index As Integer)
        Dim DesFilename As String
        Dim dbs As Database
        Dim rst, rst2 As Recordset
        On Error GoTo errhandler
        Select Case Index
        Case 0  ' New Application
                CmDialog1.DialogTitle = "New Application"
                CmDialog1.Filter = "All Files (*.*)|*.*|Application Files
        (*.mdb)|*.mdb"
                CmDialog1.FilterIndex = 2
                CmDialog1.Action = 1
                Filename = CmDialog1.Filename
                result = OpenFile(Filename, NEW_APPLICATION)
                If result = NEW_DB Then
                                mnu_App_item(2).Enabled = True
                                mnu_Edit.Enabled = True
                                mnu_Query.Enabled = True
                                FormMenu.Caption = "Temporal Knowledge : " +
        Filename
                                Call NewDatabase
```

```
                    Call UpdateMenu
                    FormRelation.Show 1
          End If
Case 1  ' Open application
          CmDialog1.DialogTitle = "Open Application"
          CmDialog1.Filter = "All Files (*.*)|*.*|Application Files
(*.mdb)|*.mdb"
          CmDialog1.FilterIndex = 2
          CmDialog1.Action = 1
          Filename = CmDialog1.Filename
          If FileLen(Filename) < 0 Then MsgBox ("File Not Found")
          Call UpdateMenu
          FormMenu.Caption = "Temporal Knowledge : " + Filename
          Set dbs = OpenDatabase(Filename)
          Set rst = dbs.OpenRecordset("TRelation")
          If rst.RecordCount > 0 Then
                    mnu_App_item(2).Enabled = True
                    mnu_Edit.Enabled = True
                    mnu_Query.Enabled = True
                    rst.Close
                    dbs.Close
          Else
                    rst.Close
                    dbs.Close
                    FormRelation.Show 1
          End If
Case 2  ' Save As Text File
          CmDialog1.DialogTitle = "Save Application As Text File For Prolog
Program"
          CmDialog1.Filter = "All Files (*.*)|*.*|Text Files (*.txt)|*.txt"
          CmDialog1.FilterIndex = 2
```

```
CmDialog1.Action = 2
Dim FileNumber, bt, et, dt
Dim first_relation, second_relation, name1, name2
FileNumber = FreeFile
Open CmDialog1.Filename For Output As #FileNumber
Set dbs = OpenDatabase(Filename)
Set rst = dbs.OpenRecordset("TRelation")
Set rst2 = dbs.OpenRecordset("TTemporal")
rst.MoveFirst
Do While Not rst.EOF
        If (rst.Fields(4) <> " ") And (rst.Fields(5) <> " ") Then
        bt = ",bt(" & Right(rst.Fields(4), 4) & "," & _
        Mid(rst.Fields(4), 4, 2) & "," & Left(rst.Fields(4), 2) &
        Left(rst.Fields(5), 2) & "," & Right(rst.Fields(5), 2)
        Else
        bt = ",A"
        End If
        If (rst.Fields(6) <> " ") And (rst.Fields(7) <> " ") Then
        et = ",et(" & Right(rst.Fields(6), 4) & "," & _
        Mid(rst.Fields(6), 4, 2) & "," & Left(rst.Fields(6), 2) &
        Left(rst.Fields(7), 2) & "," & Right(rst.Fields(7), 2)
    Else
        et = ",B"
        End If
        If (rst.Fields(8) <> " ") And (rst.Fields(9) <> " ") Then
        dt = ",dt(" & Right(rst.Fields(8), 4) & "," & _
        Mid(rst.Fields(8), 4, 2) & "," & Left(rst.Fields(8), 2) &
        Left(rst.Fields(9), 2) & "," & Right(rst.Fields(9), 2) &
    Else
        dt = ",C"
        End If
```

```
            Print #1, "event("; rst.Fields(11); bt; et; dt; ")."
            rst.MoveNext
    Loop
    Do While Not rst2.EOF
            rst.MoveFirst
            Do While Not rst.EOF
            If rst2.Fields(0) = rst.Fields(11) Then
                    first_relation = rst.Fields(11)
                    Exit Do
            Else
                    rst.MoveNext
            End If
            Loop
            rst.MoveFirst
            Do While Not rst.EOF
            If rst2.Fields(1) = rst.Fields(11) Then
                    second_relation = rst.Fields(11)
                    Exit Do
            Else
                    rst.MoveNext
            End If
            Loop
            Print #1,LCase(rst2.Fields(2));"_of(";first_relation;",";
second_relation; ")."
            rst2.MoveNext
    Loop
    rst.MoveFirst
    Do While Not rst.EOF
            name1 = Trim(LCase(rst.Fields(1)))
            If (rst.Fields(0) <> " ") Then
            Do While InStr(name1, ",")
```

```
            name2 = Trim(Left(name1, InStr(name1, ",")
            name1 = Mid(name1, InStr(name1, ",") + 1)
            Do While InStr(name2, " ")
                Mid(name2, InStr(name2, " "), 1) = "_"
            Loop
            Print #1,"concept("; rst.Fields(0); name2; ")."
        Loop
        name1 = Trim(name1)
        Do While InStr(name1, " ")
            Mid(name1, InStr(name1, " "), 1) = "_"
        Loop
        Print #1, "concept("; rst.Fields(0); ","; (name1); ")."
        End If
        name1 = Trim(LCase(rst.Fields(3)))
        If (rst.Fields(2) <> " ") Then
        Do While InStr(name1, ",")
            name2 = (Left(name1, InStr(name1, ",") - 1))
            name1 = Mid(name1, InStr(name1, ",") + 1)
            Do While InStr(name2, " ")
                Mid(name2, InStr(name2, " "), 1) = "_"
            Loop
            Print #1, "concept("; rst.Fields(2); name2; ")."
        Loop
        name1 = Trim(name1)
        Do While InStr(name1, " ")
            Mid(name1, InStr(name1, " "), 1) = "_"
        Loop
        Print #1, "concept("; rst.Fields(2); Trim(name1); ")."
        End If
        rst.MoveNext
Loop
```

```
rst.MoveFirst
Do While Not rst.EOF
    Print #1, "action(["; LCase(rst.Fields(1)); "],["; LCase
    (rst.Fields(10)); "],["; LCase(rst.Fields(3)); "],"; LCase
    (rst.Fields(11)); ")."
        rst.MoveNext
Loop
rst.Close
rst2.Close
dbs.Close
Close #FileNumber
Case 3  ' Delete
    CmDialog1.DialogTitle = "Delete Application"
    CmDialog1.Filter = "All Files (*.*)|*.*|Application Files
(*.mdb)|*.mdb"
    CmDialog1.FilterIndex = 2
    CmDialog1.Action = 1
    Filename = CmDialog1.Filename
    If FileLen(Filename) > 0 Then       Kill Filename
Case 4  'Line
Case 5  'Exit
    End
Case 6  'Line
End Select
FormMenu.Show
Exit Sub
errhandler:
    Error_ans = MsgBox(Error, 48, "Error!")
    Exit Sub
End Sub
```

```
Private Function OpenFile(NewFilename As String, Mode As Integer) As Integer
        Dim NewFileNum, a As Integer
        Dim Msg As String
        If NewFilename Like "*[;-?[* ]*" Or NewFilename Like "*]*" Then Error
Err_BadFileName
        If Mode = NEW_APPLICATION Then
                If Dir(NewFilename) <> "" Then
                        Msg = "Do you want to replace " + NewFilename + " file?"
                        If MsgBox(Msg, 49, "Replace File?") = 2 Then
                                OpenFile = 0
                                Exit Function
                        Else 'Replace DB
                                Kill NewFilename
                                OpenFile = NEW_DB
                                Exit Function
                        End If
                Else
                                OpenFile = NEW_DB
                                Exit Function
                End If
        End If
        If Mode = DELETE_APPLICATION Then
                        NewFileNum = FreeFile
                        Open NewFilename For Random As NewFileNum
                        If LOF(NewFileNum) = 0 Then
                        Msg = "File " + NewFilename + " does not exist. "
                        a = MsgBox(Msg, 64, "File Not Found")
                        End If
                        Close NewFileNum
                        Kill NewFilename
                        OpenFile = 0
```

```
            Exit Function
      End If
End Function

Private Sub NewDatabase()
      Dim wrkDefault As Workspace
      Dim dbsNew As Database
      Dim prpLoop As Property
      Set wrkDefault = DBEngine.Workspaces(0)
      Set dbsNew = wrkDefault.CreateDatabase(Filename, dbLangGeneral)
      dbsNew.Execute "CREATE TABLE TRelation " _
      & "(Concept_Type_1 Text (20), Indv_Name_1 Text (20), Concept_Type_2
      Text (20),"_ & "Indv_Name_2 Text (20), Begin_Date Text (10) ,
      Begin_Time Text (5), " _
      & "End_Date Text (10), End_Time Text (5), Duration_Date Text (10), "_
      & "Duration_Time Text (5), Relation Text (20) Not Null, Event_Name Text
(20) " _
      & "CONSTRAINT Event_Ind Primary Key);"
      dbsNew.Execute "CREATE TABLE TTemporal " _
      & "(Relation_1 Text (20), Relation_2 Text (20), Temporal_Relation Text
(10), " _
      & "Temporal_Id Integer CONSTRAINT Temporal_Ind PRIMARY KEY);"
      dbsNew.Close
End Sub

Private Sub mnu_Edit_item_Click(Index As Integer)
      On Error GoTo err_handler
      Select Case Index
            Case 0  ' Edit relation
                        FormMenu.Caption = "Temporal Knowledge : " +
Filename
```

```
                    FormRelation.Show 1
          Case 1  ' Edit temporal
                    FormMenu.Caption = "Temporal Knowledge : " +
Filename
                    FormTemporal.Show 1
     End Select
     Exit Sub
err_handler:
     Error_ans = MsgBox(Error, 48, "Error!")
     Exit Sub
End Sub


Private Sub mnu_Query_item_Click(Index As Integer)
     On Error GoTo err_handler
     Select Case Index
          Case 0  ' Event
                    FormEvent.Show 1
          Case 1  ' Temporal Relation
                    FormTopological.Show
     End Select
     Exit Sub
err_handler:
     Error_ans = MsgBox(Error, 48, "Error!")
     Exit Sub
End Sub

 % ---------------------------- Form Add Temporal -------------------------------------- %
Private AddMode As Integer
Private Error_ans As Integer
Private MaxNum As Integer
```

```
Private Sub CancelButton_Click()
        Unload FormAddTemporal
        If Data3.Recordset.RecordCount = 0 Then
                FormTemporal.Visible = False
                FormMenu.Visible = True
        Else
                FormTemporal.Visible = True
        End If
End Sub

Private Sub Data1_Reposition()
        Frame_Event_1.Caption = "Event : " + Data1.Recordset("Event_Name")
End Sub
Private Sub Data2_Reposition()
        Frame_Event_2.Caption = "Event : " + Data2.Recordset("Event_Name")
End Sub

Private Sub Form_Load()
        Data1.DatabaseName = Filename
        Data2.DatabaseName = Filename
        Data3.DatabaseName = Filename
        If Modify_Rec = -1 Then
                FormAddTemporal.Caption = "Add New Temporal"
                Data1.Refresh
                Data2.Refresh
                Data3.Refresh
                If Data3.Recordset.RecordCount = 0 Then
                                MaxNum = 0
                Else
                                Data3.Recordset.MoveLast
                                MaxNum = Data3.Recordset("Temporal_Id")
```

```
            End If
            Data3.Recordset.AddNew
            Opt_After.Value = True
        Else
            FormAddTemporal.Caption = "Edit Temporal"
            Data3.Refresh
            Data3.Recordset.Index = "Temporal_Ind"
            Data3.Recordset.Seek "=", Modify_Rec
            Select Case Data3.Recordset.Fields("Temporal_Relation")
                    Case "After"   Opt_After.Value = True
                    Case "Before"  Opt_Before.Value = True
                    Case "During"  Opt_During.Value = True
                    Case "Equal"   Opt_Equal.Value = True
                    Case "Finish"  Opt_Finish.Value = True
                    Case "Meet"    Opt_Meet.Value = True
                    Case "Overlap" Opt_Overlap.Value = True
                    Case "Start"   Opt_Start.Value = True
            End Select
            Data1.Refresh
            Data1.Recordset.Index = "Event_Ind"
            Data1.Recordset.Seek "=", Data3.Recordset("Relation_1")
            Data2.Refresh
            Data2.Recordset.Index = "Event_Ind"
            Data2.Recordset.Seek "=", Data3.Recordset("Relation_2")
            Data3.Recordset.Edit
        End If
End Sub

Private Sub OKButton_Click()
        Dim F_YR, F_MO, F_DA, F_HH, F_MM, S_YR, S_MO, S_DA, S_HH,
S_MM
```

```
    Dim Error_Msg
    On Error GoTo NoUpdate
    If Modify_Rec = -1 Then Data3.Recordset("Temporal_Id") = MaxNum + 1
    Data3.Recordset("Relation_1") = Event_1.Text
    Data3.Recordset("Relation_2") = Event_2.Text
    If (Begin_Date_1.Text = "") And (End_Date_1.Text = "") Then GoTo
Update_Line
    If (Begin_Date_2.Text = "") And (End_Date_2.Text = "") Then GoTo
Update_Line
    If Begin_Date_1.Text <> "" Then
        F_YR = Val(Right(Begin_Date_1.Text, 4))
        F_MO = Val(Mid(Begin_Date_1.Text, 4, 2))
        F_DA = Val(Left(Begin_Date_1.Text, 2))
        F_HH = Val(Left(Begin_Time_1.Text, 2))
        F_MM = Val(Mid(Begin_Time_1.Text, 4, 2))
    End If
    If End_Date_1.Text <> "" Then
            F_YR = Val(Right(End_Date_1.Text, 4))
            F_MO = Val(Mid(End_Date_1.Text, 4, 2))
            F_DA = Val(Left(End_Date_1.Text, 2))
            F_HH = Val(Left(End_Time_1.Text, 2))
            F_MM = Val(Mid(End_Time_1.Text, 4, 2))
    End If
    If End_Date_2.Text <> "" Then
            S_YR = Val(Right(End_Date_2.Text, 4))
            S_MO = Val(Mid(End_Date_2.Text, 4, 2))
            S_DA = Val(Left(End_Date_2.Text, 2))
            S_HH = Val(Left(End_Time_2.Text, 2))
            S_MM = Val(Mid(End_Time_2.Text, 4, 2))
    End If
    If Begin_Date_2.Text <> "" Then
```

```
            S_YR = Val(Right(Begin_Date_2.Text, 4))
            S_MO = Val(Mid(Begin_Date_2.Text, 4, 2))
            S_DA = Val(Left(Begin_Date_2.Text, 2))
            S_HH = Val(Left(Begin_Time_2.Text, 2))
            S_MM = Val(Mid(Begin_Time_2.Text, 4, 2))
End If
If Opt_After.Value = True Then
        If F_YR < S_YR Then GoTo Error_Line
        If F_YR > S_YR Then GoTo Update_Line
        If F_MO < S_MO Then GoTo Error_Line
        If F_MO > S_MO Then GoTo Update_Line
        If F_DA < S_DA Then GoTo Error_Line
        If F_DA > S_DA Then GoTo Update_Line
        If F_HH < S_HH Then GoTo Error_Line
        If F_HH > S_HH Then GoTo Update_Line
        If F_MM < S_MM Then GoTo Error_Line
        If F_MM > S_MM Then GoTo Update_Line
End If
If Opt_Before.Value = True Then
        If F_YR > S_YR Then GoTo Error_Line
        If F_YR < S_YR Then GoTo Update_Line
        If F_MO > S_MO Then GoTo Error_Line
        If F_MO < S_MO Then GoTo Update_Line
        If F_DA > S_DA Then GoTo Error_Line
        If F_DA < S_DA Then GoTo Update_Line
        If F_HH > S_HH Then GoTo Error_Line
        If F_HH < S_HH Then GoTo Update_Line
        If F_MM > S_MM Then GoTo Error_Line
        If F_MM < S_MM Then GoTo Update_Line
End If
If Opt_During.Value = True Then GoTo Update_Line
```

```
        If Opt_Equal.Value = True Then GoTo Update_Line
        If Opt_Finish.Value = True Then GoTo Update_Line
        If Opt_Meet.Value = True Then GoTo Update_Line
        If Opt_Overlap.Value = True Then GoTo Update_Line
        If Opt_Start.Value = True Then GoTo Update_Line
Update_Line:
        If Opt_After.Value = True Then Data3.Recordset("Temporal_Relation") =
"After"
        If Opt_Before.Value = True Then Data3.Recordset("Temporal_Relation") =
"Before"
        If Opt_During.Value = True Then Data3.Recordset("Temporal_Relation") =
"During"
        If Opt_Equal.Value = True Then Data3.Recordset("Temporal_Relation") =
"Equal"
        If Opt_Finish.Value = True Then Data3.Recordset("Temporal_Relation") =
"Finish"
        If Opt_Meet.Value = True Then Data3.Recordset("Temporal_Relation") =
"Meet"
        If Opt_Overlap.Value = True Then Data3.Recordset("Temporal_Relation") =
"Overlap"
        If Opt_Start.Value = True Then Data3.Recordset("Temporal_Relation") =
"Start"
        Data3.Recordset.Update
        Unload FormAddTemporal
        FormTemporal.Show
        Exit Sub
Error_Line:
        Error_Msg = "Time of " + Relation_1.Text + " should not come " + _
        Data3.Recordset("Temporal_Relation") + " time of " + Relation_2.Text
        MsgBox (Error_Msg)
        Unload FormAddTemporal
```

```
        FormTemporal.Show
        Exit Sub
NoUpdate:
        Error_ans = MsgBox(Error, 48, "Error!")
        Exit Sub
End Sub

 % ----------------------------------- Form Event ----------------------------------------- %
Private Sub ExitButton_Click()
        Unload FormEvent
        FormMenu.Visible = True
End Sub
Private Sub combo1_Click()
        Dim dbs As Database
        Dim rst As Recordset
        Set dbs = OpenDatabase(Filename)
        Set rst = dbs.OpenRecordset("TRelation")
        rst.MoveFirst
        Do While Not rst.EOF
                If StrComp(Combo1.Text, rst.Fields(11)) = 0 Then
                                Object1.Text = rst.Fields(0) & " : " & rst.Fields(1)
                                Object2.Text = rst.Fields(10)
                                Object3.Text = rst.Fields(2) & " : " & rst.Fields(3)
                End If
                rst.MoveNext
        Loop
        rst.Close
        dbs.Close
End Sub

Private Sub Form_Load()
```

```
        Dim dbs As Database
        Dim rst As Recordset
        Set dbs = OpenDatabase(Filename)
        Set rst = dbs.OpenRecordset("TRelation")
        rst.MoveFirst
        Do While Not rst.EOF
                Combo1.AddItem rst.Fields(11)
                rst.MoveNext
        Loop
        rst.Close
        dbs.Close
End Sub

% ------------------------------------ Form Relation ---------------------------------------- %
Private Sub DisableText()
        Concept_Type_1.Enabled = False
        Concept_Type_2.Enabled = False
        Indv_Name_1.Enabled = False
        Indv_Name_2.Enabled = False
        Relation.Enabled = False
        Event_Name.Enabled = False
        Begin_Date.Enabled = False
        Begin_Time.Enabled = False
        End_Date.Enabled = False
        End_Time.Enabled = False
        Duration_Date.Enabled = False
        Duration_Time.Enabled = False
End Sub

Private Sub AddButton_Click()
        Call EnableText
```

```
        Call HideButton
        FormRelation.Caption = "Add New Relation"
        Data1.Visible = False
        Data1.Refresh
        Data1.Recordset.AddNew
End Sub

Private Sub EnableText()
        Concept_Type_1.Enabled = True
        Concept_Type_2.Enabled = True
        Indv_Name_1.Enabled = True
        Indv_Name_2.Enabled = True
        Relation.Enabled = True
        Event_Name.Enabled = True
        Begin_Date.Enabled = True
        Begin_Time.Enabled = True
        End_Date.Enabled = True
        End_Time.Enabled = True
        Duration_Date.Enabled = True
        Duration_Time.Enabled = True
End Sub

Private Sub HideButton()
        AddButton.Visible = False
        EditButton.Visible = False
        DeleteButton.Visible = False
        ExitButton.Visible = False
        OKButton.Visible = True
        CancelButton.Visible = True
End Sub
```

```
Private Sub ShowButton()
        AddButton.Visible = True
        EditButton.Visible = True
        DeleteButton.Visible = True
        ExitButton.Visible = True
        OKButton.Visible = False
        CancelButton.Visible = False
End Sub

Private Sub Begin_Date_Change()
        If Len(Begin_Date) = 10 Then
                If (Val(Left(Begin_Date, 2)) > 31 Or Val(Left(Begin_Date, 2)) < 1 Or
                        Mid(Begin_Date, 4, 2) > 12 Or Mid(Begin_Date, 4, 2) < 1 Or
                        Val(Right(Begin_Date, 4)) > 9999 Or Val(Right(Begin_Date,
                        4)) < 0 Or Mid(Begin_Date, 3, 1) <> "/" Or Mid(Begin_Date,
                        6, 1) <> "/") Then
                        Error_ans = MsgBox("Invalid Begin Date", 48, "Error!")
                Else
                                OKButton.Enabled = True
                End If
        Else
                OKButton.Enabled = False
        End If
        If Begin_Date = "" Then OKButton.Enabled = True
End Sub
Private Sub Begin_Time_Change()
        If Len(Begin_Time) = 5 Then
                If (Val(Left(Begin_Time, 2)) > 24 Or Val(Left(Begin_Time, 2)) < 0
                Or Val(Right(Begin_Time, 2)) > 59 Or Val(Right(Begin_Time, 2)) <
                0 Or Mid(Begin_Time, 3, 1) <> ".") Then
                        Error_ans = MsgBox("Invalid Begin Time", 48, "Error!")
```

```
            Else
                        OKButton.Enabled = True
            End If
        Else
            OKButton.Enabled = False
        End If
        If Begin_Time = "" And Begin_Date = "" Then OKButton.Enabled = True
End Sub

Private Sub CancelButton_Click()
        Data1.Visible = True
        Call ShowButton
        Call DisableText
        FormRelation.Caption = "Relation : " + Filename
        If Data1.Recordset.RecordCount = 0 Then
            EditButton.Enabled = False
            DeleteButton.Enabled = False
        Else
            EditButton.Enabled = True
            DeleteButton.Enabled = True
        End If
End Sub

Private Sub DeleteButton_Click()
        FormRelation.Caption = "Delete Relation"
        If (MsgBox("Are you sure to delete this relation?", 36, "Delete relation?") =
6) Then
            Data1.Recordset.Delete
            Data1.Refresh
            Data1.Visible = True
        End If
```

```
        If Data1.Recordset.RecordCount = 0 Then
                EditButton.Enabled = False
                DeleteButton.Enabled = False
        Else
                EditButton.Enabled = True
                DeleteButton.Enabled = True
        End If
        FormRelation.Caption = "Relation : " + Filename
End Sub

Private Sub Duration_Date_Change()
        If Len(Duration_Date) = 10 Then
                If (Val(Left(Duration_Date, 2)) > 31 Or Val(Left(Duration_Date, 2))
                < 0 Or Mid(Duration_Date, 4, 2) > 12 Or Mid(Duration_Date, 4, 2) <
                0 Or Val(Right(Duration_Date, 4)) > 9999 Or Val(Right
                (Duration_Date, 4)) < 0 Or Mid(Duration_Date, 3, 1) <> "/" Or Mid
                (Duration_Date, 6, 1) <> "/") Then
                        Error_ans = MsgBox("Invalid Duration Date", 48, "Error!")
                Else
                                OKButton.Enabled = True
                End If
        Else
                OKButton.Enabled = False
        End If
        If Duration_Date = "" Then OKButton.Enabled = True
End Sub

Private Sub Duration_Time_Change()
        If Len(Duration_Time) = 5 Then
```

```
            If (Val(Left(Duration_Time, 2)) > 24 Or Val(Left(Duration_Time, 2))
            < 0 Or  Val(Right(Duration_Time, 2)) > 59 Or Val(Right
            (Duration_Time, 2)) < 0 Or Mid(Duration_Time, 3, 1) <> ".") Then
                    Error_ans = MsgBox("Invalid Duration Time", 48, "Error!")
            Else
                            OKButton.Enabled = True
            End If
    Else
            OKButton.Enabled = False
    End If
    If Duration_Time = "" And Duration_Date = "" Then OKButton.Enabled =
True
End Sub

Private Sub EditButton_Click()
    Call EnableText
    Call HideButton
    Event_Name.Enabled = False
    FormRelation.Caption = "Edit Relation"
    Data1.Visible = False
    Data1.Recordset.LockEdits = True
    Data1.Recordset.Edit
End Sub

Private Sub End_Date_Change()
    If Len(End_Date) = 10 Then
            If (Val(Left(End_Date, 2)) > 31 Or Val(Left(End_Date, 2)) < 1 Or _
            Mid(End_Date, 4, 2) > 12 Or Mid(End_Date, 4, 2) < 1 Or _
            Val(Right(End_Date, 4)) > 9999 Or Val(Right(End_Date, 4)) < 0 Or _
            Mid(End_Date, 3, 1) <> "/" Or Mid(End_Date, 6, 1) <> "/") Then
                            Error_ans = MsgBox("Invalid End Date", 48, "Error!")
```

```
        Else
            OKButton.Enabled = True
        End If
    Else
        OKButton.Enabled = False
    End If
    If End_Date = "" Then OKButton.Enabled = True
End Sub

Private Sub End_Time_Change()
    If Len(End_Time) = 5 Then
        If (Val(Left(End_Time, 2)) > 24 Or Val(Left(End_Time, 2)) < 0 Or _
        Val(Right(End_Time, 2)) > 59 Or Val(Right(End_Time, 2)) < 0 Or _
            Mid(End_Time, 3, 1) <> ".") Then
            Error_ans = MsgBox("Invalid End Time", 48, "Error!")
        Else
            OKButton.Enabled = True
        End If
    Else
        OKButton.Enabled = False
    End If
    If End_Time = "" And Begin_Time = "" Then OKButton.Enabled = True
End Sub

Private Sub ExitButton_Click()
    Data1.Refresh
    If Data1.Recordset.RecordCount <= 0 Then
        FormMenu!mnu_Edit_item(1).Enabled = False
        FormMenu!mnu_Query.Enabled = False
        FormMenu!mnu_App_item(2).Enabled = False
    Else
```

```
            FormMenu!mnu_Edit_item(0).Enabled = True
            FormMenu!mnu_Edit_item(1).Enabled = True
            FormMenu!mnu_Query.Enabled = True
            FormMenu!mnu_App_item(2).Enabled = True
        End If
        Unload FormRelation
        FormMenu.Visible = True
End Sub

Private Sub Form_Load()
        Dim Error_ans As Integer
        AddMode = False
        Call ShowButton
        Call DisableText
        Data1.DatabaseName = Filename
        FormRelation.Caption = "Relation : " + Filename
        Data1.Visible = True
        Data1.Refresh
        If Data1.Recordset.RecordCount <= 0 Then AddButton_Click
End Sub

Private Sub OKButton_Click()
        Dim BT_YR, BT_MO, BT_DA, BT_HH, BT_MM
        Dim ET_YR, ET_MO, ET_DA, ET_HH, ET_MM
        On Error GoTo NoUpdate
        If (Begin_Date.Text = "") Or (End_Date.Text = "") Then GoTo Update_Line
        If (Begin_Date.Text <> "") And (End_Date.Text <> "") Then
            BT_YR = Val(Right(Begin_Date.Text, 4))
            BT_MO = Val(Mid(Begin_Date.Text, 4, 2))
            BT_DA = Val(Left(Begin_Date.Text, 2))
            BT_HH = Val(Left(Begin_Time.Text, 2))
```

```
            BT_MM = Val(Mid(Begin_Time.Text, 4, 2))
            ET_YR = Val(Right(End_Date.Text, 4))
            ET_MO = Val(Mid(End_Date.Text, 4, 2))
            ET_DA = Val(Left(End_Date.Text, 2))
            ET_HH = Val(Left(End_Time.Text, 2))
            ET_MM = Val(Mid(End_Time.Text, 4, 2))
            If BT_YR > ET_YR Then GoTo Error_Line
            If BT_YR < ET_YR Then GoTo Update_Line
            If BT_MO > ET_MO Then GoTo Error_Line
            If BT_MO < ET_MO Then GoTo Update_Line
            If BT_DA > ET_DA Then GoTo Error_Line
            If BT_DA < ET_DA Then GoTo Update_Line
            If BT_HH > ET_HH Then GoTo Error_Line
            If BT_HH < ET_HH Then GoTo Update_Line
            If BT_MM > ET_MM Then GoTo Error_Line
            If BT_MM < ET_MM Then GoTo Update_Line
        End If
        GoTo Update_Line
Error_Line:
        Error_Msg = "Begin time should before end time"
        MsgBox (Error_Msg)
        Data1.Recordset.CancelUpdate
        GoTo Continue
Update_Line:
        Data1.Recordset.Update
        GoTo Continue
Continue:
        Data1.Visible = True
        Data1.Refresh
        Call DisableText
        Call ShowButton
```

```
    FormRelation.Caption = "Relation : " + Filename
    If Data1.Recordset.RecordCount = 0 Then
        EditButton.Enabled = False
        DeleteButton.Enabled = False
    Else
        EditButton.Enabled = True
        DeleteButton.Enabled = True
    End If
    Exit Sub
NoUpdate:
    Error_ans = MsgBox(Error, 48, "Error!")
    Exit Sub
End Sub

Private Sub Relation_Change()
    If Relation = "" Then
        OKButton.Enabled = False
    Else
        OKButton.Enabled = True
    End If
End Sub

% ---------------------------------------- Form Temporal ---------------------------------------- %
Private Error_ans As Integer
Private MaxNum As Integer
Private Sub AddButton_Click()
    Modify_Rec = -1
    Unload FormTemporal
    FormAddTemporal.Show 1
End Sub
```

```
Private Sub Data3_Reposition()
        On Error GoTo No_Update
        If Data3.Recordset.RecordCount = 0 Then Exit Sub
        Data1.RecordSource = "SELECT * FROM TRelation WHERE
        Event_Name='" + Data3.Recordset("Relation_1") + "'"
        Data1.Refresh
        Data2.RecordSource = "SELECT * FROM TRelation WHERE
        Event_Name='" + Data3.Recordset("Relation_2") + "'"
        Data2.Refresh
        Frame_Event_1.Caption = "Event : " + Event_1.Text
        Frame_Event_2.Caption = "Event : " + Event_2.Text
        Modify_Rec = Data3.Recordset("Temporal_Id")
        Select Case Data3.Recordset("Temporal_Relation")
                Case "After"    Opt_After.Value = True
                Case "Before"  Opt_Before.Value = True
                Case "During"  Opt_During.Value = True
                Case "Equal"   Opt_Equal.Value = True
                Case "Finish"   Opt_Finish.Value = True
                Case "Meet"    Opt_Meet.Value = True
                Case "Overlap"  Opt_Overlap.Value = True
                Case "Start"   Opt_Start.Value = True
        End Select
        Exit Sub
No_Update:
        Error_ans = MsgBox(Error, 48, "Error!")
        Exit Sub
End Sub

Private Sub DeleteButton_Click()
        FormTemporal.Caption = "Delete Temporal Relation"
```

```
        If (MsgBox("Are you sure to delete this temporal relation?", 36, "Delete
        relation?") = 6) Then Data3.Recordset.Delete
        Data3.Refresh
        If Data3.Recordset.RecordCount = 0 Then
                EditButton.Enabled = False
                DeleteButton.Enabled = False
                Data3.Visible = False
        Else
                EditButton.Enabled = True
                DeleteButton.Enabled = True
        End If
        FormTemporal.Caption = "Temporal Relation : " + Filename
End Sub

Private Sub EditButton_Click()
        Modify_Rec = 1
        Unload FormTemporal
        FormAddTemporal.Show
End Sub

Private Sub ExitButton_Click()
        Unload FormTemporal
        Unload FormAddTemporal
        FormMenu.Visible = True
End Sub

Private Sub Form_Load()
        Data3.DatabaseName = Filename
        FormTemporal.Caption = "Temporal Relation : " + Filename
        Data3.Refresh
        If Data3.Recordset.RecordCount = 0 Then
```

```
            DeleteButton.Enabled = False
            EditButton.Enabled = False
            Data3.Visible = False
        Else
            Data1.DatabaseName = Filename
            Data2.DatabaseName = Filename
            DeleteButton.Enabled = True
            EditButton.Enabled = True
            Data3.Visible = True
        End If
End Sub
% ---------------------------------- Form Topological ---------------------------------- %
Dim Check_Click, MaxCol, MaxList, MaxRow, MaxLabel, MaxLine
Dim Relation(50), LineX(50), LineY(50), pX(50), pY(50)
Private Sub Form_Load()
        Dim FileNumber, Tmp_Relation(50)
        Dim I, J, K
        Check_Click = False
        FileNumber = FreeFile
        I = 0,   K = 0,   MaxCol = 0
        Open "C:\CG\ORDER.OUT" For Input As #FileNumber
        Do While Not EOF(FileNumber)
            Input #FileNumber, Tmp_Relation(I)
            I = I + 1
        Loop
        Close #FileNumber
        I = I - 1
        Tmp_Relation(0) = Mid(Tmp_Relation(0), 2)
        Tmp_Relation(I) = Left(Tmp_Relation(I), Len(Tmp_Relation(I)) - 1)
        For J = 0 To I
            Relation(K) = Tmp_Relation(J)
```

```
        K = K + 1
        If Left(Tmp_Relation(J), 1) = "[" Then Relation(K - 1) = Mid
        (Tmp_Relation(J), 2)
        If Right(Tmp_Relation(J), 1) = "]" Then
        Relation(K - 1) = Left(Tmp_Relation(J), Len(Tmp_Relation(J)) - 1)
                Relation(K) = "NextOrder"
                K = K + 1
        End If
Next J
MaxList = K - 1
BackColor = QBColor(7)
ScaleMode = 3   ' Set ScaleMode to pixels.
Left = 0
Width = Screen.Width
Height = 5000
Top = (Screen.Height - Height) / 2
ScaleMode = 3
VScroll1.Top = 0
VScroll1.Height = ScaleHeight - 20
VScroll1.Left = ScaleWidth - 17
HScroll1.Left = 0
HScroll1.Top = ScaleHeight - 17
HScroll1.Width = ScaleWidth - 17
K = 0, J = 0,  X = 0,  Y = 0
For I = 0 To MaxList
        If Relation(I) <> "NextOrder" Then
                If K > 0 Then
                Load Label1(K)
                Load Shape1(K)
                End If
            If Len(Relation(I)) > 15 Then
```

```
                    Label1(K).Alignment = 0
                    Else
                    Label1(K).Alignment = 2
                    End If
                    Label1(K).Visible = True
                    Label1(K).Caption = Relation(I)
                    pX(K) = 15 + (100 * X)
                    pY(K) = 140 + Y
                    Label1(K).Move pX(K), pY(K)
                    Shape1(K).Visible = True
                    Shape1(K).Move pX(K) - 5, pY(K) - 10
                    If J > 0 And Relation(I + 1) <> "NextOrder" Then
                Load Line15(J)
                If Relation(I + 1) <> "NextOrder" Then
                    Line15(J).Visible = True
                    LineX(J) = pX(K) + 75
                    LineY(J) = pY(K) + 10
                    Line15(J).X1 = LineX(J)
                    Line15(J).Y1 = LineY(J)
                    Line15(J).X2 = LineX(J) + 20
                    Line15(J).Y2 = LineY(J)
                    J = J + 1
                    End If
                    K = K + 1
                    X = X + 1
            Else
                    If MaxCol < X Then MaxCol = X
                    X = 0
                    Y = Y + 50
            End If
        Next I
```

```
        MaxLabel = K - 1
        MaxLine = J - 1
        MaxRow = (Y - 130) / 50
        FormTopological.Visible = True
        VScroll1.Max = MaxRow * 3
        VScroll1.Min = -MaxRow * 3
        VScroll1.LargeChange = 10
        VScroll1.SmallChange = 1
        HScroll1.Max = 80 + MaxCol
        HScroll1.Min = -15 * MaxCol
        HScroll1.LargeChange = MaxCol * 5
        HScroll1.SmallChange = MaxCol - 1
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y
As Single)
        Frame1.Visible = False
        If Check_Click = True Then
                Dim I
                FormTopological.Cls
                For I = 0 To MaxLabel
                        Label1(I).Move pX(I), pY(I)
                        Shape1(I).Move pX(I) - 5, pY(I) - 10
                Next I
                For I = 0 To MaxLine
                        Line15(I).X1 = LineX(I)
                        Line15(I).Y1 = LineY(I)
                        Line15(I).X2 = LineX(I) + 20
                        Line15(I).Y2 = LineY(I)
                Next I
        End If
```

```
        Check_Click = False
End Sub

Private Sub HScroll1_Change()
        Dim I
        FormTopological.Cls
        For I = 0 To MaxLabel
                pX(I) = pX(I) + (HScroll1.Value / 100 * ScaleWidth)
                Label1(I).Move pX(I), pY(I)
                Shape1(I).Move pX(I) - 5, pY(I) - 10
        Next I
        For I = 0 To MaxLine
                LineX(I) = LineX(I) + (HScroll1.Value / 100 * ScaleWidth)
                Line15(I).X1 = LineX(I)
                Line15(I).Y1 = LineY(I)
                Line15(I).X2 = LineX(I) + 20
                Line15(I).Y2 = LineY(I)
        Next I
End Sub

Private Sub Label1_Click(Index As Integer)
        Dim dbs As Database
        Dim rst As Recordset
        Check_Click = True
        Set dbs = OpenDatabase(Filename)
        Set rst = dbs.OpenRecordset("TRelation")
        rst.MoveFirst
        Do While Not rst.EOF
                If StrComp(Label1(Index).Caption, rst.Fields(11)) = 0 Then
                                Text1.Text = rst.Fields(0) & " : " & rst.Fields(1)
                                Text2.Text = rst.Fields(10)
```

```
                Text3.Text = rst.Fields(2) & " : " & rst.Fields(3)
        End If
        rst.MoveNext
    Loop
    rst.Close
    dbs.Close
    Frame1.Visible = True
End Sub

Private Sub VScroll1_Change()
    Dim I
    FormTopological.Cls
    For I = 0 To MaxLabel
        pY(I) = pY(I) + (VScroll1.Value / 100 * ScaleWidth)
        Label1(I).Move pX(I), pY(I)
        Shape1(I).Move pX(I) - 5, pY(I) - 10
    Next I
    For I = 0 To MaxLine
        LineY(I) = LineY(I) + (VScroll1.Value / 100 * ScaleWidth)
        Line15(I).X1 = LineX(I)
        Line15(I).Y1 = LineY(I)
        Line15(I).X2 = LineX(I) + 20
        Line15(I).Y2 = LineY(I)
    Next I
        End Sub
```