A Study on Natural Language Understanding Based on Mental Image Directed Semantic Theory

by

BD14002
Rojanee Khummongkol

Engineering Doctor's Program

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Engineering

Intelligent Information System Engineering
Fukuoka Institute of Technology
Fukuoka, Japan

Acknowledgement

In order to accomplish my dissertation, first of all I would like to express my deep appreciation to Prof. Masao Yokota, Ph.D., my great supervisor for his guidance, enthusiastic encouragement and useful critiques of my research work during my study of doctor course in Japan and henceforth. He does not advise me only academic knowledge, but also remind me the way of life. However, I would not have succeeded in this attempt if I had lacked of opportunity from him to enter Fukuoka Institute of Technology (FIT) as a Ph.D. student with the support of education fund from my original affiliation, University of Phayao.

Moreover, I would like to extend my gratitude to Prof. Dr. Yu Song, Prof. Dr. Hiroyuki Fujioka, Prof. Dr. Hitoshi Kino, and others for their suggestion, advices and assistances. Also furthermore, I wish my grateful thanks to Mr. Hiroshi Chikamatsu, Ms. Sawako Tsunenoki, and Ms. Hiroko Yoshida, FIT staffs, who always help and embolden me to go over troubles. Besides, I am particularly sincere to thank my special guests, i.e. Janelle Chang, Christine C. Cabugao, Mingyue Qiu, Taiyo Nishiguchi, Jiraporn Pooksook, and my father (Chairoj Khummongkol) for their helps as a sample group in my psychological experiment of this research, and another one whom I cannot miss is Marie Ibarra and Steven Garratt, my American friends whom I usually consult on English usage.

In addition, I wish my proper thanks to be given to Mr. and Mrs. Toyama, dormitory master and matron, for their kindness and warm tendance all over three years for my staying in Japan, together with, Mr. Yoshitsuku Saiga, my Japanese friend who guides me many places in Fukuoka and neighborhood all the time.

Finally, from the bottom of my heart, I wish to profoundly thank to my family, fiancé, and friends for their support, unction, encouragement, and so on throughout my life and study. Without moral support from them, I would not able to succeed in my life as today.

Rojanee Khummongkol
Intelligent Information System Engineering
BD14002

Abstract

Rapid development of technology has brought usual scenes where robots and computers are helping various human activities, not only in industrial sector but also in other widespread areas, such as household, agriculture, medicine, nursery, and transportation. However, it is rather difficult for ordinary people to communicate with them in special technical languages. Reflecting our casual life, natural language (NL) is the most conventional communication means among us, and therefore it is much easier for us to communicate with machines in NL. This thesis proposes a methodology for natural language understanding (NLU) named Mental-image Based Understanding (MBU), intending that a robot can understand NL as people do. Its application system can reach the most plausible semantic interpretation of an input text and return desirable outcomes based on word concepts, postulates, and inference rules formalized in Mental-image Description Language ($\boldsymbol{L_{md}}$) developed and proposed by M.Yokota in his original theory "Mental Image Directed Semantic Theory (MIDST)". According to our experiments, it has been proved that MBU is simple to use and utilize for semantic interpretation and will be able to influence robots with capability of NLU as well as humans based on a mental image model.

Keywords: Natural Language Understanding, Mental Image Model, Mental-Image Based Understanding, Semantic Interpretation

概要

急速な技術の発達により、ロボットやコンピュータが人間の多様な活動を支援する場面が普通に見られるようになった。それは、工業のみならず他の広い分野、例えば、家事、農業、医療、看(介)護や交通運輸などの分野があげられる。しかしながら、普通の人々が特別な技術言語でそれらの機械とコミュニケーションを行うのはかなり困難である。人々の日常においては自然言語が最も普及したコミュニケーション手段であり、したがって、人々にとって機械とでも自然言語を用いて意思疎通を図れればより便利である。本学位論文は、心像に基づく理解方式 (Mental-image Based Understanding: MBU) と呼ぶ自然言語理解 (Natural Language Understanding: NLU)方法を提案する。これは、ロボットに人間と同様の自然言語理解を行わせることを目的としている。それを応用したシステムは心像記述言語 (Mental-image Description Language: ***L_{md}***)を用いて形式化された単語概念、公準および推論規則に基づき妥当な意味解釈を推定し期待された結果を出力することができる。***L_{md}*** は横田の独自の理論、心像意味論(Mental Image Directed Semantic Theory: MIDST) において開発・提案されたものであり、この応用システムに関する実験の結果、提案した MBU 方式は意味解釈方法として簡便かつ有効であり、将来、心像モデルに基づきロボットに人間同様の自然言語理解能力を与えうるとの確証がえられた。

キーワード：自然言語理解、心像モデル、心像に基づく理解、意味解釈

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Nowadays, most of our activities always relate to computer equipment. For example, many people use mobile phones for alarm when they wake up and for news reading. Therefore, we cannot refuse that a cell phone is a very basic and useful device that facilitates us not only to contact to each other, but also contains many functions that help us to achieve our desired goals more easily. On the contrary, rapid development of technologies, communications, transportations, and economics has resulted in unnatural and poor social interaction for people. This fact leads to decline of nuptiality and birth rate destined to aging societies. Therefore, many governments are setting policies against these problems. As one of the solutions, robots have come to be widely used items that help people in industrial factories, household sectors and so on. However, these automata will not be able to ease people perfectly, if they lack of smart software called Artificial Intelligence System (AIS), embedded in their organizations.

If we talk about communication methods among people, we may think of many things, for instance, gesture, art, music, etc. Among all, natural language is the most common and convenient method to express our ideas, feeling, thinking, and so forth. In addition, this fact holds as well between man and robot, so called Human-Robot Interaction (HRI).

Reflecting the casual scenes of people and robots, spatiotemporal (4D) language, i.e. language to express topological, directional, metric, and time relations, is the most important to be concerned in order to share knowledge amongst them. 4D language has received special attention in the field of ontology because it includes fundamental issues concerning human cognition such as vagueness and ambiguity. However, traditional approaches have been focused on computing purely objective three-dimension (3D) geometric relations as visual scene [1.1] while people thinking process is often intuitive as shown in Fig. 1.1.

Why can human understand such sentences as S1 – S3 easily?

(S1) Tom was on the hill flying a flag back and forth.

(S2) Tom was with the book in the bus running from town to university.

(S3) I saw the planet in my room.

Along with Fig 1.1 and S1 – S3 expressions, English speakers can experience their understanding about spatial relations as pictures drawn in mind. Therefore, we can recognize objects by mental image. Moreover, we also can cognize and conceive the external world as spatial gestalts shown in Fig. 1.2. That is, human brain is a super marvelous organ!

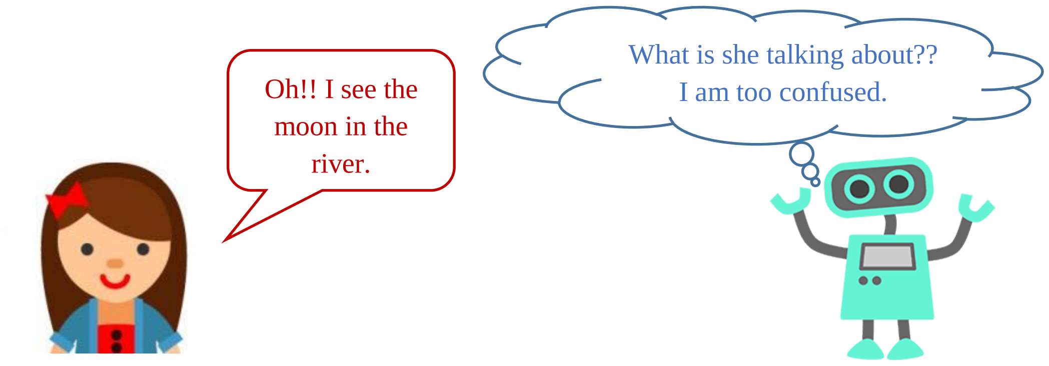

Oh!! I see the moon in the river.
What is she talking about?? I am too confused.

Fig. 1.1 Intuitive thinking process in human.

(S4) Five dots are in *line*.

(S5) Nine dots are placed in *the shape of X*.

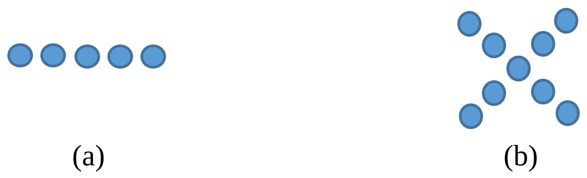


Fig. 1.2 Spatial gestalts.

Fig. 1.2 (a) and (b) show that human perceives the continuous forms among the dots located separately, and would express them like S4 and S5 respectively.

T. Winograd [1.2] stated that the relationships between Human-Computer Interaction (HCI) and AI for human-like intellect should be described in T-Shaped where horizontal and vertical line were represented to HCI and AI discipline respectively. The famous computer program ELIZA [1.3] is positioned in a very shallow level of this model, that is, doesn't really understand NL but is designed well to almost pass the Turing test. Then, to test machine intelligence and improve the Turing Test [1.4], Winograd Schema Challenge (WSC) was proposed by H. J. Levesque [1.5] in 2011, and so on [1.6] – [1.7]. Anyway, Natural Language Understanding (NLU) is still one of attractive fields among researchers these days, such as "Watson", the Q&A cognitive system developed by IBM [1.8] – [1.9]. However, Watson could not understand Natural Language in real sense because it applied Natural Language

Processing (NLP) techniques to Big Data Analysis. Against this approach, B. M. Lake et al. [1.10] presented the outline in order to build a machine to imitate human learning and thinking processes. Although many works have focused on NLP, it is still extremely difficult to make a machine reach real NLU like people.

This thesis proposes a methodology for natural language understanding (NLU) named Mental-image Based Understanding (MBU), intending that a robot can understand NL as people do. Its application system can reach the most plausible semantic interpretation of an input text and return desirable outcomes based on word concepts, postulates, and inference rules formalized in Mental-image Description Language ($\boldsymbol{L_{md}}$) developed and proposed by M.Yokota in his original theory "Mental Image Directed Semantic Theory (MIDST)" [1.11] - [1.12]. The methodology MBU is implemented in two types of NLU systems named Mental-image Based Understanding System and Conversation Management System. The former is aimed for question-answering with users and the latter performs dialogue management between users and an imaginary robot named Anna. According to our experiments, it has been proved that MBU is simple to use and utilize for semantic interpretation and will be able to influence robots with capability of NLU as well as humans based on a mental image model.

This thesis document consists of 6 chapters that can be summarized as follows. Chapter 1 introduces the background knowledge and the purpose of this work. Chapter 2 describes MIDST to represent human perception process in a computable logical form. Chapter 3 analyzes natural event concepts in association with human attention mechanism from the viewpoint of logical computation. Chapter 4 proposes a problem finding and solving method based on MIDST. Chapter 5 presents Mental-image Based Understanding System and Conversation Management System developed in this work. Finally, chapter 6 discusses and concludes this thesis.

CHAPTER 2

PRELIMINARIES

In this chapter, we will introduce Mental Image Directed Semantic Theory (MIDST), a model that we employ to represent human perception process (as shown in Fig. 2.1) in computable logical from, named Language for Mental-Image Description (***L_{md}***) that suitable for "Temporal change events" and "Spatial change events" using attributes of physical objects, and so on.

2.1 Formal System

Formal system is a system that consists of formal language and deductive method where inference rules and postulates are used to derive the theorem. As above mentioned, ***L_{md}*** is a formal language that employs many-sorted predicate logic with five-kind individual terms specific to the mental image model involved in predicate constant (L), named "Atomic locus" , where the deductive apparatus here is intended to base on deductive system of predicate calculus.

The symbols of the deductive apparatus for ***L_{md}*** are presented from (i) – (xi) as follows [2.1].

(i) Logical operators: a symbol used to join sentences, i.e. $\sim$, $\wedge$, $\vee$, $\supset$, $\equiv$

(ii) Quantifiers: a symbol used to specify the quantity of instances, i.e. $\forall$, $\exists$

(iii) Meta-symbols: a symbol used to indicate one concept to another concept, i.e. $\Leftrightarrow$, $\rightarrow$, $\leftrightarrow$, etc.

(iv) Auxiliary constants: a symbol used to qualify or insert into a sentence or passage, i.e. ., (,)

(v) Sentence constant: a symbol used to indicate sentence, i.e. N.

(vi) Predicate constants: a symbol used to indicate predicate, i.e. L, or specify meaning in universal set, e.g. $=$, $\neq$, $>$, $<$, etc.

(vii) Individual constants: consisting of five types of them:

a. Matter constant[*]

b. Attribute constants: A, B

c. Value constant[*]

d. Pattern constant: G

e. Standard constant: K

*Please remark that matter and value constant will be introduced when need.

(viii) Function constants: arithmetic operations, e.g. +, - , *, /, etc.

(ix) Sentence variables: a symbol used to represent sub-sentence in sentence constant, i.e. χ.

(x) Individual variables: the variables used to indicate variables in individual constants that contain of five kinds of terms:

a. Matter variable: x, y, z

b. Attribute variable: a

c. Value variable: p, q, r, s, t

d. Pattern variables: g

e. Standard variable: k

(xi) Others: a symbol that will be defined by above symbols.

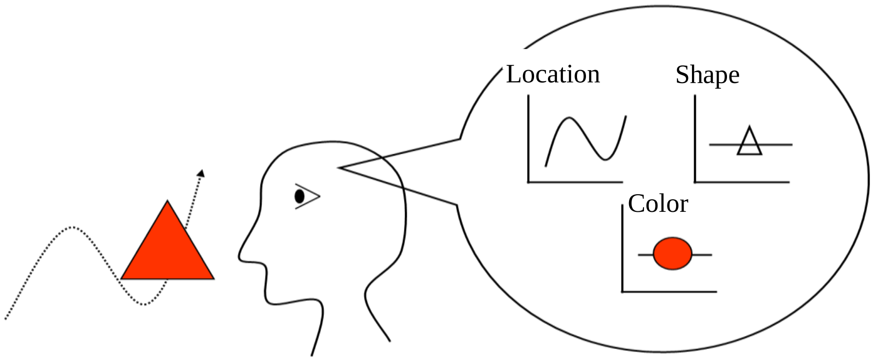


Fig. 2.1 Loci of the observer's attention in attribute spaces.

Atomic locus or Atomic locus formula is gathered by seven-place predicate that given by eq. 2.1.

$$L(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, \omega_7) \qquad \text{(eq. 2.1)}$$

Where ω_1 is a matter as event causer

ω_2 is a matter as attribute carrier

ω_3 is a value or matter

ω_4 is a value or matter

ω_5 is an attribute of ω_2

ω_6 is a pattern (Temporal or spatial change event)

ω_7 is a standard or matter

In eq. 2.1, ω_3, ω_4, and ω_7 should represent their values over time-interval. Moreover, if $\omega_{1,}\omega_3, \omega_4$, or ω_7 is unknown variable, the symbol "_" can be replaced instead of that variable, while, ω_5, the attribute term, can be seen its detail in Appendix A.

However, to simplify eq. 2.1, we can employ variables or constants instead of terms as the following:

$$L(x, y, p, q, a, g, k) \qquad \text{(eq. 2.2)}$$

where eq. 2.2 can interpret using Fig 2.2 means that:

"*Matter 'x' causes Attribute 'a' of Matter 'y' to keep or change its value temporally or spatially over a time-interval, where the value 'p' and 'q' are relative to Standard 'k'*"

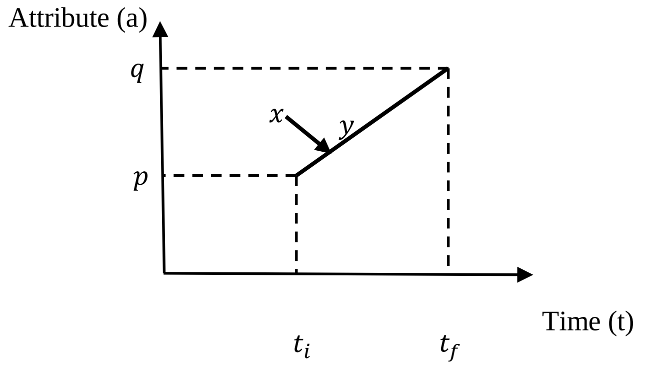


Fig. 2.2 Atomic locus.

As above mentioned, ***L**_{md}* is designed to support for natural event concepts of an attribute in time or space domain (g *in* $eq.$ 2.2). Therefore, if the event tends to monotonous change/stability over time domain, g will be replaced by Temporal change event (G_t). As well as space domain, g will be substituted by G_s.

Consider S6 – S7 expressions.

(S6) The train runs from Fukuoka to Kumamoto.

(S7) The track runs from Fukuoka to Kumamoto.

S6 and S7 are looked similarly, but when reading these two sentences, we can notice that our eyes (or the Focus of the Attention of the Observer (FAO)) will keep on different objects (or Attribute Carriers (AC)) as show in Fig. 2.3.

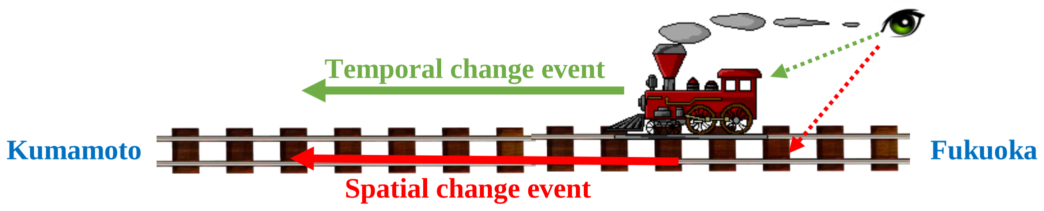


Fig. 2.3 Temporal/spatial change event.

The green and red arrows in Fig. 2.3 show the differences over time domain. The green one represents the moving train in S6, while the red arrow stands for extended railway from Fukuoka to Kumamoto in S7. So, G_t and G_s will supplant g in eq. 2.2 respectively. Then, logical form of S6 and S7 can be written as eq. 2.3 and 2.4 severally, when A_{12} is physical location.

$$\exists(x,y,k)L(x,y,Fukuoka,Kumamoto,A_{12},G_t,k)\wedge train(y) \qquad \text{(eq. 2.3)}$$

$$\exists(x,y,k)L(x,y,Fukuoka,Kumamoto,A_{12},G_s,k)\wedge track(y) \qquad \text{(eq. 2.4)}$$

Normally, when an NL expression is interpreted in $\boldsymbol{L_{md}}$, it always forms of many Atomic Loci. Hence, “Tempo-Logical Connectives (TLCs)”, binary connectives, will be used in deductive system for representing temporal and logical relationships between loci, that is to say “∧” for conjunction, “∨” for disconjunction, “⊃” for implication, and “≡” for equivalence. Anyway, the most frequently used TLCs in MIDST are “Simultaneous AND” and “Consecutive AND” that can be denoted by “SAND” or “Π” for Simultaneous AND, while “CAND” or “●” are used in place of Consecutive AND.

$$\chi_1 C_i \chi_2 \Leftrightarrow (\chi_1 C \chi_2)\wedge\tau_i(\chi_1,\chi_2) \qquad \text{(eq. 2.5)}$$

Where, $\tau_{-i}(\chi_2,\chi_1) \equiv \tau_i(\chi_1,\chi_2)(\forall i \in \{0,\pm1,\pm2,\ldots,\pm6\})$

Eq. 2.5 is the description of tempo-logical connective C_i, where $\chi, C,$ and τ refer to a locus, a binary logical connective, and a temporal relation respectively, while i is a indexed number. Here, the temporal relation (τ) can be divided into thirteen groups by indexed number ranging from -6 to 6 [2.2] (for further details, see APPENDIX A).

Fig. 2.4 illustrates the mental image evoked by the verb *fetch*, a temporal change event concept, which can be formalized as the locus formula (eq. 2.6) that was detailed in Yokota [2.3]

$$\exists(x,y,p_1,p_2,k)L(x,x,p_1,p_2,A_{12},G_t,k)\bullet$$
$$L(x,x,p_2,p_1,A_{12},G_t,k)\Pi L(x,y,p_2,p_1,A_{12},G_t,k)\wedge x \neq y\Lambda p_1 \neq p_2 \qquad \text{(eq. 2.6)}$$

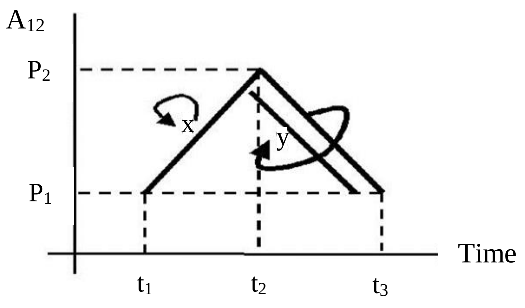


Fig. 2.4 Mental image evoked by the verb "fetch".

From all of above, for simplicity in reading and understanding, quantifiers: ∀ and ∃ will be neglected, and $a, g,$ and k terms in eq. 2.2 will group them as α. (See eq. 2.7, where $\alpha = (a, g, k)$)

$$L(x,y,p,q,\alpha) \qquad \text{(eq. 2.7)}$$

Then, eq. 2.3 and 2.4 can be written in short as follows:

$$L(\_, train, Fukuoka, Kumamoto, \alpha_t) \qquad \text{(eq. 2.8)}$$

$$L(\_, track, Fukuoka, Kumamoto, \alpha_s) \qquad \text{(eq. 2.9)}$$

According to the Event Causers in eq. 2.8 and 2.9 are anonymous, so, "_" is employed to place in the first term of these two expressions.

While Fig. 2.5 shows some more examples of event patterns in attribute spaces.

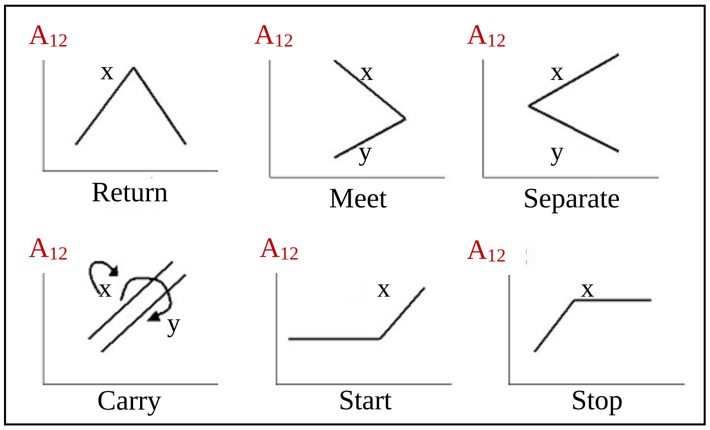


Fig. 2.5 Examples of event patterns of physical location.

2.2 Mental Image Processing

At the topic 2.1, we have described about formal system, the logical expression that is the basic knowledge uses to implement this work. However, in order to make a machine can be aware of NL like people, a methodology that simulate human mental image, Mental-Image Based Understanding (MBU), must be applied. How can this work interpret mental image to $\boldsymbol{L_{md}}$?

Please consider S8:

(S8) The bus runs from Town to University.

Normally, when people read S8 sentence, if we can bring the scene out of our brains, we always think of a moving picture of the running bus that depicted in Fig. 2.6. (Please note that, the words "Town" and "University" in S8 are initialed by capital letters due to bus stop names.)

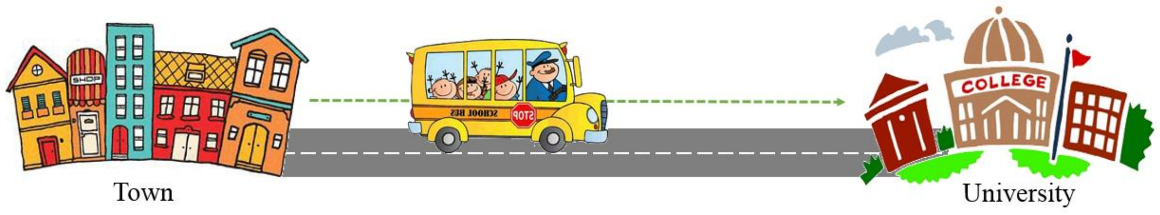


Fig. 2.6 Mental image depiction of S8.

Anyway, because of immoderate information, so in order to interpret NL expression or image into computable logic, therefore, abstract picture is employed to explain the imagery.

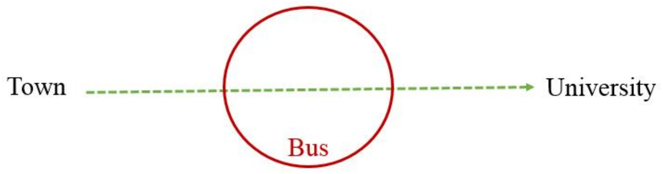


Fig. 2.7 Highly abstract picture of S8.

The circle in Fig. 2.7 refers to the object, bus, in S8 expression, while broken arrow is used to represent the movement of the object. Anyhow, please remark that color and shape in the figure are not significant at all.

Now please consider S9.

(S9) Tom was with the book in the bus running from Town to University.

If applied highly abstract picture to the sentence, what will happen?

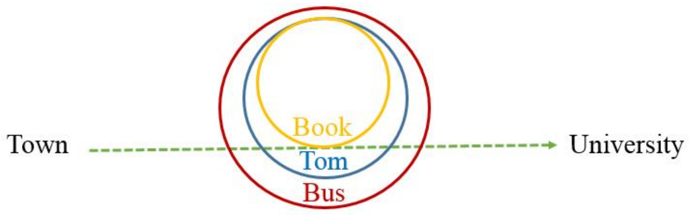


Fig. 2.8 Highly abstract picture of S9.

Fig. 2.8 shows the relationships between "Tom", "Book", and "Bus" in S9. As above, color and shape are not significant, but if notice, we can see that Tom circle and book circle are touched, what does it mean? The contacting circles mean that "Tom was holding the book by himself", because the book circle is enclosed by Tom circle. As well as Tom and book circle are also surrounded by bus circle, it implies that "Tom and book were in the bus, while the bus was running from Town to University".

So, if we ask the S10 to S9, it will not suspect why MBU can return the correct answer as shown in Fig. 2.9.

(S10) Did Tom carry the book from Town to University?

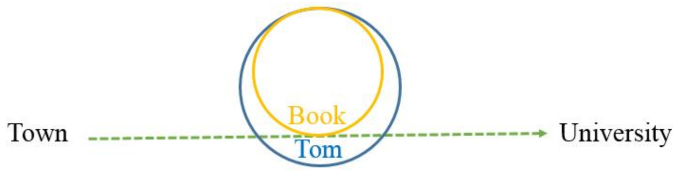


Fig. 2.9 Highly abstract picture of S10.

CHAPTER 3

FORMULATION OF SUBJECTIVE SPATIOTEMPORAL KNOWLEDGE AND NATURAL LANGUAGE UNDERSTANDING

Chapter 2 introduced MIDST about its main methodology for mental image formalization. In this chapter, the difference between Spatial and Temporal Change Event is explained based on the relations between Attribute Carrier (AC) and the Focus of the Attention of the Observer (FAO).

3.1 Attribute Carrier and the Focus of the Attention of the Observer

MIDST hypothesizes that the difference between temporal and spatial change event concepts can be attributed to the relationship between the Attribute Carrier (AC) and the Focus of the Attention of the Observer (FAO). To be brief, it is hypothesized that FAO is fixed on the whole AC in a temporal change event but runs about on the AC in a spatial change event. Consequently, the train and FAO move together in the case of S6 while FAO solely moves along the track in the case of S7. That is, all loci in attribute spaces are assumed to correspond one to one with movements or, more generally, temporal change events of FAO.

Therefore, an event expressed in $\boldsymbol{L_{md}}$ is compared to a movie film recorded through a floating camera because it is necessarily grounded in FAO's movement over the event. And this is why S11 and S12 can refer to the same scene in spite of their appearances, where what "sinks" or "rises" is the FAO as illustrated in Fig. 3.1 and whose conceptual descriptions are given as eq. 3.1 and 3.2, respectively.

(S11) The path *sinks* to the stream.

(S12) The path *rises* from the stream.

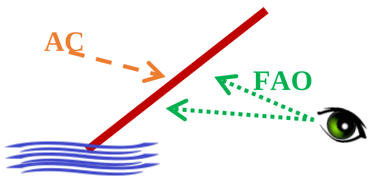


Fig. 3.1 Mental image depiction of S11 and S12.

$$L(\_, y, p, z, A_{12}, G_s, \_)\Pi L(\_, y, \downarrow, \downarrow, A_{13}, G_s, \_)\Lambda path(y)\Lambda stream(z)\Lambda p \neq z \qquad \text{(eq. 3.1)}$$

$$L(\_, y, p, z, A_{12}, G_s, \_)\Pi L(\_, y, \uparrow, \uparrow, A_{13}, G_s, \_)\Lambda path(y)\Lambda stream(z)\Lambda p \neq z \qquad \text{(eq. 3.2)}$$

Where “A_{13}”, “↑” and “↓” in eq. 3.1 and 3.2 refer to the attribute “Direction” and its values “upward” and “downward”, respectively.

Such a fact is generalized as “Postulate of Reversibility of Spatial Change Event (PRS)”. This postulate is also valid for such a pair of S13 and S14 (depicted as Fig. 3.2) as interpreted approximately into eq. 3.3 and 3.4, respectively. These pairs of conceptual descriptions are called equivalent in the PRS, and the paired sentences are treated as paraphrases each other.

(S13) Route A and Route B *meet* at the city.
(S14) Route A and Route B *separate* at the city.

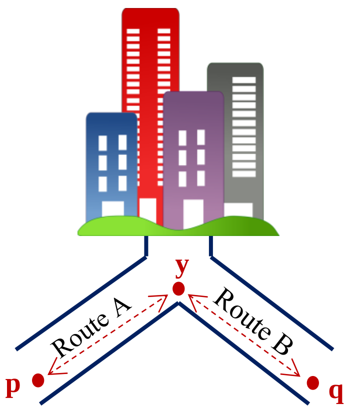


Fig. 3.2 Mental image depiction of S13 and S14.

$$L(\_, Route\_A, p, y, A_{12}, G_s, \_)\Pi L(\_, Route\_B, q, y, A_{12}, G_s, \_)\Lambda city(y)\Lambda p \neq q \qquad \text{(eq. 3.3)}$$

$$L(\_, Route\_A, y, p, A_{12}, G_s, \_)\Pi L(\_, Route\_B, y, q, A_{12}, G_s, \_)\Lambda city(y)\Lambda p \neq q \qquad \text{(eq. 3.4)}$$

For another example of Spatial Change Event, please consider S15 that Fig. 3.3 concerns the perception of the formation of multiple objects, where FAO runs along an imaginary object so called “Imaginary Space Region (ISR)”.

(S15) The square is between the triangle and the circle.

$$(L(\_, y, x_1, x_2, A_{12}, G_s, \_)\Pi L(\_, y, p, p, A_{13}, G_s, \_))$$
$$\bullet(L(\_, y, x_2, x_3, A_{12}, G_s, \_)\Pi L(\_, y, q, q, A_{13}, G_s, \_))\Lambda ISR(y)$$
$$\Lambda p = q\Lambda triangle(x_1)\Lambda square(x_2)\Lambda circle(x_3) \quad \text{(eq. 3.5)}$$

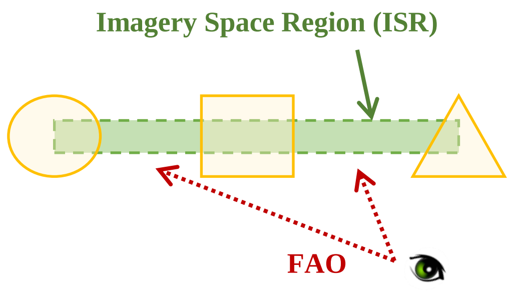


Fig. 3.3 Mental image depiction of S15.

Fig. 3.3 is mental image of S15 that preposition "between" is used and formulated as eq. 3.5 or eq. 3.6, corresponding also to such concepts as "row", "line-up", etc.

$$(L(\_, y, x_1, x_2, A_{12}, G_s, \_)\bullet(L(\_, y, x_2, x_3, A_{12}, G_s, \_))$$
$$\Pi L(\_, y, p, p, A_{13}, G_s, \_))\Lambda ISR(y)\Lambda triangle(x_1)\Lambda square(x_2)\Lambda circle(x_3) \quad \text{(eq. 3.6)}$$

In order to employ ISRs and the 9 - intersection model [3.1], topological relations between two objects can be formulated in such expressions, S16 and S17.

(S16) Tom is in the room.

$$L(Tom, x, y, Tom, A_{12}, G_s, \_)\Pi L(Tom, x, In, In, A_{44}, G_t, K_{9IM})\Lambda ISR(x)\Lambda room(y) \quad \text{(eq. 3.7)}$$

$$L(Tom, x, Tom, y, A_{12}, G_s, \_)\Pi L(Tom, x, Cont, Cont, A_{44}, G_t, K_{9IM})\Lambda ISR(x)\Lambda room(y) \quad \text{(eq. 3.8)}$$

(S17) Tom exits the room.

$$L(Tom, Tom, p, q, A_{12}, G_t, \_)\Pi L(Tom, x, y, Tom, A_{12}, G_s, \_)$$
$$\Pi L(Tom, x, In, Dis, A_{44}, G_t, K_{9IM})\Lambda ISR(x)\Lambda room(y)\Lambda p \neq q \quad \text{(eq. 3.9)}$$

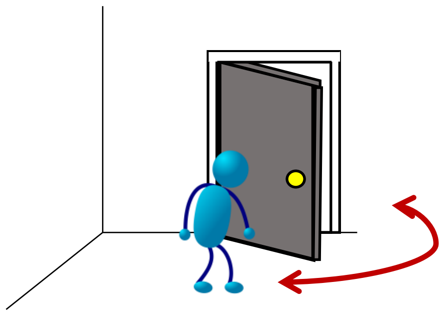
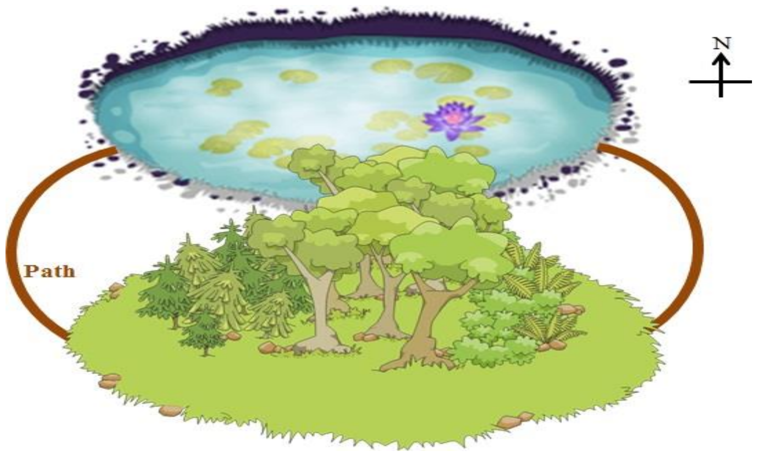


Fig. 3.4 Mental image depiction of S16 and S17.

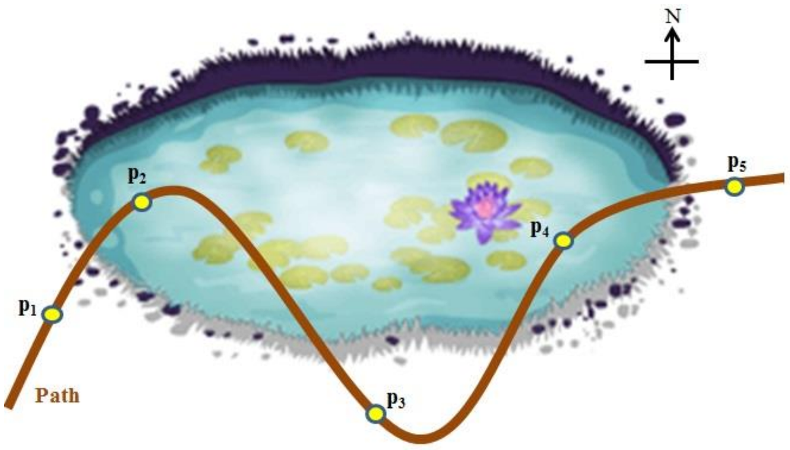
From above, eq. 3.7 and eq. 3.8 represent $\boldsymbol{L_{md}}$ expression of S16, while eq. 3.9 is the interpretation of S17, where "*In*", "*Cont*" and "*Dis*" refer to "inside", "contains" and "disjoint" of the attribute "Topology (A_{44})" with the standard "9 - intersection model (K_{9IM})", respectively. Please remark that, these topological values are given as 3×3 matrices with each element equal to 0 or 1 and therefore, for example, "*In"* and "*Cont"* are transposes each other.

The mathematically rigid topology between two objects must be determined with the perfect knowledge of their insides, outsides and boundaries [3.1]. Ordinary people, however, would often comment on matters without knowing all about them. This is the very case when they encounter an unknown object too large to observe at a glance just like a road in a strange country.

For example, Fig. 3.5(a) shows bird's-eye view of a path that is partly hidden by woods.



(a)



(b)

Fig. 3.5 Delicate topological relations:

(a) path partially hidden by woods and (b) path winding in-out-in-out of swamp.

In this case, the topological relation between the path as a whole and the swamp/woods depends on how the path starts and ends in the woods, but people could utter such sentences as S18 and S19 about this scene. Actually, these sentences refer to such events that reflect certain temporal changes in the topological relation between the swamp/woods and the FAO running along the path.

(S18) The path goes into the swamp/woods.

(S19) The path comes out of the swamp/woods.

Therefore, their conceptual descriptions are to be given as eq. 3.10 and eq. 3.11, respectively.

$$\begin{gathered} L(\_, z, p, q, A_{12}, G_s, \_) \Pi L(\_, x, y, z, A_{12}, G_s, \_) \\ \Pi L(\_, x, Dis, In, A_{44}, G_s, K_{9IM}) \Lambda ISR(x) \\ \Lambda \{swamp(y)/woods(y)\} \Lambda path(z) \Lambda p \neq q \end{gathered} \qquad \text{(eq. 3.10)}$$

$$\begin{gathered} L(\_, z, p, q, A_{12}, G_s, \_) \Pi L(\_, x, y, z, A_{12}, G_s, \_) \\ \Pi L(\_, x, In, Dis, A_{44}, G_s, K_{9IM}) \Lambda ISR(x) \\ \Lambda \{swamp(y)/woods(y)\} \Lambda path(z) \Lambda p \neq q \end{gathered} \qquad \text{(eq. 3.11)}$$

For another example, Fig. 3.5(b), a portrayed image of S20, shows a more complicated spatial event in topology that can be formulated as eq. 3.12.

$$L(\_,z,y,x,A_{12},G_s,\_)\Pi((L(\_,x,p_1,p_2,A_{12},G_s,\_)\Pi(L(\_,z,Dis,In,A_{44},G_s,K_{9IM}))$$
$$\bullet(L(\_,x,p_2,p_3,A_{12},G_s,\_)\Pi L(\_,z,In,Dis,A_{44},G_s,K_{9IM}))$$
$$\bullet(L(\_,x,p_3,p_4,A_{12},G_s,\_)\Pi L(\_,z,Dis,In,A_{44},G_s,K_{9IM}))$$
$$\bullet\big(L(\_,x,p_4,p_5,A_{12},G_s,\_)\Pi L(\_,z,In,Dis,A_{44},G_s,K_{9IM})))$$
$$\Lambda path(x)\Lambda swamp(y)\Lambda ISR(z) \qquad \text{(eq. 3.12)}$$

Conventional approaches to spatial or 3D language understanding have inevitably employed a tremendously great number of axioms such as eq. 3.13. It is noticeable that these axioms are part of the definition of “between” valid only for verbalized directions such as “left” and “above” and that actually much more axioms should be needed for other directions such as “before” and “behind”.

$$right(x,y) \equiv left(y,x)$$
$$above(x,y) \equiv under(y,x)$$
$$above(y,x)\&above(x,z) \supset between(x,y,z)$$
$$right(y,x)\&right(x,z) \supset between(x,y,z) \qquad \text{(eq. 3.13)}$$

Distinguishably, MIDST gives the definition of “between” in such a simple and language-free formula as the underlined part of eq. 3.14 (as same as eq. 3.6) and moreover that is applicable to every direction, whether or not verbalized. The concepts of 40 English prepositions, so-called, spatial prepositions such as “along” were analyzed and formulated in accordance with MIDST. To be most remarkable, the concepts of spatial prepositions are defined as 4D images in MIDST but not as 3D (i.e. 4D exclude “time”) images in conventional approaches.

$$\underline{(L(\_,y,x_1,x_2,A_{12},G_s,\_)\bullet(L(\_,y,x_2,x_3,A_{12},G_s,\_))}$$
$$\underline{\Pi L(\_,y,p,p,A_{13},G_s,\_))\Lambda ISR(y)}$$
$$\Lambda triangle(x_1)\Lambda square(x_2)\Lambda circle(x_3) \qquad \text{(eq. 3.14)}$$

3.2 Translation Process between Natural Language Expression and *Lmd*

Natural language and ***Lmd*** are translated into each other via dependency tree employed in conventional natural language processing for grammatical description. The bidirectional translation between dependency tree and ***Lmd*** is operated by mapping rules assigned to functional words such as verbs and prepositions indicating how their context in NL should be mapped into or generated from the counterpart in ***Lmd***. Here, the mapping rule of a word *W* is generalized as eq. 3.15.

$$Context\ governed\ by\ W \Leftrightarrow Concept\ of\ W\ in\ \boldsymbol{Lmd} \qquad \text{(eq. 3.15)}$$

For example, the mapping rules of “with”, “at”, “move”, “carry”, “run”, “take” and “bring back” are given by eq. 3.16 – eq. 3.24, where $\Lambda_t = (A_{12}, G_t, k)$ and $\Lambda_s = (A_{12}, G_s, k)$. Hereafter, for the sake of simplicity, explicit indications of the quantifiers (i.e., ∀ and ∃) and the relations among variables (e.g., $x \neq y$) are often omitted without risk of confusion.

$$x\ (be)\ with\ y \Leftrightarrow L(x, y, x, x, \Lambda_t) \qquad \text{(eq. 3.16)}$$

$$x\ (be)\ at\ y \Leftrightarrow L(x, x, y, x, \Lambda_t) \qquad \text{(eq. 3.17)}$$

$$x\ move\ y\ from\ p\ to\ q \Leftrightarrow L(x, y, p, q, \Lambda_t) \qquad \text{(eq. 3.18)}$$

$$x\ carry^1\ y\ from\ p\ to\ q \Leftrightarrow L(x, x, p, q, \Lambda_t) \Pi L(x, y, p, q, \Lambda_t) \qquad \text{(eq. 3.19)}$$

$$x\ carry^2\ y\ from\ p\ to\ q \Leftrightarrow L(x, y, x, x, \Lambda_t) \Pi L(x, x, p, q, \Lambda_t) \qquad \text{(eq. 3.20)}$$

$$x\ run\ from\ p\ to\ q \Leftrightarrow L(x, x, p, q, \Lambda_t)\ \text{(Temporal Change Event)} \qquad \text{(eq. 3.21)}$$

$$x\ run\ from\ p\ to\ q \Leftrightarrow L(x, x, p, q, \Lambda_s)\ \text{(Spatial Change Event)} \qquad \text{(eq. 3.22)}$$

$$x\ take\ y\ from\ z \Leftrightarrow L(x, y, z, x, \Lambda_t) \qquad \text{(eq. 3.23)}$$

$$x\ bring\ back\ y\ to\ p\ from\ q \Leftrightarrow L(x, x, p, q, \Lambda_t) \bullet \chi$$
$$\bullet (L(x, y, x, x, \Lambda_t) \Pi L(x, x, q, p, \Lambda_t))$$
$$\text{(where } \chi \text{ is an } \boldsymbol{L_{md}} \text{ expression)} \qquad \text{(eq. 3.24)}$$

[1,2] Please remark that, “carry” can be defined in the two ways as shown in eq. 3.19 and eq. 3.20.

Consider to interpret S20 in $\boldsymbol{L_{md}}$.

(S20) Tom carries the book from Town to University.

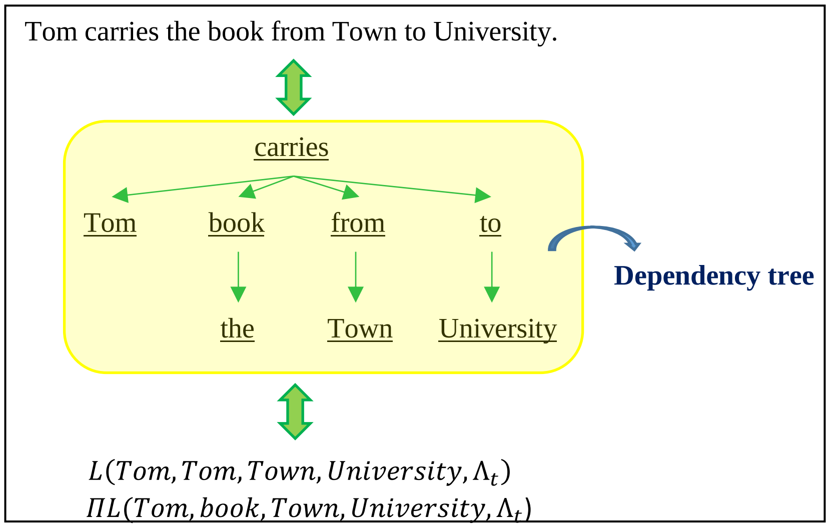

Tom carries the book from Town to University.
carries
Tom
book
from
to
the
Town
University
Dependency tree
$L(Tom, Tom, Town, University, \Lambda_t)$
$\Pi L(Tom, book, Town, University, \Lambda_t)$

Fig. 3.6 Bidirectional translation between NL and $\boldsymbol{L_{md}}$ via Dependency tree.

Fig. 3.6 shows the bidirectional translation between S20 and its semantic interpretation (eq. 3.25) via the dependency tree formulated as eq. 3.26.

$$L(Tom, Tom, Town, University, \Lambda_t)\Pi L(Tom, book, Town, University, \Lambda_t) \qquad \text{(eq. 3.25)}$$

$$carries(Tom, book(the), from(Town), to(University)) \qquad \text{(eq. 3.26)}$$

The formula eq. 3.24 reads "Tom moves from Town to University by himself <u>and simultaneously (Π)</u> Tom moves the book from Town to University" and it can be depicted as Fig. 3.7.

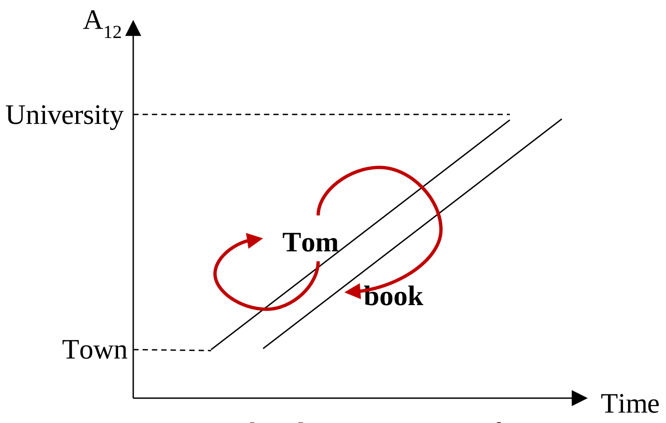


Fig. 3.7 Graphical interpretation of eq. 3.24.

3.3 Evaluation of Semantic Interpretation

Such a semantic interpretation as eq. 3.25 is to be evaluated about its plausibility. For this purpose, entity word concepts such as “cat” and “car” are exclusively utilized. The concept of an entity can be defined as a set of locus formulas representing its potentiality at every attribute. For example, “Tom”, “book” and “hill” can be defined as eq. 3.27 – eq. 3.29 at the attribute “Physical Location (A_{12})” with $g = G_t$, where the symbol + or – denotes whether the following image is positive (i.e., probable) or negative (i.e., improbable), respectively.

$$\underline{\{+L(Tom, Tom, p, q, \Lambda_t),} +L(Tom, x, p, q, \Lambda_t), \ldots\} \qquad \text{(eq. 3.27)}$$

$$\{-L(Book, Book, p, q, \Lambda_t), +L(y, Book, p, q, \Lambda_t), \ldots\} \qquad \text{(eq. 3.28)}$$

$$\{-L(z, Hill, p, q, \Lambda_t), -L(Hill, z, p, q, \Lambda_t), \ldots\} \qquad \text{(eq. 3.29)}$$

The meaning of each formula in these is as follows.

$+L(Tom, Tom, p, q, \Lambda_t)$:	“Tom moves by himself” is probable.
$+L(Tom, x, p, q, \Lambda_t)$:	“Tom moves something” is probable.
$-L(Book, Book, p, q, \Lambda_t)$:	“A book moves itself” is improbable.
$+L(y, Book, p, q, \Lambda_t)$:	“A book is moved” is probable.
$-L(z, Hill, p, q, \Lambda_t)$:	“A hill is moved” is improbable.
$-L(Hill, z, p, q, \Lambda_t)$:	“A hill moves something” is improbable.

The definitions of “Tom” and “book” do not conflict with eq. 3.25 and therefore the semantic interpretation of S20 proves to be plausible. For another example, consider S21.

(S21) Tom was on the hill, moving about.

This sentence is syntactically ambiguous in two ways as shown in Fig. 3.8, that is, both “Tom” and “hill” can be the subject of “moving” grammatically. However, “Tom” is more plausible because “Tom moves (by himself)” is probable but “a hill moves” is improbable.

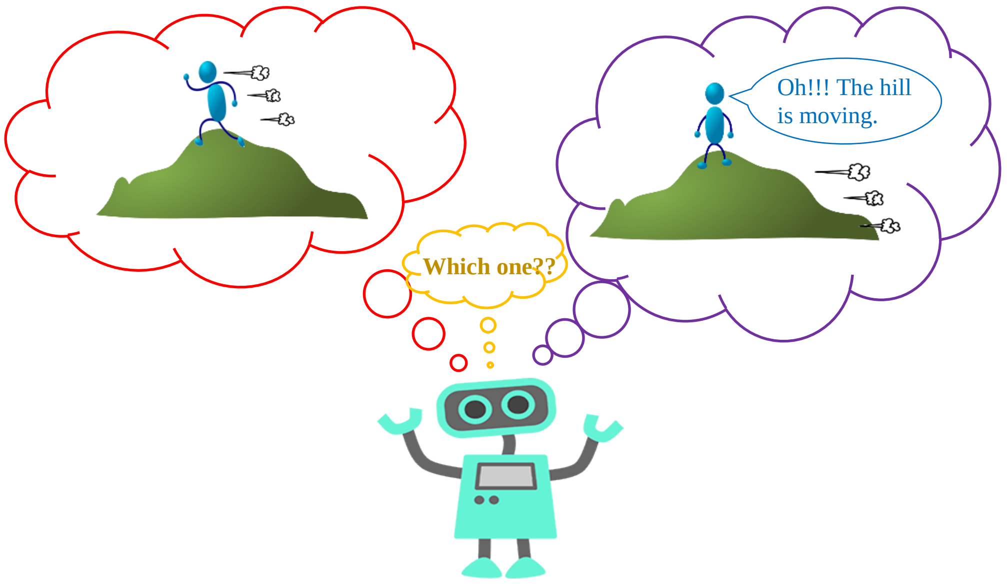


Fig. 3.8 Ambiguities in S21

For the semantic evaluation described above, a simple pattern matching operation (PMO), so-called unification in conventional AI, is employed to search one $\boldsymbol{L_{md}}$ expression such as eq. 3.25 for another $\boldsymbol{L_{md}}$ expression such as the underlined part of eq. 3.27. This kind of semantic evaluation can prevent cognitive robots from meaningless and endless work due to such an anomalous command as S21 given by mischievous people.

CHAPTER 4

PROBLEM FINDING AND SOLVING IN L_{md}

This section is about problem finding and solving method in this work, consisting of "definition of problem and task", "creation of problem finding and solving", and "maintenance problem finding and solving", respectively.

Imagine such a scene that a robot is working in order to achieve its mission assigned by some people. The robot must find and solve problems concerning the mission. The problems here are considered to belong to the category *Exploration problem* introduced in Heylighen [4.1] for robots partially or wholly ignorant of their environments. Such problems can be classified roughly into two subcategories as follows.

- Creation Problem (P_C): house building, food cooking, etc.
- Maintenance Problem (P_M): fire extinguishing, room cleaning, etc**.**

In general, a P_M is relatively simple one that the robot can find and solve autonomously while a P_C is relatively difficult one that is given to the robot, possibly, by humans and to be solved in cooperation with them**.**

4.1 Definition of Problem and Task

The robot must determine its task to solve a problem in the world. In general, during such problem solving, the robot needs to interpolate some transit event $X_T$between the two events, namely, *Current Event* (X_C) and *Goal Event* (X_G) as shown by (eq. 4.1).

$$X_C \bullet X_T \bullet X_G \qquad \text{(eq. 4.1)}$$

According to this formalization, a problem X_P is defined as $X_T \bullet X_G$ and a task for the robot is defined as its realization in the same way as the conventional AI referred to by Russell and Norvig [4.2], etc., where a problem is defined as the difference or gap between a *Current State* and a *Goal State* and a task as its cancellation. Here, the term *Event* is preferred to the term *State*, and instead *State* is defined as static *Event* which corresponds to a level locus. The events in the world are described as loci in certain attribute spaces and a problem is to be detected by the unit of atomic locus as *event gaps*. For example, employing such a postulate of *Continuity in attribute values* (P$_{CAV}$), the event gap X in (eq. 4.2) is to be inferred as (eq. 4.3).

$$L(x, y, q_1, q_2, a, g, k) \bullet X \bullet L(z, y, q_3, q_4, a, g, k) \qquad \text{(eq. 4.2)}$$

$$L(z', y, q_2, q_3, a, g, k) \qquad \text{(eq. 4.3)}$$

4.2 Creation Problem Finding and Solving

Consider such a verbal command as S22 uttered by a human. Its interpretation is given by (eq. 4.4) as the goal event X_G concerning the attribute, *Height* (*A*$_{03}$). If the current event X_C is given by (eq. 4.5), then (eq. 4.6) with the transit event X_T underlined can be inferred as the problem corresponding to S22.

(S22) Keep balloon C_7 flying 7-9 meters high.

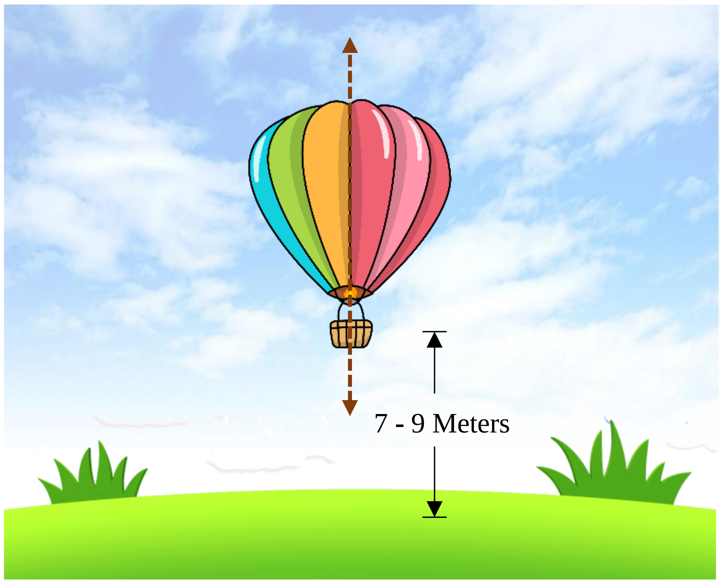


Fig. 4.1 Mental image depiction of S22.

$$L(z, C_7, q, q, A_{03}, G_t, k) \bullet balloon(C_7) \Lambda 7m \leq q \leq 9m \qquad \text{(eq. 4.4)}$$

$$L(x, C_7, p, p, A_{03}, G_t, k) \Lambda balloon(C_7) \qquad \text{(eq. 4.5)}$$

$$L(z_1, C_7, p, q, A_{03}, G_t, k) \bullet L(z, C_7, q, q, A_{03}, G_t, k)$$
$$\Lambda balloon(C_7) \Lambda 7m \leq q \leq 9m \qquad \text{(eq. 4.6)}$$

For this problem, the robot is to execute a task deploying a certain height sensor and actors, z_1 and z. The selection of the actor z_1 is performed as follows:

If 9m-p <0 then z_1 *is a sinker, otherwise*

if 7m-p >0 then z_1 *is a raiser, otherwise*

$7m \leq p \leq 9m$ and no actor is deployed as z_1.

The selection of z is a task in case of P_M described below.

4.3 Maintenance Problem Finding and Solving

In general, the goal event X_G for a P_M is that for another P_C such as S22 given possibly by humans and solved by the robot in advance. That is, the task in this case is to autonomously restore the goal event X_G created in advance to the current event X_C as shown in eq. 4.7, where the transit event X_T is the reversal of such X_{-T}that has been already detected as *abnormal* by the robot.

For example, if X_G is given by eq. 4.4 in advance, X_T is also represented as the underlined part of eq. 4.6 while X_{-T}as eq. 4.8. Therefore the task here is quite the same that was described in the previous subsection 4.2.

$$X_G \bullet X_{-T} \bullet X_C \bullet X_T \bullet X_G \qquad \text{(eq. 4.7)}$$

$$L(z_1, C_7, q, p, A_{03}, G_t, k) \Lambda balloon(C_7) \qquad \text{(eq. 4.8)}$$

CHAPTER 5

APPLICATIONS TO NATURAL LANGUAGE UNDERSTANDING

In this work, we have been developing a Mental – Image Based Understanding system (MBU) and Conversation Management System (CMS) in Python for simulating human-robot interaction in NL [5.1] that the details are as the following.

5.1 Mental – Image Based Understanding

Mental – Image Based Understanding (MBU) is a system based on MIDST to simulate human mental imagery, focusing on 4D expressions to obtain the results in an acceptable way (like Q&A) where mental images are represented by the formal language $\boldsymbol{L_{md}}$.

For this work on MBU, $\boldsymbol{L_{md}}$ was applied to three types of stimulus sentences as follows, where SS, PrP, PaP and C denote "simple sentence", "present particle construction", "past particle construction" and "conjunction", respectively.

- [Type I] Simple sentence + Present particle construction (SS + PrP)
  For example:
  (S23) Tom was with the book in the bus *running* from Town to University. (= S9)

- [Type II] Simple sentence + Past particle construction (SS + PaP)
  For example:
  (S24) Tom was with the book in the car *driven* from Town to University by Mary.

- [Type III] Simple sentence + Conjunction + Simple sentence (SS + C + SS)
  For example:
  (S25) Tom kept the book in a box *before* he drove the car from Town to University with the box.

As easily convinced, S23 – S25 are syntactically ambiguous that may be rather easy for humans to understand, but it is not the case for robots.

For example, consider S23. How can the machine know who/what was running from Town to University? —Tom, or book, or bus? Here, to see its syntactic possibilities, Dependency Grammar (DG) is employed to determine the relations between head words and their dependents. In principle, S23 can have twelve possible dependency trees, that is, syntactically ambiguous in twelve ways as shown in Fig.5.1. This can be formulated by a set of local dependencies such as eq. 5.1, where each pair of parentheses is for the alternatives causing the syntactic ambiguity.

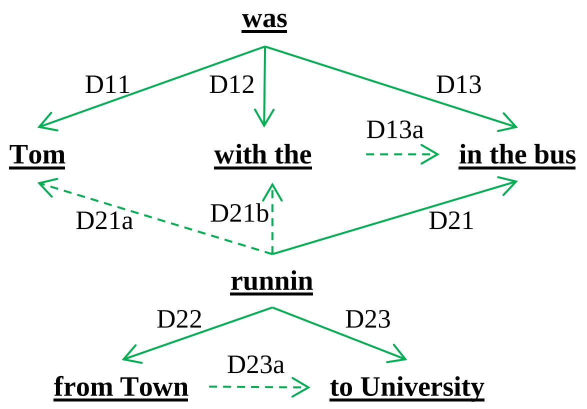


Fig. 5.1 Dependencies possible for S23.

$$\{D11,\ D12,\ (D13|D13a),\ (D21|D21a|D21b),\ D22,\ (D23|D23a)\} \qquad \text{(eq. 5.1)}$$

According to our psychological experiment, almost all the human subjects reach very easily the most plausible image (i.e., Fig. 5.2) that corresponds directly to the dependency tree defined by eq. 5.2 and can be formulated as eq. 5.3 in ***L****md*.

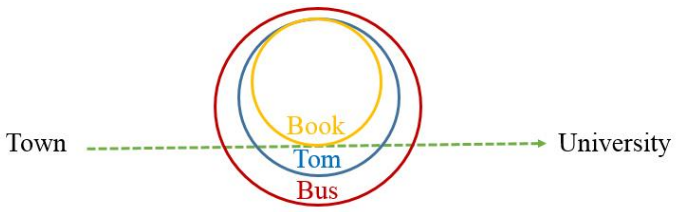


Fig 5.2 Highly abstract picture of S23.

$$\{D11,\ D12,\ D13,\ D21,\ D22,\ D23\} \qquad \text{(eq. 5.2)}$$

$$L(Tom, Book, Tom, Tom, \Lambda_t)\Pi L(z, Tom, Bus, Bus, \Lambda_t)$$
$$\Pi L(Bus, Bus, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.3)}$$

Quite in the same way, the most plausible interpretations of S24 and S25 are given by eq. 5.4 and eq. 5.5, respectively.

$$L(Tom, Book, Tom, Tom, \Lambda_t)\Pi L(z, Tom, Car, Car, \Lambda_t)$$
$$\Pi L(Mary, Mary, Car, Car, \Lambda_t)\Pi L(Mary, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.4)}$$

$$L(Tom, Book, Box, Box, \Lambda_t)\bullet(L(Tom, Tom, Car, Car, \Lambda_t)$$
$$\Pi L(Tom, Car, Town, Univ., \Lambda_t)\Pi L(Tom, Box, Tom, Tom, \Lambda_t)) \qquad \text{(eq. 5.5)}$$

Every semantic interpretation (e.g., eq. 5.3) of an NL expression (i.e., S23) is generated by unifying the word meanings according to its corresponding dependency tree (i.e., eq. 5.2). In this process, functional words such as verbs and prepositions are employed for structuring the locus formulas, such as eq. 5.6 – 5.7 (for more example, see section 3.2, Translation process between Natural Language expression and L_{md})

$$x \; drives \; y \; from \; p \; to \; q \Leftrightarrow L(x, x, y, y, \Lambda_t)\Pi L(x, y, p, q, \Lambda_t) \qquad \text{(eq. 5.6)}$$

$$x \; keeps \; y \; in \; z \Leftrightarrow L(x, y, z, z, \Lambda_t) \qquad \text{(eq. 5.7)}$$

On the other hand, entity names such as “Tom”, “book” and “bus” are non-functional but utilized for disambiguation in syntactic dependency. Our psychological experiment (for more information, see APPENDIX B) revealed that the subjects remembered their own experiences in association with the entity names and that they selected the dependency corresponding to their most familiar experience among all the possibilities (The detail of disambiguation process is described in section 3.3, Evaluation of Semantic Interpretation).

$$\{+L(Bus, Bus, p, q, \Lambda_t), +L(Bus, x, p, q, \Lambda_t), +L(x, Human, Bus, Bus, \Lambda_t), \dots\} \qquad \text{(eq. 5.8)}$$

It is sure that the subjects reached the most plausible interpretation eq. 5.3 almost unconsciously by using these evoked images for disambiguation. For example, the image for D13 in Fig. 5.1 is more probable than that for D13a because of eq. 3.26 and eq. 5.8, D21b is improbable because of eq. 3.27, and the combination of D13 and D21a results in somewhat strange image that Tom was running in the bus, and therefore D21a is seldom selected.

Furthermore, as well as disambiguation, question - answering in MBU was simulated, which is performed by “Pattern Matching (PM)” between the locus formulas of an assertion and a question, for example, eq. 5.3 of S23 and eq. 5.9 of S26.

(S26) Did Tom carry the book from Town to University?

$$L(z, Book, Tom, Tom, \Lambda_t)\Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.9)}$$

Actually, "carry" is defined in the two ways as eq. 3.19 and eq. 3.20 but, from now on, only either of them is considered for the sake of simplicity. For example, eq. 5.9 adopts eq. 3.20.

If eq. 5.3 includes eq. 5.9 as is, the answer is positive, but this is not the case. That is, direct trial of PM to the locus formulas eq. 5.3 – eq. 5.5 does not always lead to the desirable outcomes. Therefore, a number of postulates and inference rules must be introduced. The postulates such as P1 – P4 are formulas representing pieces of people's commonsense knowledge about the world, where "A → B" reads "A implies B" or "if A then B".

- (P1) Postulate of matters as values:
  $L(z, x, p, q, \Lambda_t)\Pi L(w, y, x, x, \Lambda_t) \rightarrow L(z, x, p, q, \Lambda_t)\Pi L(w, y, p, q, \Lambda_t)$

- (P2) Postulate of shortcut in causal chain:
  $L(z, x, p, q, \Lambda_t)\Pi L(w, y, x, x, \Lambda_t) \rightarrow L(z, x, p, q, \Lambda_t)\Pi L(z, y, p, q, \Lambda_t)$

- (P3) Postulate of conservation of values in time:
  $(L(z, x, p, p, \Lambda_t)\Pi X_1)\bullet X_2 \rightarrow (L(z, x, p, p, \Lambda_t)\Pi X_1)\bullet(L(z, x, p, p, \Lambda_t)\Pi X_2)$

- (P4) Postulate of continuity in attribute values:
  $L(x, y, p, q_1, \Lambda_t)\bullet X\bullet L(z, y, q_2, r, \Lambda_t) \rightarrow X = L(z', y, q_2, q_3, \Lambda_t)$
  $, where\ q_2 = q_3$

P1 reads that if *"z causes x to move from p to q while w causes y to stay with x"* then "*w causes y to move from p to q"*. Similarly, P2, so that if *"z causes x to move from p to q while w causes y to stay with x"* then "*z causes y to move from p to q as well as x"*.

Distinguished from these two, P3 is conditional. That is, it is valid only when X_2 does not contradict with "$L(z, x, p, p, \Lambda_t)$". While P4 is used to detect event gap as problem finding and its cancellation as problem solving (as shown in Fig. 5.3).

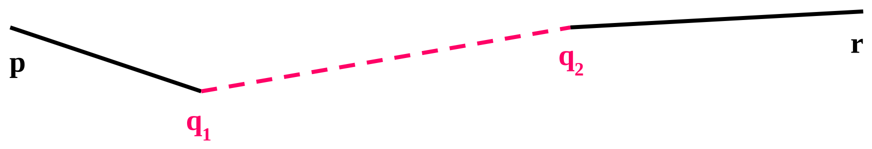


Fig. 5.3 Postulate of continuity in attribute values.

On the other hand, inference rules such as CS, SS and SC are introduced as follows.

- (CS) Commutativity law of Π:
  $X\Pi Y \leftrightarrow Y\Pi X$
- (SS) Simplification law of Π:
  $X\Pi Y \rightarrow X$
- (SC) Simplification law of $\bullet$:
  $X \bullet Y \rightarrow X, \quad X \bullet Y \rightarrow Y$

In order to answer the question S26 to S23, PM is used to compare eq. 5.3 and eq. 5.9 as follows:

Apply CS to eq. 5.3:

$$(eq.5.3) \Rightarrow L(Tom, Book, Tom, Tom, \Lambda_t)$$
$$\underline{\Pi L(Bus, Bus, Town, Univ., \Lambda_t)}$$
$$\underline{\Pi L(Bus, Bus, Town, Univ., \Lambda_t)} \qquad \text{(eq. 5.10)}$$

Apply P1 to eq. 5.10 (at the underlined part):

$$(eq.5.10) \Rightarrow L(Tom, Book, Tom, Tom, \Lambda_t)$$
$$\Pi L(Bus, Bus, Town, Univ., \Lambda_t)$$
$$\Pi L(z, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.11)}$$

Apply SS to eq. 5.11:

$$(eq.5.11) \Rightarrow L(Tom, Book, Tom, Tom, \Lambda_t)$$
$$\Pi L(z, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.12)}$$

Apply P2 to eq. 5.12:

$$(eq.5.12) \Rightarrow L(z, Book, Tom, Tom, \Lambda_t)$$
$$\Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.13)}$$

The PM process finds that eq. 5.9 = eq. 5.13, and then it is proved that Tom carried the book from Town to University.

For another example, consider the stimulus sentence S24 and the question S27.

(S27) Did Mary carry the car from Town to University?

Adopting eq. 3.19 for "carry", the interpretation of S27 can be given by eq. 5.14.

$$L(z, Mary, Town, Univ., \Lambda_t)\Pi L(Mary, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.14)}$$

In order to answer the question S27 to S24, PM works as follows, where "$A \Rightarrow B$" reads "B is deduced from A".

Apply CS to eq. 5.4:

$$\begin{aligned}(eq. 5.4) \Rightarrow\ & L(Tom, Book, Tom, Tom, \Lambda_t)\\ & \Pi L(z, Tom, Car, Car, \Lambda_t)\\ & \Pi L(Mary, Car, Town, Univ., \Lambda_t)\\ & \Pi L(Mary, Mary, Car, Car, \Lambda_t) \qquad \text{(eq. 5.15)}\end{aligned}$$

Apply P1 to eq. 5.15:

$$\begin{aligned}(eq. 5.15) \Rightarrow\ & L(Tom, Book, Tom, Tom, \Lambda_t)\\ & \Pi L(z, Tom, Car, Car, \Lambda_t)\\ & \Pi L(Mary, Car, Town, Univ., \Lambda_t)\\ & \Pi L(Mary, Mary, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.16)}\end{aligned}$$

Apply CS to eq. 5.16:

$$\begin{aligned}(eq. 5.16) \Rightarrow\ & L(Tom, Book, Tom, Tom, \Lambda_t)\\ & \Pi L(z, Tom, Car, Car, \Lambda_t)\\ & \Pi L(Mary, Car, Town, Univ., \Lambda_t)\\ & \Pi L(Mary, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.17)}\end{aligned}$$

Apply P2 to eq. 5.17:

$$
\begin{aligned}
(eq. 5.17) \Rightarrow\ & L(Tom, Book, Tom, Tom, \Lambda_t) \\
& \Pi L(z, Tom, Car, Car, \Lambda_{t)} \\
& \Pi L(z, Mary, Town, Univ., \Lambda_{t)} \\
& \Pi L(Mary, Car, Town, Univ., \Lambda_t)
\end{aligned}
\qquad \text{(eq. 5.18)}
$$

Apply SS to eq. 5.18:

$$
\begin{aligned}
(eq. 5.18) \Rightarrow\ & L(z, Mary, Town, Univ., \Lambda_t) \\
& \Pi L(Mary, Car, Town, Univ., \Lambda_{t)}
\end{aligned}
\qquad \text{(eq. 5.19)}
$$

Hence, the PM proves that eq. 5.14 = eq. 5.19 and it is concluded that Mary carried the bus from Town to University.

For the last example, consider the Type III sentence S25, and the question S28 whose interpretation is given by eq. 5.20.

(S28) Did Tom move the book from Town to University?

$$L(Tom, Book, Town, Univ., \Lambda_t) \qquad \text{(eq. 5.20)}$$

Apply P3 to eq. 5.5:

$$
\begin{aligned}
(eq. 5.5) \Rightarrow\ & L(Tom, Book, Box, Box, \Lambda_t) \\
& \bullet\sout{(L(Tom, Book, Box, Box, \Lambda_t)} \\
& \Pi L(Tom, Tom, Car, Car, \Lambda_t) \\
& \Pi L(Tom, Box, Tom, Tom, \Lambda_t) \\
& \Pi L(Tom, Car, Town, Univ., \Lambda_t))
\end{aligned}
\qquad \text{(eq. 5.21)}
$$

Apply CS to eq. 5.21 several times:

$$
\begin{aligned}
(eq. 5.21) \Rightarrow\ & L(Tom, Book, Box, Box, \Lambda_t) \\
& \bullet(L(Tom, Car, Town, Univ., \Lambda_t) \\
& \Pi L(Tom, Tom, Car, Car, \Lambda_t) \\
& \Pi L(Tom, Box, Tom, Tom, \Lambda_t) \\
& \Pi L(Tom, Book, Box, Box, \Lambda_t))
\end{aligned}
\qquad \text{(eq. 5.22)}
$$

Apply P2 to eq. 5.22:

$$
\begin{aligned}
(eq. 5.22) \Rightarrow\ & L(Tom, Book, Box, Box, \Lambda_t) \\
& \bullet(L(Tom, Car, Town, Univ., \Lambda_t) \\
& \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \\
& \Pi L(Tom, Box, Tom, Tom, \Lambda_t) \\
& \Pi L(Tom, Book, Box, Box, \Lambda_t))
\end{aligned}
\qquad \text{(eq. 5.23)}
$$

Apply P2 to eq. 5.23 twice:

$$
\begin{aligned}
(eq. 5.23) \Rightarrow\ & L(Tom, Book, Box, Box, \Lambda_t) \\
& \bullet(L(Tom, Car, Town, Univ., \Lambda_t) \\
& \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \\
& \Pi L(Tom, Box, Town, Univ., \Lambda_t) \\
& \Pi L(Tom, Book, Town, Univ., \Lambda_t))
\end{aligned}
\qquad \text{(eq. 5.24)}
$$

Apply SS and SC to eq. 5.24:

$$(eq. 5.24) \Rightarrow L(Tom, Book, Town, Uni., \Lambda_t) \qquad \text{(eq. 5.25)}$$

In this way, the system finds that eq. 5.20 is deduced from eq. 5.5.

We have implemented our theory of MBU on a PC in Python, high-level programming language, while it is still experimental and evolving. The next sub-section 5.2 shows some of the results of question - answering by the MBU system. This can understand User's assertions and answer the questions where the locus formulas were given in Polish notation, for example, as $\bullet\Pi ABC$ for $(A\Pi B)\bullet C$.

In the actual implementation, the theorem proving process was simplified as the PM process programmed to apply all the possible postulates to the locus formula of the assertion in advance and detect any match with the question in the assertion (extended by the postulates) by using the inference rules on the way. During PM, the system is to control its awareness in a top - down way driven by the pair of AC and attribute contained in the question, for example, "Book" and "Physical Location (Λ_t)", which is very efficient compared to conventional PM methods without employing any kind of semantic information.

5.2 Application of MBU system

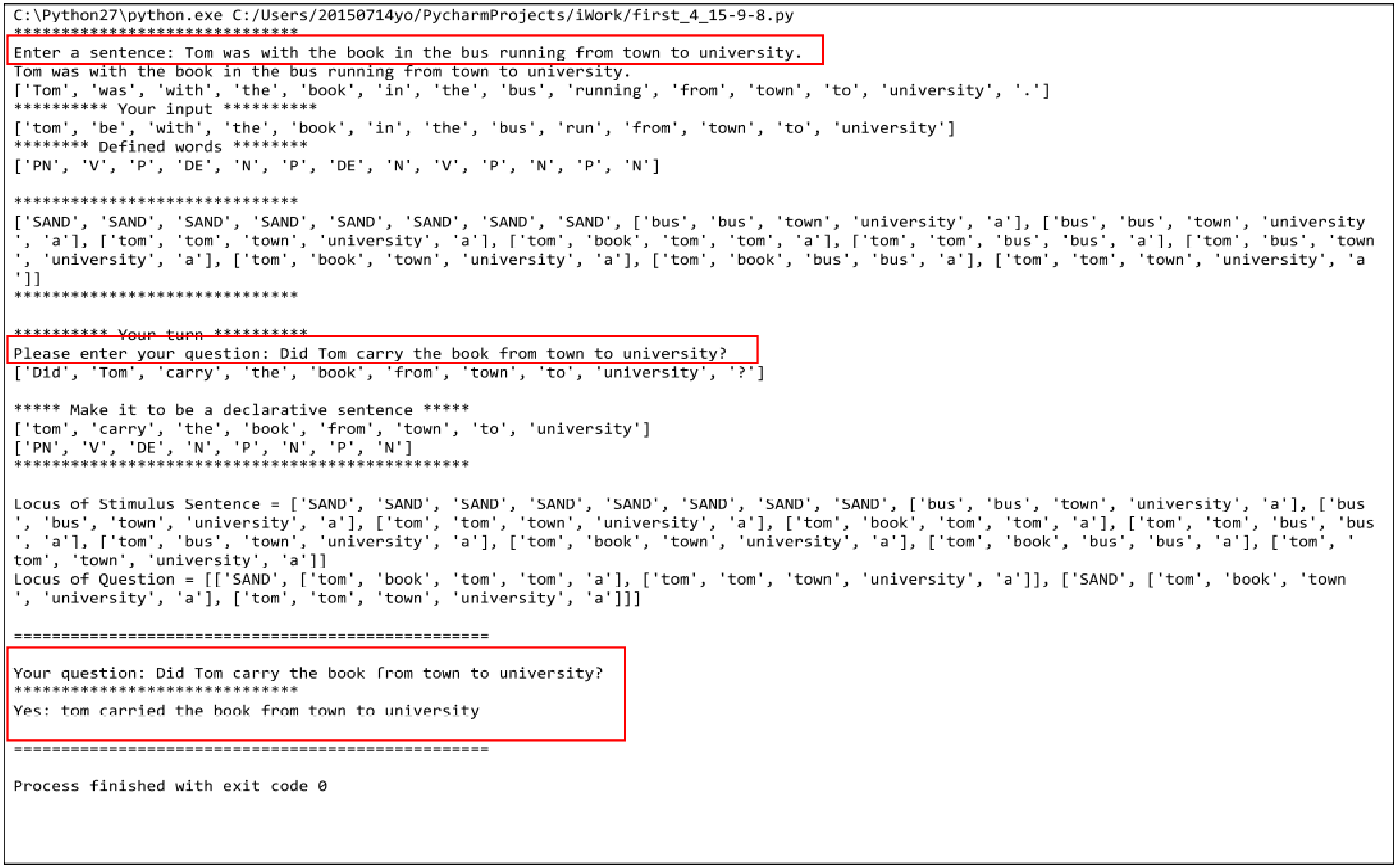
This part shows some examples of MBU system with three - type of stimulus sentence. When user enters an input sentence, the system will execute as the following steps:

- User enter an input sentence, for example "*Tom was with the book in the bus running from town to university.*".
- Interpreting the input sentence to ***L_{md}*** expression.
- Asking for a question from user, e.g. "*Did Tom carry the book from town to university?*".
- Interpreting that question to $\boldsymbol{L_{md}}$ expression.
- Using postulates and inference rules to $\boldsymbol{L_{md}}$ expression of stimulus sentence, then employing pattern matching process to compare $\boldsymbol{L_{md}}$ expressions of input sentence and its question.
- Return output to the user.

Next is our experiment results that shown in Fig. 5.4 – 5.10. Please remark that, red rectangles in each figure refer to input/stimulus sentence, question, and output from the system, respectively.

Stimulus sentence: Tom was with the book in the bus running from town to university.

Question: Did Tom carry the book from town to university?

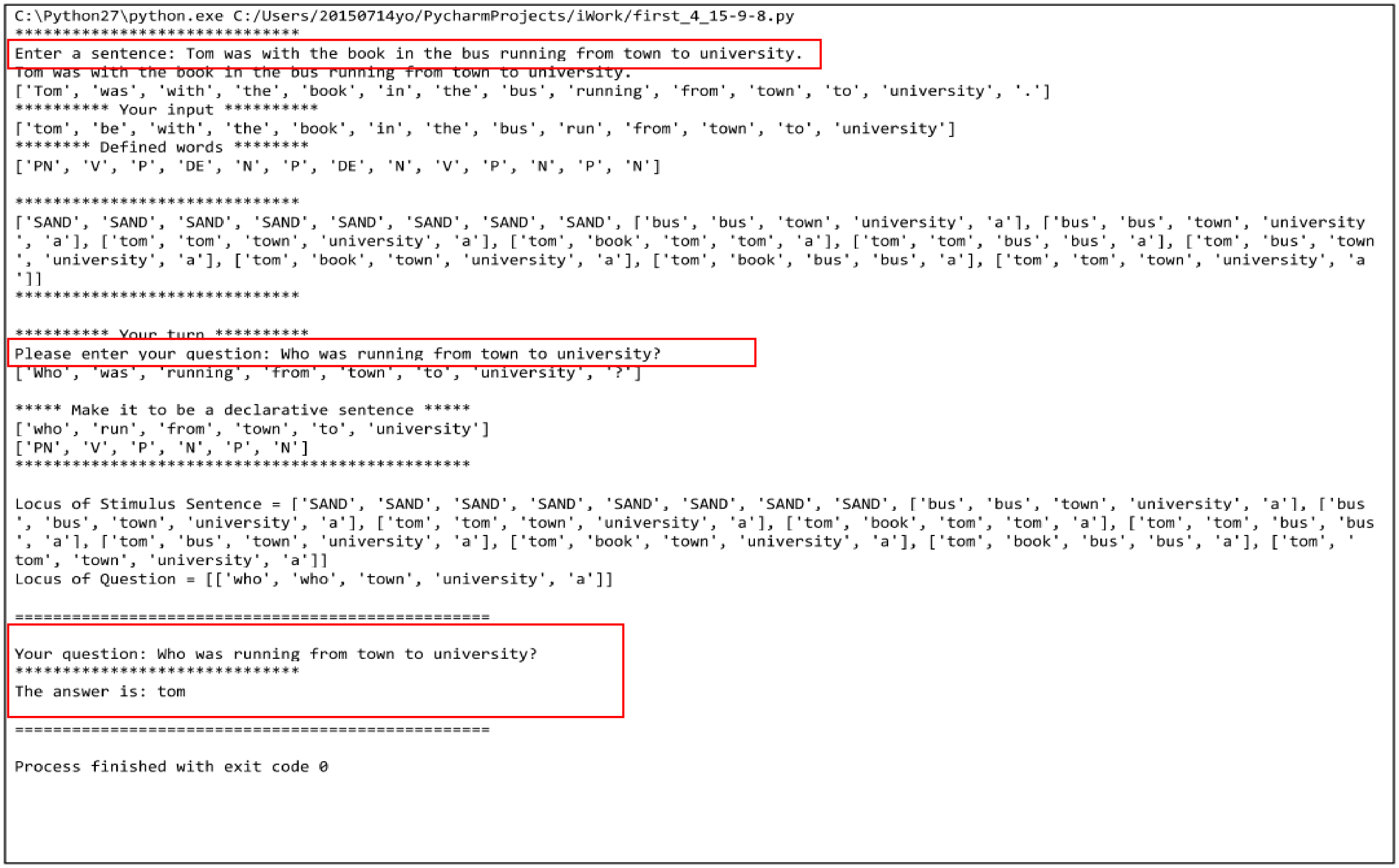


```
C:\Python27\python.exe C:/Users/20150714yo/PycharmProjects/iWork/first_4_15-9-8.py
******************************
Enter a sentence: Tom was with the book in the bus running from town to university.
Tom was with the book in the bus running from town to university.
['Tom', 'was', 'with', 'the', 'book', 'in', 'the', 'bus', 'running', 'from', 'town', 'to', 'university', '.']
********** Your input **********
['tom', 'be', 'with', 'the', 'book', 'in', 'the', 'bus', 'run', 'from', 'town', 'to', 'university']
******** Defined words ********
['PN', 'V', 'P', 'DE', 'N', 'P', 'DE', 'N', 'V', 'P', 'N', 'P', 'N']

******************************
['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['bus', 'bus', 'town', 'university', 'a'], ['bus', 'bus', 'town', 'university
', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'bus', 'bus', 'a'], ['tom', 'bus', 'town
', 'university', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'bus', 'bus', 'a'], ['tom', 'tom', 'town', 'university', 'a
']]
******************************

********** Your turn **********
Please enter your question: Did Tom carry the book from town to university?
['Did', 'Tom', 'carry', 'the', 'book', 'from', 'town', 'to', 'university', '?']

***** Make it to be a declarative sentence *****
['tom', 'carry', 'the', 'book', 'from', 'town', 'to', 'university']
['PN', 'V', 'DE', 'N', 'P', 'N', 'P', 'N']
************************************************

Locus of Stimulus Sentence = ['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['bus', 'bus', 'town', 'university', 'a'], ['bus
', 'bus', 'town', 'university', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'bus', 'bus
', 'a'], ['tom', 'bus', 'town', 'university', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'bus', 'bus', 'a'], ['tom', '
tom', 'town', 'university', 'a']]
Locus of Question = [['SAND', ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'town', 'university', 'a']], ['SAND', ['tom', 'book', 'town
', 'university', 'a'], ['tom', 'tom', 'town', 'university', 'a']]]

==================================================

Your question: Did Tom carry the book from town to university?
******************************
Yes: tom carried the book from town to university

==================================================

Process finished with exit code 0
```

Fig 5.4 1st example of MBU processing for stimulus sentence Type I.

Stimulus sentence: Tom was with the book in the bus running from town to university.

Question: Who was running from town to university?

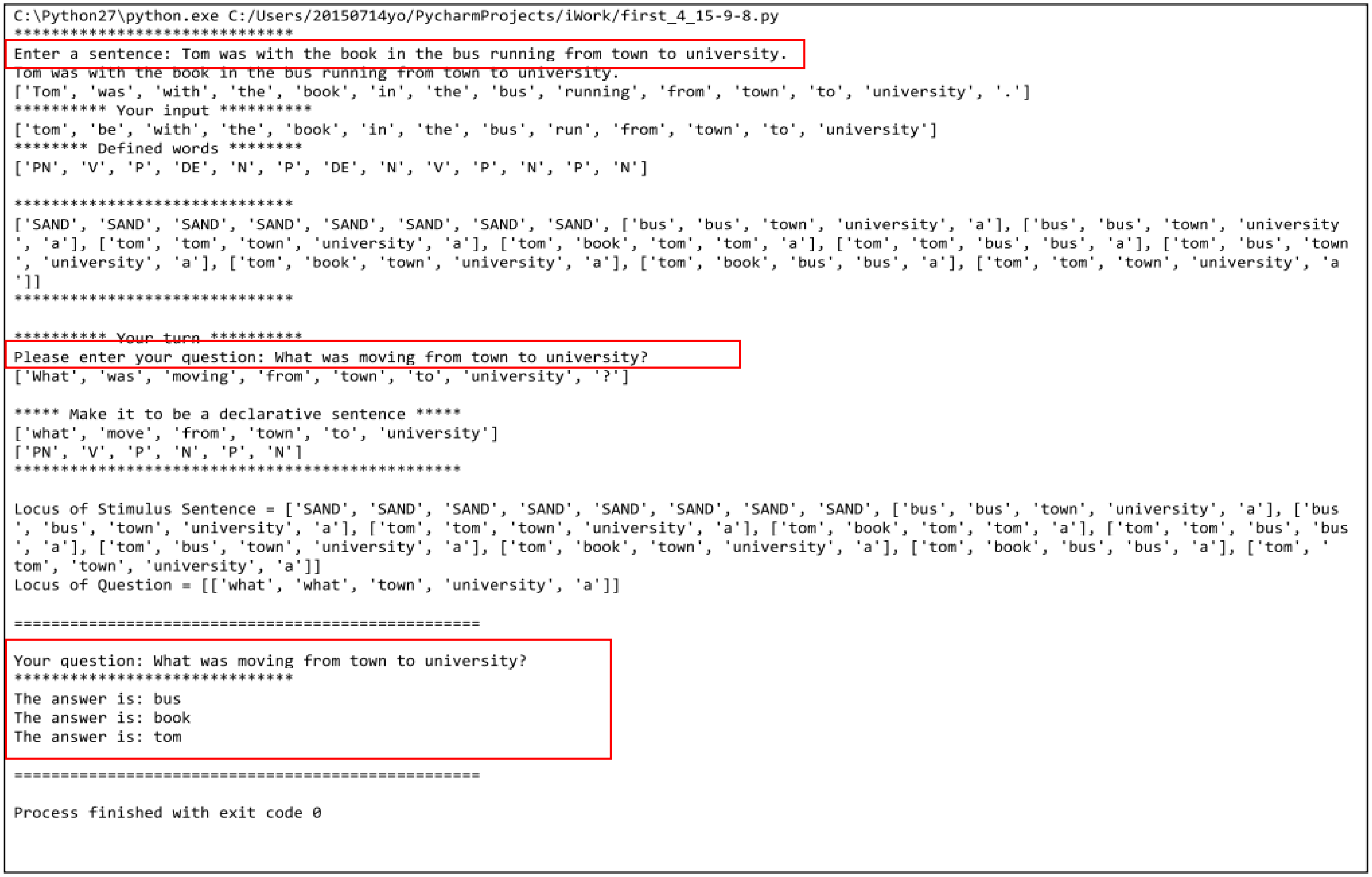


```
C:\Python27\python.exe C:/Users/20150714yo/PycharmProjects/iWork/first_4_15-9-8.py
******************************
Enter a sentence: Tom was with the book in the bus running from town to university.
Tom was with the book in the bus running from town to university.
['Tom', 'was', 'with', 'the', 'book', 'in', 'the', 'bus', 'running', 'from', 'town', 'to', 'university', '.']
********** Your input **********
['tom', 'be', 'with', 'the', 'book', 'in', 'the', 'bus', 'run', 'from', 'town', 'to', 'university']
******** Defined words ********
['PN', 'V', 'P', 'DE', 'N', 'P', 'DE', 'N', 'V', 'P', 'N', 'P', 'N']

******************************
['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['bus', 'bus', 'town', 'university', 'a'], ['bus', 'bus', 'town', 'university
', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'bus', 'bus', 'a'], ['tom', 'bus', 'town
', 'university', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'bus', 'bus', 'a'], ['tom', 'tom', 'town', 'university', 'a
']]
******************************

********** Your turn **********
Please enter your question: Who was running from town to university?
['Who', 'was', 'running', 'from', 'town', 'to', 'university', '?']

***** Make it to be a declarative sentence *****
['who', 'run', 'from', 'town', 'to', 'university']
['PN', 'V', 'P', 'N', 'P', 'N']
************************************************

Locus of Stimulus Sentence = ['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['bus', 'bus', 'town', 'university', 'a'], ['bus
', 'bus', 'town', 'university', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'bus', 'bus
', 'a'], ['tom', 'bus', 'town', 'university', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'bus', 'bus', 'a'], ['tom', '
tom', 'town', 'university', 'a']]
Locus of Question = [['who', 'who', 'town', 'university', 'a']]

==================================================

Your question: Who was running from town to university?
******************************
The answer is: tom

==================================================

Process finished with exit code 0
```

Fig 5.5 2nd example of MBU processing for stimulus sentence Type I.

Stimulus sentence: Tom was with the book in the bus running from town to university.

Question: What was moving from town to university?

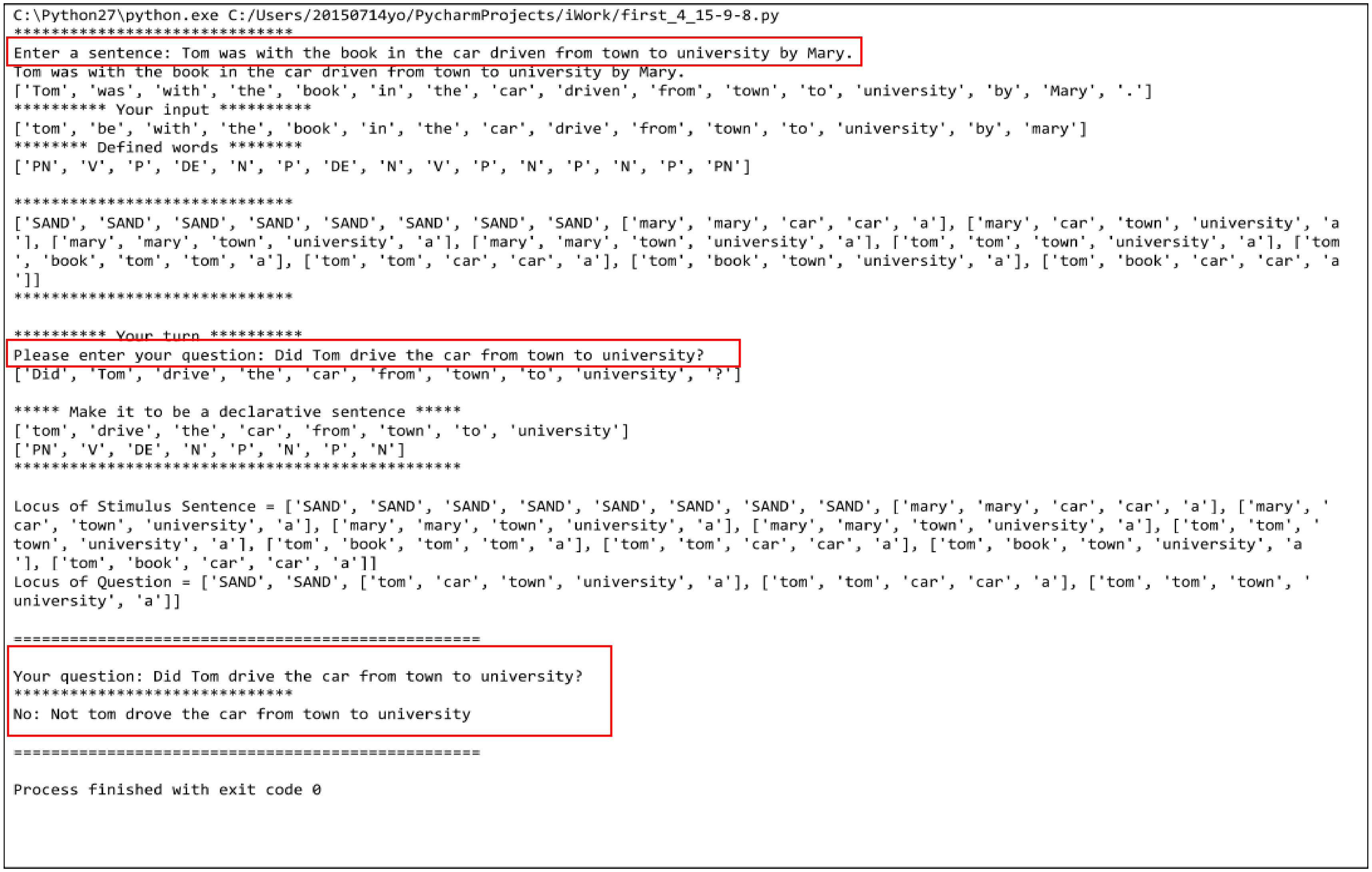


```
C:\Python27\python.exe C:/Users/20150714yo/PycharmProjects/iWork/first_4_15-9-8.py
******************************
Enter a sentence: Tom was with the book in the bus running from town to university.
Tom was with the book in the bus running from town to university.
['Tom', 'was', 'with', 'the', 'book', 'in', 'the', 'bus', 'running', 'from', 'town', 'to', 'university', '.']
********** Your input **********
['tom', 'be', 'with', 'the', 'book', 'in', 'the', 'bus', 'run', 'from', 'town', 'to', 'university']
******** Defined words ********
['PN', 'V', 'P', 'DE', 'N', 'P', 'DE', 'N', 'V', 'P', 'N', 'P', 'N']

******************************
['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['bus', 'bus', 'town', 'university', 'a'], ['bus', 'bus', 'town', 'university
', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'bus', 'bus', 'a'], ['tom', 'bus', 'town
', 'university', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'bus', 'bus', 'a'], ['tom', 'tom', 'town', 'university', 'a
']]
******************************

********** Your turn **********
Please enter your question: What was moving from town to university?
['What', 'was', 'moving', 'from', 'town', 'to', 'university', '?']

***** Make it to be a declarative sentence *****
['what', 'move', 'from', 'town', 'to', 'university']
['PN', 'V', 'P', 'N', 'P', 'N']
************************************************

Locus of Stimulus Sentence = ['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['bus', 'bus', 'town', 'university', 'a'], ['bus
', 'bus', 'town', 'university', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'bus', 'bus
', 'a'], ['tom', 'bus', 'town', 'university', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'bus', 'bus', 'a'], ['tom', '
tom', 'town', 'university', 'a']]
Locus of Question = [['what', 'what', 'town', 'university', 'a']]

==================================================

Your question: What was moving from town to university?
******************************
The answer is: bus
The answer is: book
The answer is: tom

==================================================

Process finished with exit code 0
```

Fig 5.6 3rd example of MBU processing for stimulus sentence Type I.

Stimulus sentence: Tom was with the book in the car driven from town to university by Mary.

Question: Did Tom drive the car from town to university?

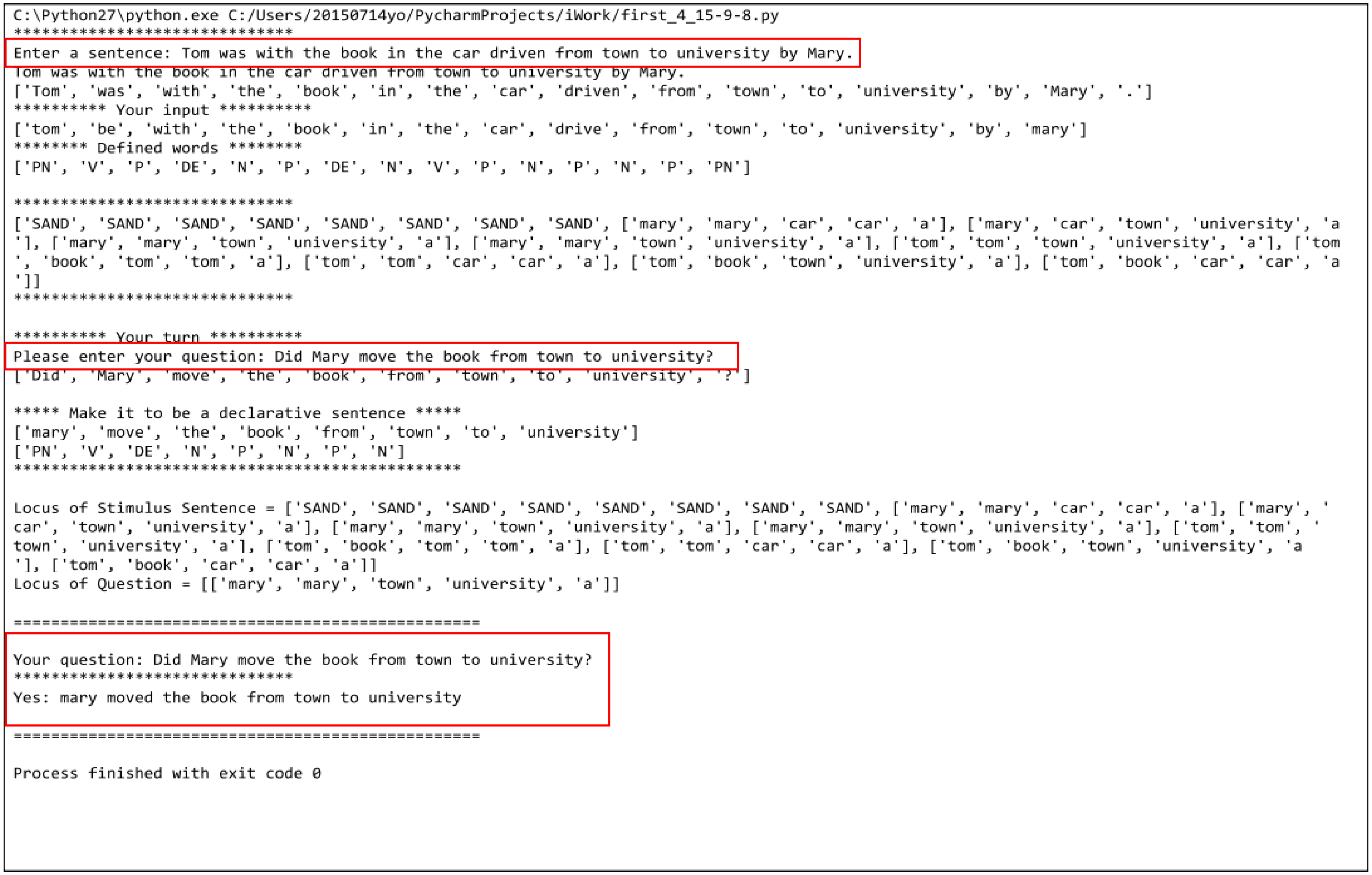


```
C:\Python27\python.exe C:/Users/20150714yo/PycharmProjects/iWork/first_4_15-9-8.py
******************************
Enter a sentence: Tom was with the book in the car driven from town to university by Mary.
Tom was with the book in the car driven from town to university by Mary.
['Tom', 'was', 'with', 'the', 'book', 'in', 'the', 'car', 'driven', 'from', 'town', 'to', 'university', 'by', 'Mary', '.']
********** Your input **********
['tom', 'be', 'with', 'the', 'book', 'in', 'the', 'car', 'drive', 'from', 'town', 'to', 'university', 'by', 'mary']
******** Defined words ********
['PN', 'V', 'P', 'DE', 'N', 'P', 'DE', 'N', 'V', 'P', 'N', 'P', 'N', 'P', 'PN']

******************************
['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['mary', 'mary', 'car', 'car', 'a'], ['mary', 'car', 'town', 'university', 'a
'], ['mary', 'mary', 'town', 'university', 'a'], ['mary', 'mary', 'town', 'university', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom
', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'car', 'car', 'a
']]
******************************

********** Your turn **********
Please enter your question: Did Tom drive the car from town to university?
['Did', 'Tom', 'drive', 'the', 'car', 'from', 'town', 'to', 'university', '?']

***** Make it to be a declarative sentence *****
['tom', 'drive', 'the', 'car', 'from', 'town', 'to', 'university']
['PN', 'V', 'DE', 'N', 'P', 'N', 'P', 'N']
************************************************

Locus of Stimulus Sentence = ['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['mary', 'mary', 'car', 'car', 'a'], ['mary', '
car', 'town', 'university', 'a'], ['mary', 'mary', 'town', 'university', 'a'], ['mary', 'mary', 'town', 'university', 'a'], ['tom', 'tom', '
town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'book', 'town', 'university', 'a
'], ['tom', 'book', 'car', 'car', 'a']]
Locus of Question = ['SAND', 'SAND', ['tom', 'car', 'town', 'university', 'a'], ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'tom', 'town', '
university', 'a']]

==================================================

Your question: Did Tom drive the car from town to university?
******************************
No: Not tom drove the car from town to university

==================================================

Process finished with exit code 0
```

Fig 5.7 1st example of MBU processing for stimulus sentence Type II.

Stimulus sentence: Tom was with the book in the car driven from town to university by Mary.

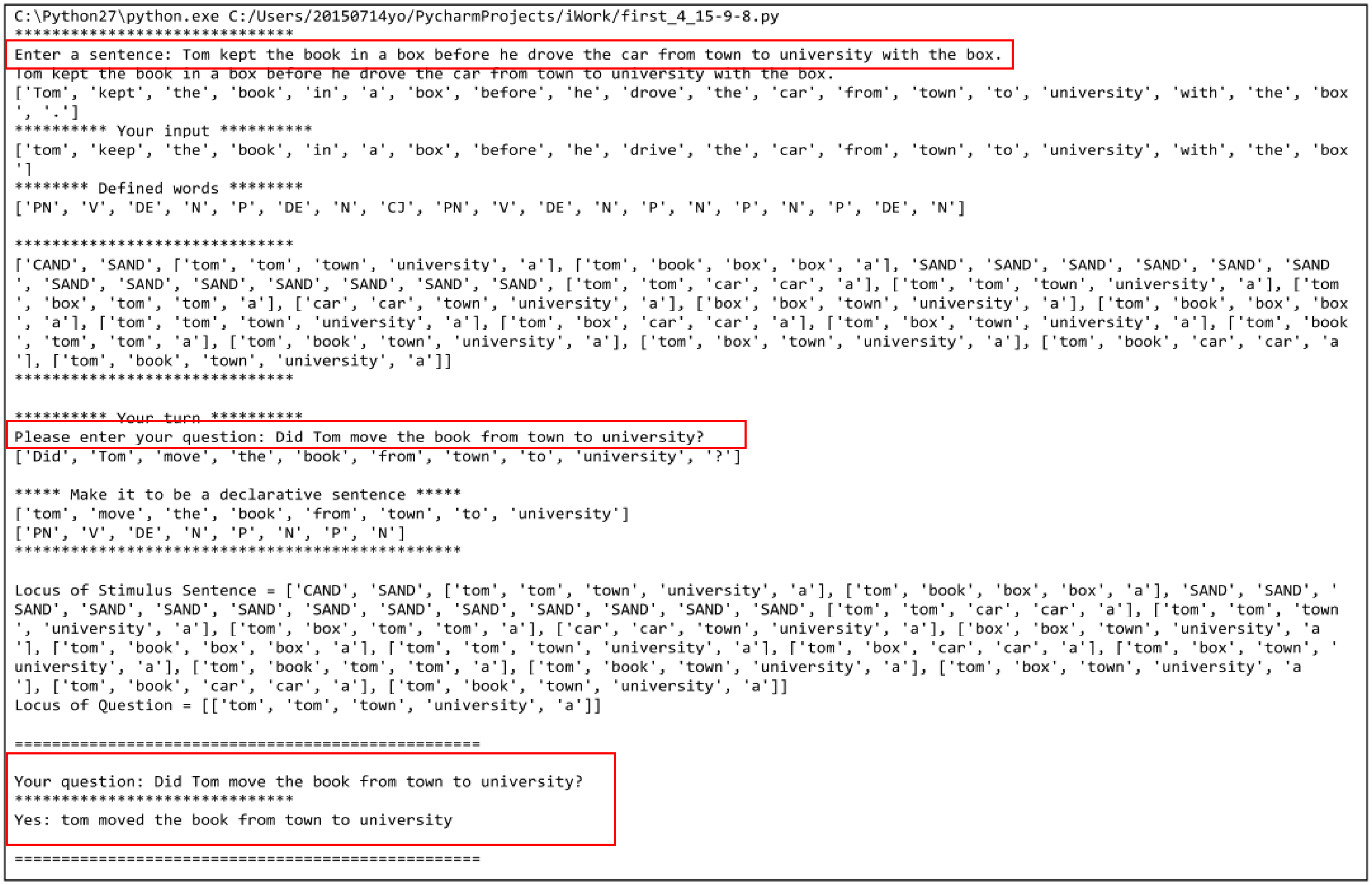
Question: Did Mary move the book from town to university?



```
C:\Python27\python.exe C:/Users/20150714yo/PycharmProjects/iWork/first_4_15-9-8.py
******************************
Enter a sentence: Tom was with the book in the car driven from town to university by Mary.
Tom was with the book in the car driven from town to university by Mary.
['Tom', 'was', 'with', 'the', 'book', 'in', 'the', 'car', 'driven', 'from', 'town', 'to', 'university', 'by', 'Mary', '.']
********** Your input **********
['tom', 'be', 'with', 'the', 'book', 'in', 'the', 'car', 'drive', 'from', 'town', 'to', 'university', 'by', 'mary']
******** Defined words ********
['PN', 'V', 'P', 'DE', 'N', 'P', 'DE', 'N', 'V', 'P', 'N', 'P', 'N', 'P', 'PN']

******************************
['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['mary', 'mary', 'car', 'car', 'a'], ['mary', 'car', 'town', 'university', 'a
'], ['mary', 'mary', 'town', 'university', 'a'], ['mary', 'mary', 'town', 'university', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom
', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'book', 'car', 'car', 'a
']]
******************************

********** Your turn **********
Please enter your question: Did Mary move the book from town to university?
['Did', 'Mary', 'move', 'the', 'book', 'from', 'town', 'to', 'university', '?']

***** Make it to be a declarative sentence *****
['mary', 'move', 'the', 'book', 'from', 'town', 'to', 'university']
['PN', 'V', 'DE', 'N', 'P', 'N', 'P', 'N']
**************************************************

Locus of Stimulus Sentence = ['SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['mary', 'mary', 'car', 'car', 'a'], ['mary', '
car', 'town', 'university', 'a'], ['mary', 'mary', 'town', 'university', 'a'], ['mary', 'mary', 'town', 'university', 'a'], ['tom', 'tom', '
town', 'university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'book', 'town', 'university', 'a
'], ['tom', 'book', 'car', 'car', 'a']]
Locus of Question = [['mary', 'mary', 'town', 'university', 'a']]

==================================================

Your question: Did Mary move the book from town to university?
******************************
Yes: mary moved the book from town to university

==================================================

Process finished with exit code 0
```

Fig 5.8 2nd example of MBU processing for stimulus sentence Type II.

Stimulus sentence:	Tom kept the book in a box before he drove the car from town to university with the box.
Question:	Did Tom move the book from town to university?

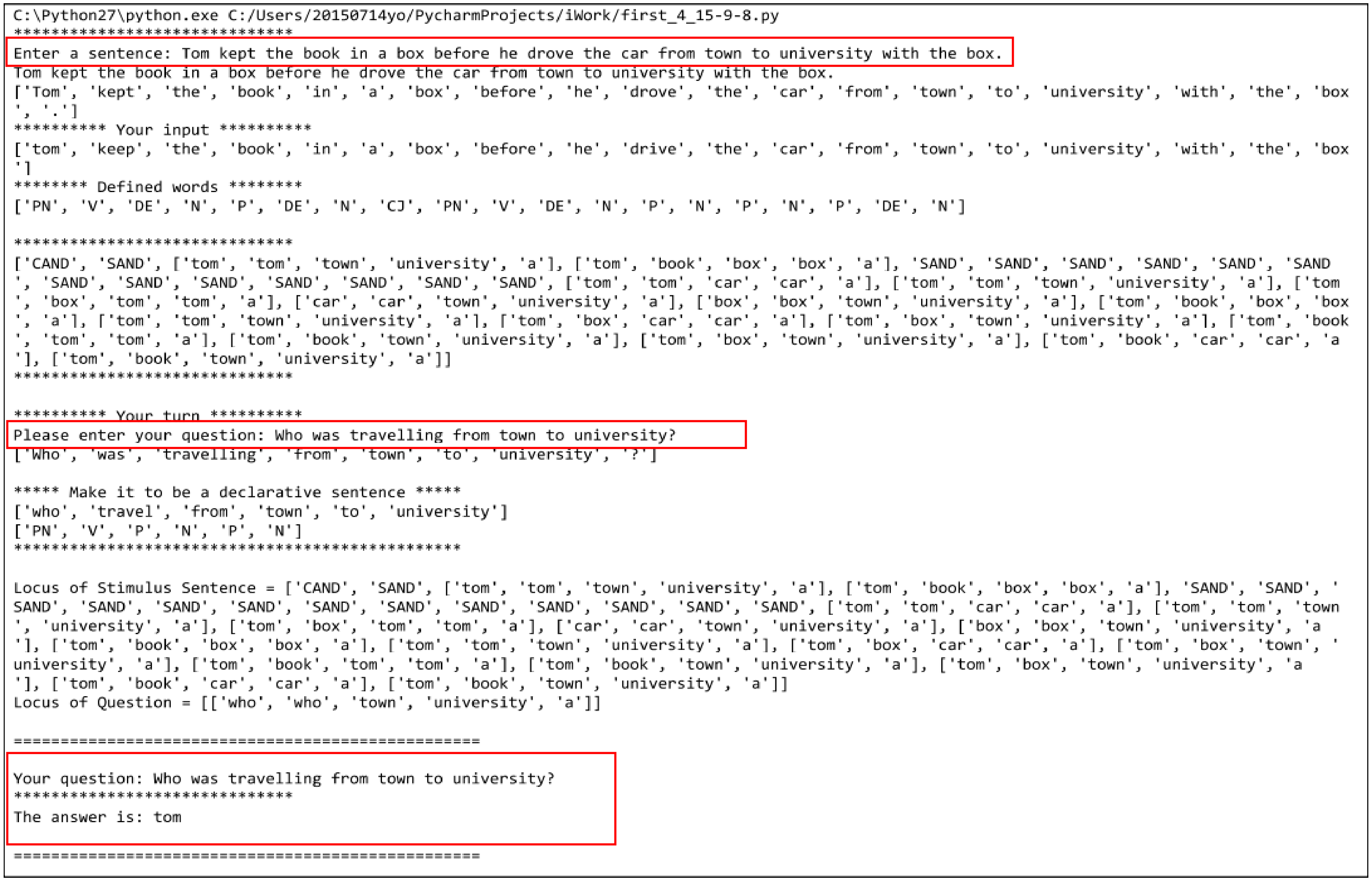


```
C:\Python27\python.exe C:/Users/20150714yo/PycharmProjects/iWork/first_4_15-9-8.py
******************************
Enter a sentence: Tom kept the book in a box before he drove the car from town to university with the box.
Tom kept the book in a box before he drove the car from town to university with the box.
['Tom', 'kept', 'the', 'book', 'in', 'a', 'box', 'before', 'he', 'drove', 'the', 'car', 'from', 'town', 'to', 'university', 'with', 'the', 'box
', '.']
********** Your input **********
['tom', 'keep', 'the', 'book', 'in', 'a', 'box', 'before', 'he', 'drive', 'the', 'car', 'from', 'town', 'to', 'university', 'with', 'the', 'box
']
******** Defined words ********
['PN', 'V', 'DE', 'N', 'P', 'DE', 'N', 'CJ', 'PN', 'V', 'DE', 'N', 'P', 'N', 'P', 'N', 'P', 'DE', 'N']

******************************
['CAND', 'SAND', ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'box', 'box', 'a'], 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND
', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom
', 'box', 'tom', 'tom', 'a'], ['car', 'car', 'town', 'university', 'a'], ['box', 'box', 'town', 'university', 'a'], ['tom', 'book', 'box', 'box
', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'box', 'car', 'car', 'a'], ['tom', 'box', 'town', 'university', 'a'], ['tom', 'book
', 'tom', 'tom', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'box', 'town', 'university', 'a'], ['tom', 'book', 'car', 'car', 'a
'], ['tom', 'book', 'town', 'university', 'a']]
******************************

********** Your turn **********
Please enter your question: Did Tom move the book from town to university?
['Did', 'Tom', 'move', 'the', 'book', 'from', 'town', 'to', 'university', '?']

***** Make it to be a declarative sentence *****
['tom', 'move', 'the', 'book', 'from', 'town', 'to', 'university']
['PN', 'V', 'DE', 'N', 'P', 'N', 'P', 'N']
************************************************

Locus of Stimulus Sentence = ['CAND', 'SAND', ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'box', 'box', 'a'], 'SAND', 'SAND', '
SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'tom', 'town
', 'university', 'a'], ['tom', 'box', 'tom', 'tom', 'a'], ['car', 'car', 'town', 'university', 'a'], ['box', 'box', 'town', 'university', 'a
'], ['tom', 'book', 'box', 'box', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'box', 'car', 'car', 'a'], ['tom', 'box', 'town', '
university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'box', 'town', 'university', 'a
'], ['tom', 'book', 'car', 'car', 'a'], ['tom', 'book', 'town', 'university', 'a']]
Locus of Question = [['tom', 'tom', 'town', 'university', 'a']]

==================================================

Your question: Did Tom move the book from town to university?
******************************
Yes: tom moved the book from town to university

==================================================
```

Fig 5.9 1st example of MBU processing for stimulus sentence Type III.

Stimulus sentence: Tom kept the book in a box before he drove the car from town to university with the box.

Question: Who was travelling from town to university?



```
C:\Python27\python.exe C:/Users/20150714yo/PycharmProjects/iWork/first_4_15-9-8.py
******************************
Enter a sentence: Tom kept the book in a box before he drove the car from town to university with the box.
Tom kept the book in a box before he drove the car from town to university with the box.
['Tom', 'kept', 'the', 'book', 'in', 'a', 'box', 'before', 'he', 'drove', 'the', 'car', 'from', 'town', 'to', 'university', 'with', 'the', 'box
', '.']
********** Your input **********
['tom', 'keep', 'the', 'book', 'in', 'a', 'box', 'before', 'he', 'drive', 'the', 'car', 'from', 'town', 'to', 'university', 'with', 'the', 'box
']
******** Defined words ********
['PN', 'V', 'DE', 'N', 'P', 'DE', 'N', 'CJ', 'PN', 'V', 'DE', 'N', 'P', 'N', 'P', 'N', 'P', 'DE', 'N']

******************************
['CAND', 'SAND', ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'box', 'box', 'a'], 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND
', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom
', 'box', 'tom', 'tom', 'a'], ['car', 'car', 'town', 'university', 'a'], ['box', 'box', 'town', 'university', 'a'], ['tom', 'book', 'box', 'box
', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'box', 'car', 'car', 'a'], ['tom', 'box', 'town', 'university', 'a'], ['tom', 'book
', 'tom', 'tom', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'box', 'town', 'university', 'a'], ['tom', 'book', 'car', 'car', 'a
'], ['tom', 'book', 'town', 'university', 'a']]
******************************

********** Your turn **********
Please enter your question: Who was travelling from town to university?
['Who', 'was', 'travelling', 'from', 'town', 'to', 'university', '?']

***** Make it to be a declarative sentence *****
['who', 'travel', 'from', 'town', 'to', 'university']
['PN', 'V', 'P', 'N', 'P', 'N']
************************************************

Locus of Stimulus Sentence = ['CAND', 'SAND', ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'book', 'box', 'box', 'a'], 'SAND', 'SAND', '
SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', 'SAND', ['tom', 'tom', 'car', 'car', 'a'], ['tom', 'tom', 'town
', 'university', 'a'], ['tom', 'box', 'tom', 'tom', 'a'], ['car', 'car', 'town', 'university', 'a'], ['box', 'box', 'town', 'university', 'a
'], ['tom', 'book', 'box', 'box', 'a'], ['tom', 'tom', 'town', 'university', 'a'], ['tom', 'box', 'car', 'car', 'a'], ['tom', 'box', 'town', '
university', 'a'], ['tom', 'book', 'tom', 'tom', 'a'], ['tom', 'book', 'town', 'university', 'a'], ['tom', 'box', 'town', 'university', 'a
'], ['tom', 'book', 'car', 'car', 'a'], ['tom', 'book', 'town', 'university', 'a']]
Locus of Question = [['who', 'who', 'town', 'university', 'a']]

==================================================

Your question: Who was travelling from town to university?
******************************
The answer is: tom

==================================================
```

Fig 5.10 2nd example of MBU processing for stimulus sentence Type III.

5.3 Conversation Management System

We have been developing a conversation management system (CMS) in Python for simulating human-robot interaction in NL [5.1], [5.2], and [5.3]. The implemented AI named *Anna* understands NL utterances in text by a person and each person as her dialogue partner is to be played as by the system user named *Taro.* Anna is assumed as a lady-shaped helper robot for Taro, a physically handicapped elderly man.

On imagination, a robot named *Anna* lives in a house with an elderly man named *Taro*. Taro cannot walk by himself and must use a wheelchair to move. Anna is a lady-shaped robot to help him in every scene of his daily life, namely, physically, mentally and socially. She is just a new comer to Taro's home and therefore she must explore there by deploying her intrinsic sensors, actuators and brain.

Fig. 5.11 Anna and Taro.

This time, Taro and his assistance, Anna, are shown in Figs. 5.11 and Fig. 5.12 to comprehend Taro's intention through dialogue and her final response to his intention is animated graphically. When Anna finds any problem in every situation in helping him, she tries to solve the problem by reasoning based on her knowledge and the information acquired by inquiry to the people who are Taro and the other residents in the town shown in Fig. 5.13 – 5.14.

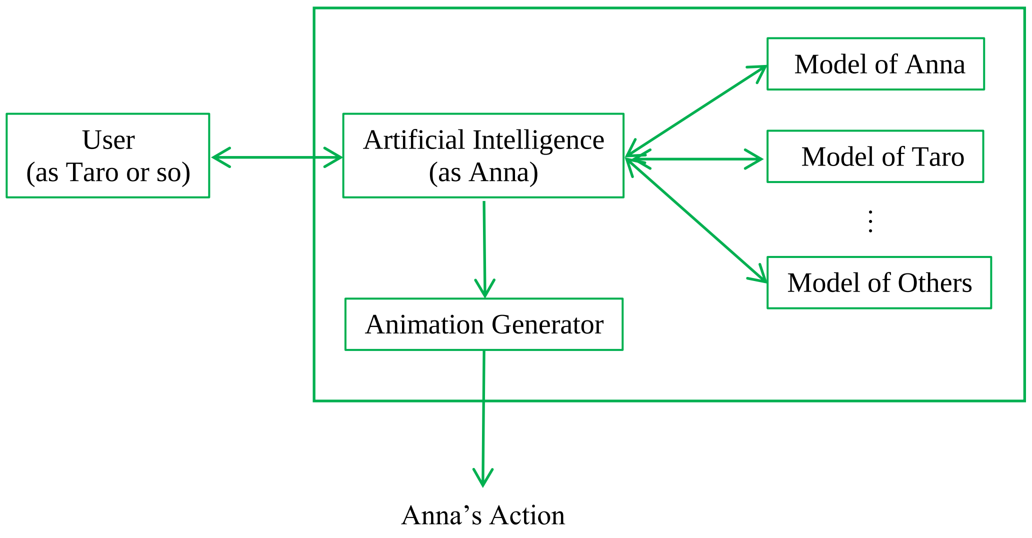


Fig 5.12 Conversation management system modules.

Anna as an AI understands NL utterances in text by a person and each person as her dialogue partner is to be played as by the system user. The major modules of CMS as shown in Fig. 5.12 are roughly defined as follows while it is still evolving.

- (M1) Natural language understanding module (inside AI)

Interprets an input text into ***L**$_{md}$* expressions and selects the most plausible interpretation by employing all the kinds of knowledge, especially, word-meaning definitions intrinsic to this module.

- (M2) Thing-specific model

Consists of knowledge about each person such as his/her belief, physical mobility, mental tendency, and social activity, or about each non-personal thing (e.g. restaurant, flower bed, and apartment) such as its function. Every knowledge piece is represented in ***L**$_{md}$* in the same way as in section 3.3.

- (M3) Problem finding/solving module (inside AI)

Finds event gaps in ***L**$_{md}$* expressions and cancel them by employing commonsense knowledge pieces such as postulate of Continuity in attribute values (P_{CAV}) and the thing-specific models. When problem solving is successful, the solution is sent to Animation generator in the form of ***L**$_{md}$* expression. Otherwise, the dialogue partner or so is asked for further information.

- (M4) Animation generator

Animates the solution in ***L**$_{md}$* sent from the problem solver.

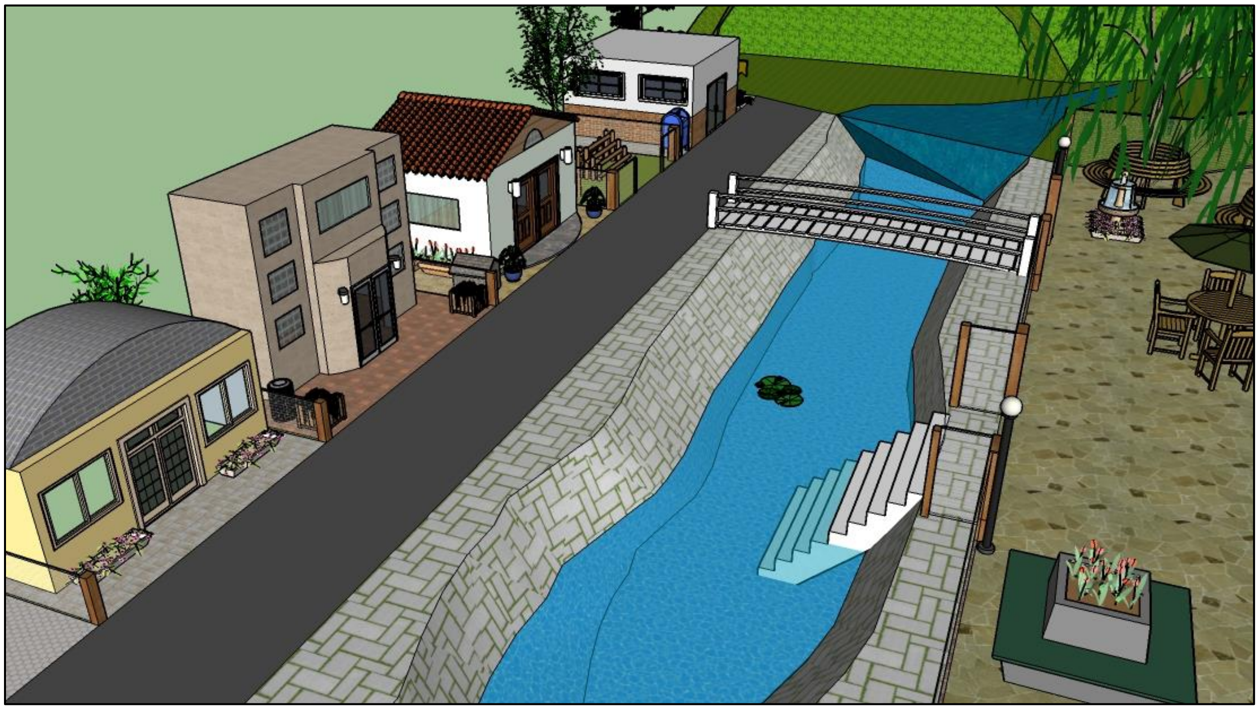


Fig 5.13 Town map in CMS.

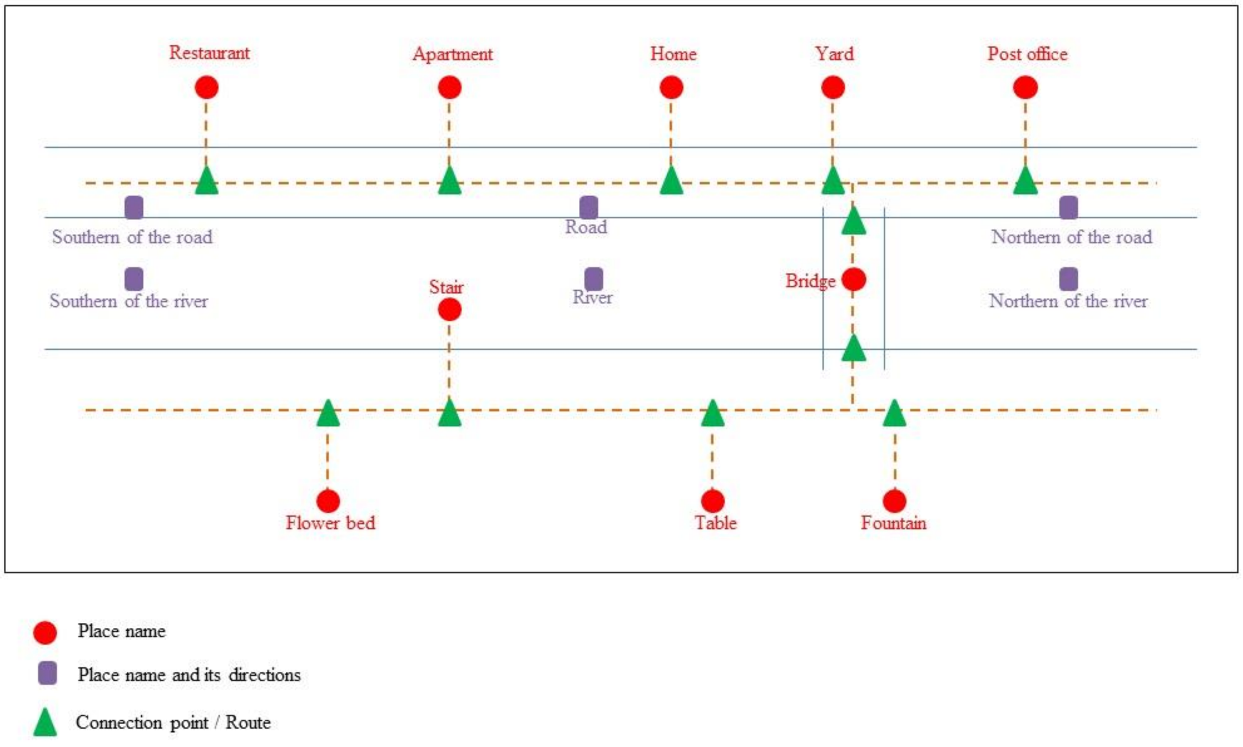


Fig 5.14 Graph data of Town map in CMS.

This section details CMS based on a typical example of dialogue between Taro and Anna. Basically, Anna tries to understand Taro's utterance as a certain request, namely, *intention* for her to do something because her mission, one of Anna's beliefs, is to help him in every aspect. This time, for the sake of simplicity, Taro's utterances were limited to what should represent his intentions explicitly without any rhetorical sophistication, for example, by employing such expressions as "I want…", imperatives, question-word questions and yes - no questions.

Under this assumption, Taro's request is logically formulated as eq. 5.26. This formula reads "If D, then Taro will get happier (by something z, possibly, D itself)", where $H-ness = (B_{031}, G_t, k)$ representing temporal change in *Happiness*, one of subspaces of the attribute *Emotion (B_{03})*, and the pair of values p and $p<$ implies certain increase in happiness. Hereafter, such a relation pRq between two values p and q is to be represented simply by the pair of values p and pR, where R is such as $<$ and $\neq$.

$$D \rightarrow L(z, Taro, p, p<, H-ness) \qquad \text{(eq. 5.26)}$$

The aim of Anna's mission is the right hand of eq. 5.26, namely, Taro's getting happier, and she believes that if she realizes D then Taro will get happier. Therefore, she tries to realize D, where the general algorithm implemented can be digested as the steps below.

- (STEP 1) Evaluate D by the thing-specific models.
- (STEP 2) Remove any part of D that is already feasible as it is, namely, of no problem to be solved. The remaining part is the goal event (X_G) to be attained by Anna.
- (STEP 3) Find the current event ($= X_C$) and compose $X_C \bullet X_T \bullet X_G$ as defined in eq. 4.1.
- (STEP 4) Assign constants to variables appearing in X_C or X_G by asking User (= Taro, here) or consulting the thing-specific models.
- (STEP 5) Solve X_T according to the postulate of *continuity in attribute values* as explained by eq. 4.2 and eq. 4.3.
- (STEP 6) Align and revise X_T to be feasible (in animation) by consulting the models concerned.
- (STEP 7) If another event gap is detected in association with the revised X_T, find another X_C to compose another $X_C \bullet X_T \bullet X_G$ and return to STEP 4. Otherwise, send the solution to Animation Generator.

For example, consider such a situation as follows.

Anna is at Taro. Taro remembers to call his friend Tom and Anna knows that he uses his cellular phone for calling someone.

In this situation, the dialogues between them and Anna's conversation management involved and the performances of CMS are as follows.

Taro: "I want to call Tom."

Anna's understanding of this utterance is given by eq. 5.27.

$$\begin{gathered} L(Taro, Phone, Taro, Taro, \Lambda_t) \\ \underline{\Pi L(Taro, Voice, Taro, Tom, \Lambda_t)} \\ \rightarrow L(z, Taro, p, p <, H - ness) \end{gathered} \qquad \text{(eq. 5.27)}$$

(D = "Taro holds the phone, sending voice to Tom") The meaning of "$x\ call\ y$" is defined as eq. 5.28.

$$\begin{gathered} x\ call\ y \Leftrightarrow L(x, Phone, x, x, \Lambda_t) \\ \Pi L(x, Voice, x, y, \Lambda_t) \end{gathered} \qquad \text{(eq. 5.28)}$$

At STEP 1, Anna infers eq. 5.29 from Taro's model.

$$\begin{gathered} Taro \approx> \{+L(Taro, Voice, Taro, Taro \neq, \Lambda_t), \\ +L(Taro, Phone, p, p, Loc)\} \end{gathered} \qquad \text{(eq. 5.29)}$$

This is about Taro's probability directly associated with D in eq. 5.27, implying "Taro can send voice out" by the pair of values $Taro$ and $Taro\neq$, and "Taro can keep the phone somewhere (p)".

Then, at STEP 2, she finds no problem in Taro's sending voice out, the underlined event in eq. 5.27, and gets aware that she has only to make <u>Taro hold the phone,</u> namely, the italicized part of eq. 5.27.

At STEP 3, she does not know where ($= p$) Taro keeps the phone. Therefore, her problem ($= X_T$) is to know <u>its current place ($= X_T$) and make it move to him</u>. The event concerning the phone's location is formalized by eq. 5.30 and eq. 5.31. If Taro has the phone with him currently and Anna knows that (namely, already included in X_C), there is no event gap.

$$\underline{L(z_1, Phone, ?p, ?p, \Lambda_t) \bullet X_T}$$
$$\bullet L(Taro, Phone, Taro, Taro, \Lambda_t) \tag{eq. 5.30}$$

$$X_T = L(z_2, Phone, ?p, Taro, \Lambda_t) \tag{eq. 5.31}$$

At STEP 4, Anna asks Taro about the place of the phone.

Anna: "Where is the phone?"

This is the verbalization of the underlined part of eq. 5.30 for problem solving. That is, the semantic definition of "$where\ be\ x$?" is "$L(z_1, x, ?p, ?p, \Lambda_t)$".

Taro: "On the bed."

Anna understands that $?p = Bed$.

At STEP 5, she tries to solve the underlined part ($= X_T$) of eq. 5.32. Here, for the sake of simplicity, it is deemed that "$x\ on\ y$" $=$ "$x\ at\ y$" (c.f., eq. 3.17).

$$L(Taro, Phone, Bed, Bed, \Lambda_t)$$
$$\underline{\bullet L(z_2, Phone, Bed, Taro, \Lambda_t)}$$
$$\bullet L(Taro, Phone, Taro, Taro, \Lambda_t) \tag{eq. 5.32}$$

At STEP 6, Anna supposes that $z_2 = Anna$, namely, that she moves the phone to Taro by herself and knows that <u>when Anna moves something, she carries it by herself</u>. (She has no telekinesis to move anything remotely.) This fact is formalized as the postulate eq. 5.33 prepared in Anna's model. This is the elaboration process to make the $\boldsymbol{L_{md}}$ expression feasible by the actor, namely, Anna here.

$$L(Anna, x, p, p \neq, \Lambda_t) \leftarrow\rightarrow$$
$$L(Anna, x, Anna, Anna, \Lambda_t)$$
$$\Pi L(Anna, Anna, p, p \neq, \Lambda_t) \tag{eq. 5.33}$$

Applying this postulate to eq. 5.32, Anna can deduce eq. 5.34 whose right hand is the revised X_T to be the new goal event to be attained.

$$L(Anna, Phone, Bed, Taro, \Lambda_t) \leftarrow\rightarrow$$
$$L(Anna, Phone, Anna, Anna, \Lambda_t)$$
$$\Pi L(Anna, Anna, Bed, Taro, \Lambda_t) \tag{eq. 5.34}$$

At STEP 7, another event gap is found in the revised X_T as X_{T1} in eq. 5.35. Anna knows that <u>currently, Anna is at Taro</u> and this fact is appended to the previous X_C $(= L(Taro, Phone, Bed, Bed, \Lambda_t))$ as underlined in eq. 5.35, where X_{T1} is another transit event newly aroused in the previous one.

$$\begin{gathered}\underline{(L(Anna, Anna, Taro, Taro, \Lambda_t)} \\ \Pi L(Taro, Phone, Bed, Bed, \Lambda_t)) \bullet X_{T1} \\ \bullet (L(Anna, Phone, Anna, Anna, \Lambda_t) \\ \Pi L(Anna, Anna, Bed, Taro, \Lambda_t))\end{gathered} \qquad \text{(eq. 5.35)}$$

Again at STEP 3, the new transit event X_{T1} is solved as eq. 5.36.

$$\begin{gathered}X_{T1} = L(Anna, Anna, Taro, Bed, \Lambda_t) \\ \bullet L(Anna, Phone, Bed, Anna, \Lambda_t)\end{gathered} \qquad \text{(eq. 5.36)}$$

Then, the solution of X_T is finally given as eq. 5.37. This is the final form of the solution that Anna can make feasible by herself.

$$\begin{gathered}X_T = (L(Anna, Anna, Taro, Bed, \Lambda_t) \\ \bullet L(Anna, Phone, Bed, Anna, \Lambda_t)) \\ \bullet L(Anna, Phone, Anna, Anna, \Lambda_t) \\ \Pi L(Anna, Anna, Bed, Taro, \Lambda_t))\end{gathered} \qquad \text{(eq. 5.37)}$$

Anna: “Sure, I’ll go to the bed, then take the phone, and bring it back to you.”

This is the verbalization of eq. 5.37, where pattern matching operation is performed to search eq. 4.4 for verb concepts such as eq. 3.16 – eq. 3.24. The final solution is sent to Animation Generator as well to be translated into a procedure for animation.

For more understanding, please consider such a question as S29 to a certain NLU system from its human user. Then, this question can be read roughly as eq. 5.38, where ➔ means that the right hand is deduced from the left hand by applying some pieces of knowledge.

(S29) When Tom drives with Mary, does she move?

$$?\, drive(Tom)\ \&\ with(Tom, Mary) \Rightarrow move(Mary) \qquad \text{(eq. 5.38)}$$

For conventional NLU systems to answer such a question correctly, some special postulate like eq. 5.39 should be needed, where → means that if the left hand is true then the right hand is true, too.

$$drive(x)\,\&\,with(x,y) \rightarrow move(y) \qquad \text{(eq. 5.39)}$$

However, people can easily answer "yes" without eq. 5.39. How do they do that? They must employ their mental image evoked by their own experiences. Anna can imitate such a human thinking process. Her understanding of S29 is given by eq. 5.39 and depicted as Fig. 5.15, where $\Lambda_t = (A_{12}, G_t, k)$. Anna reasons about this case employing such postulates or inference rules that is described in section 5.1.

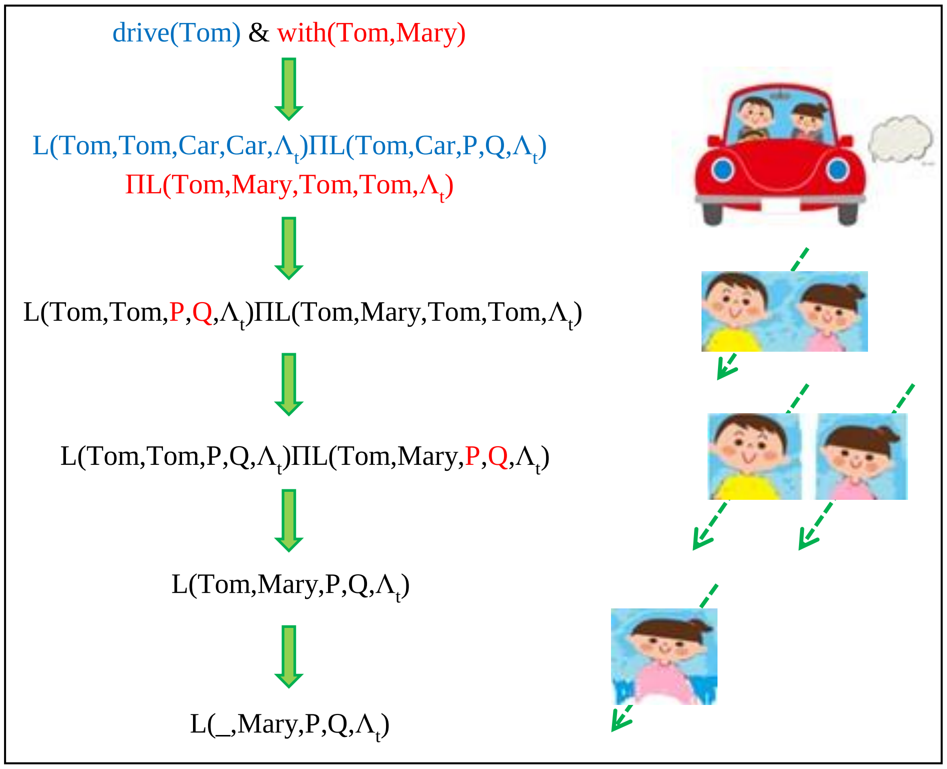
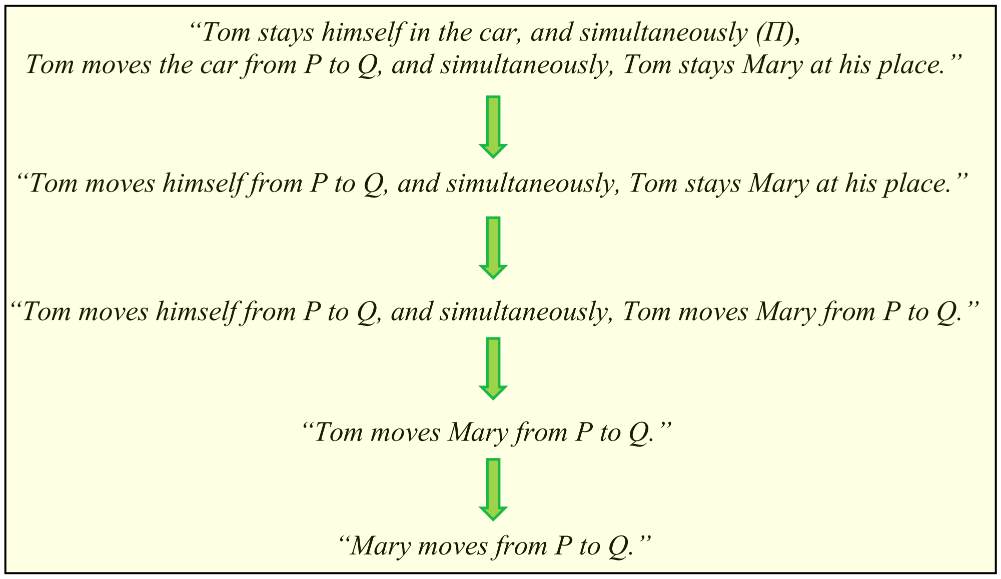

drive(Tom) & with(Tom,Mary)
L(Tom,Tom,Car,Car,Λt)ΠL(Tom,Car,P,Q,Λt)
ΠL(Tom,Mary,Tom,Tom,Λt)
L(Tom,Tom,P,Q,Λt)ΠL(Tom,Mary,Tom,Tom,Λt)
L(Tom,Tom,P,Q,Λt)ΠL(Tom,Mary,P,Q,Λt)
L(Tom,Mary,P,Q,Λt)
L(_,Mary,P,Q,Λt)

Fig 5.15 Anna's thinking process for S29.

The reasoning process in Fig. 5.15 can be translated as following figure (Fig. 5.16).



"Tom stays himself in the car, and simultaneously (Π),
Tom moves the car from P to Q, and simultaneously, Tom stays Mary at his place."

$\Downarrow$

"Tom moves himself from P to Q, and simultaneously, Tom stays Mary at his place."

$\Downarrow$

"Tom moves himself from P to Q, and simultaneously, Tom moves Mary from P to Q."

$\Downarrow$

"Tom moves Mary from P to Q."

$\Downarrow$

"Mary moves from P to Q."

Fig 5.16 Translation of Anna's thinking process for S29.

5.4 Application of CMS system

As described in section 5.3, this part will demonstrate some examples of CMS. In the system, our two main characters, Anna and Taro, live at Taro's house (see Fig. 5.13 and Fig. 5.14 for more understanding about town map), while Taro's friends (i.e. Tom and Mary) are in their apartment. Therefore, when user (or Taro) utter about Tom and Mary, some scenery will swap to Tom and Mary instead.

The following figures are results from CMS that show time consuming in each process as well.

The first screen when user runs the program.

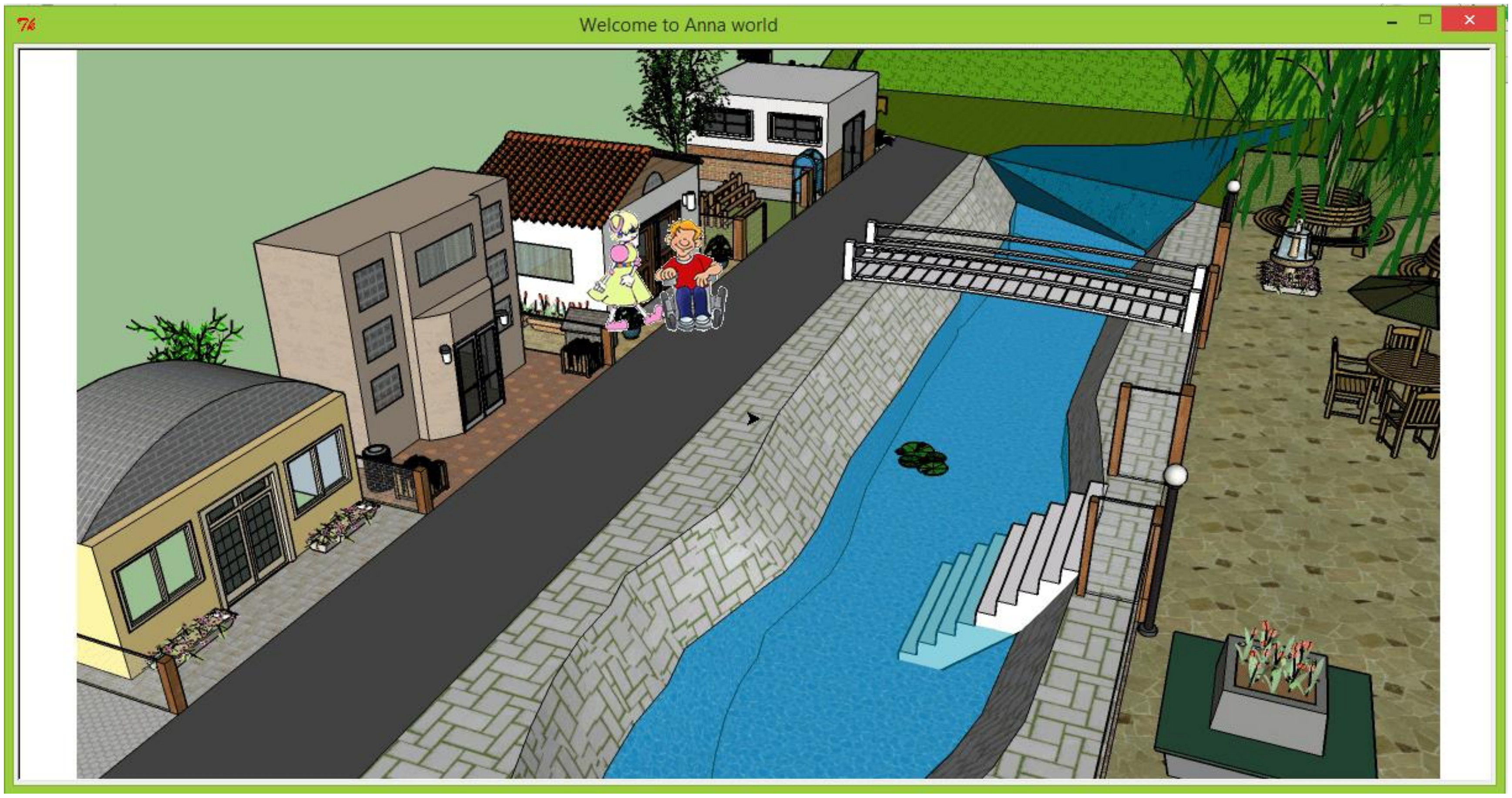
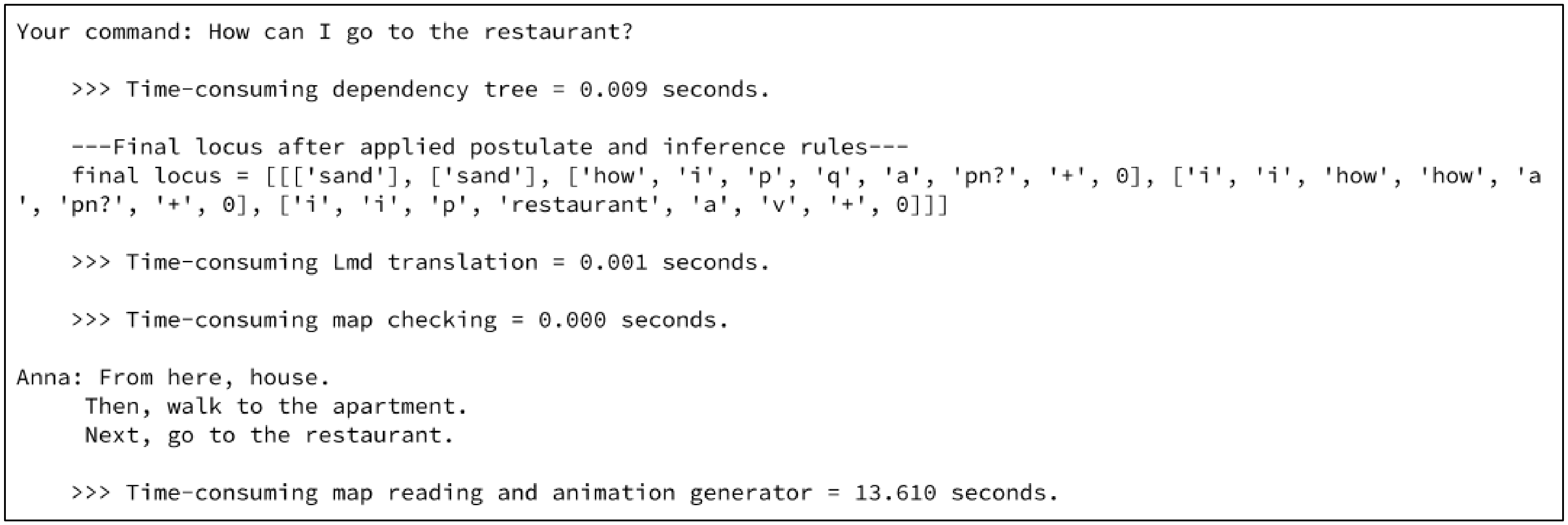


Fig. 5.17 Initial screen after the system is executed.

Input sentence (as Taro): “How can I go to the restaurant?”

Output from the system: “From here, house. Next go to the apartment. Then, walk to the restaurant.” (As shown in Fig 5.18)



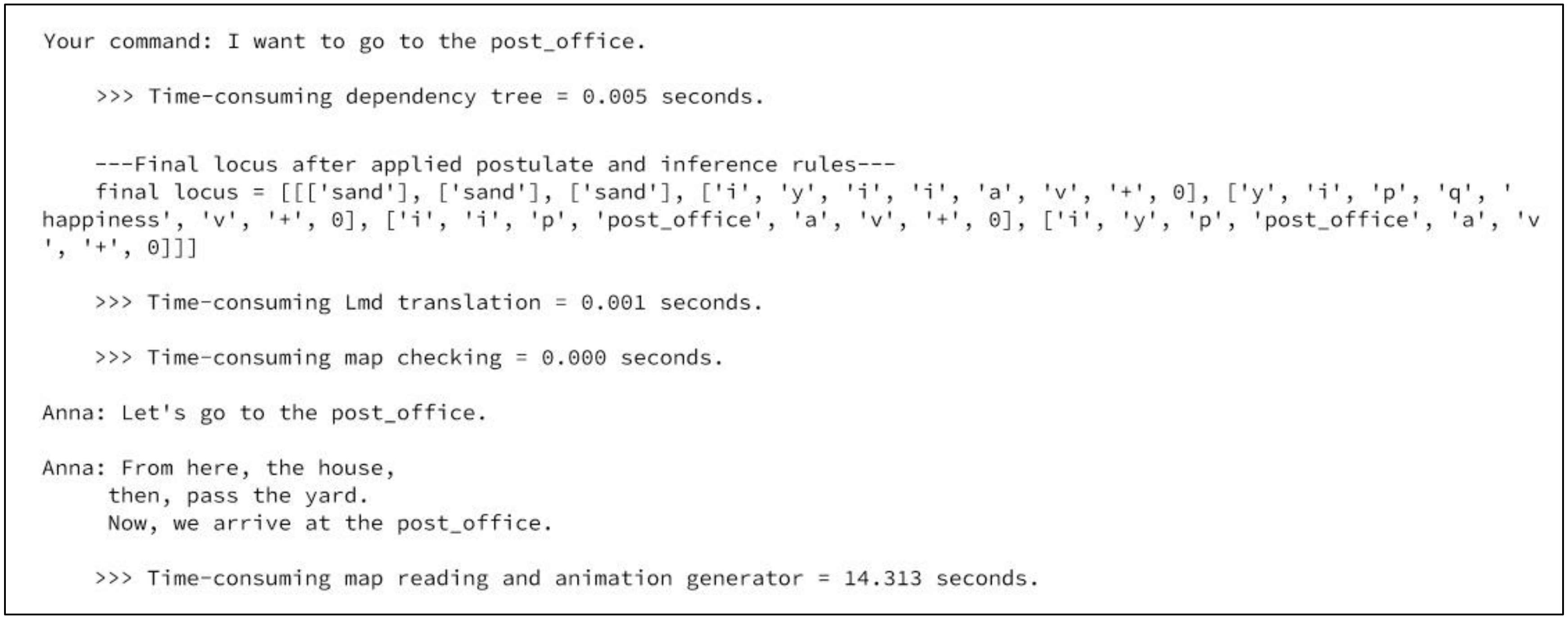
```
Your command: How can I go to the restaurant?

    >>> Time-consuming dependency tree = 0.009 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand'], ['sand'], ['how', 'i', 'p', 'q', 'a', 'pn?', '+', 0], ['i', 'i', 'how', 'how', 'a
', 'pn?', '+', 0], ['i', 'i', 'p', 'restaurant', 'a', 'v', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: From here, house.
     Then, walk to the apartment.
     Next, go to the restaurant.

    >>> Time-consuming map reading and animation generator = 13.610 seconds.
```

Fig. 5.18 Conversation screen when input is “How can I go to the restaurant?”.

Input sentence (as Taro): “I want to go to the post office.”

Output from the system: “Let’s go to the post office. From here, house. Then, pass the yard. Now, we arrive at the post office.” (As shown in Fig 5.19)

While Fig. 5.20 shows the changement of its animation screen.



```
Your command: I want to go to the post_office.

    >>> Time-consuming dependency tree = 0.005 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand'], ['sand'], ['sand'], ['i', 'y', 'i', 'i', 'a', 'v', '+', 0], ['y', 'i', 'p', 'q', '
happiness', 'v', '+', 0], ['i', 'i', 'p', 'post_office', 'a', 'v', '+', 0], ['i', 'y', 'p', 'post_office', 'a', 'v
', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: Let's go to the post_office.

Anna: From here, the house,
     then, pass the yard.
     Now, we arrive at the post_office.

    >>> Time-consuming map reading and animation generator = 14.313 seconds.
```

Fig. 5.19 Conversation screen when input is “I want to go to the post office”.

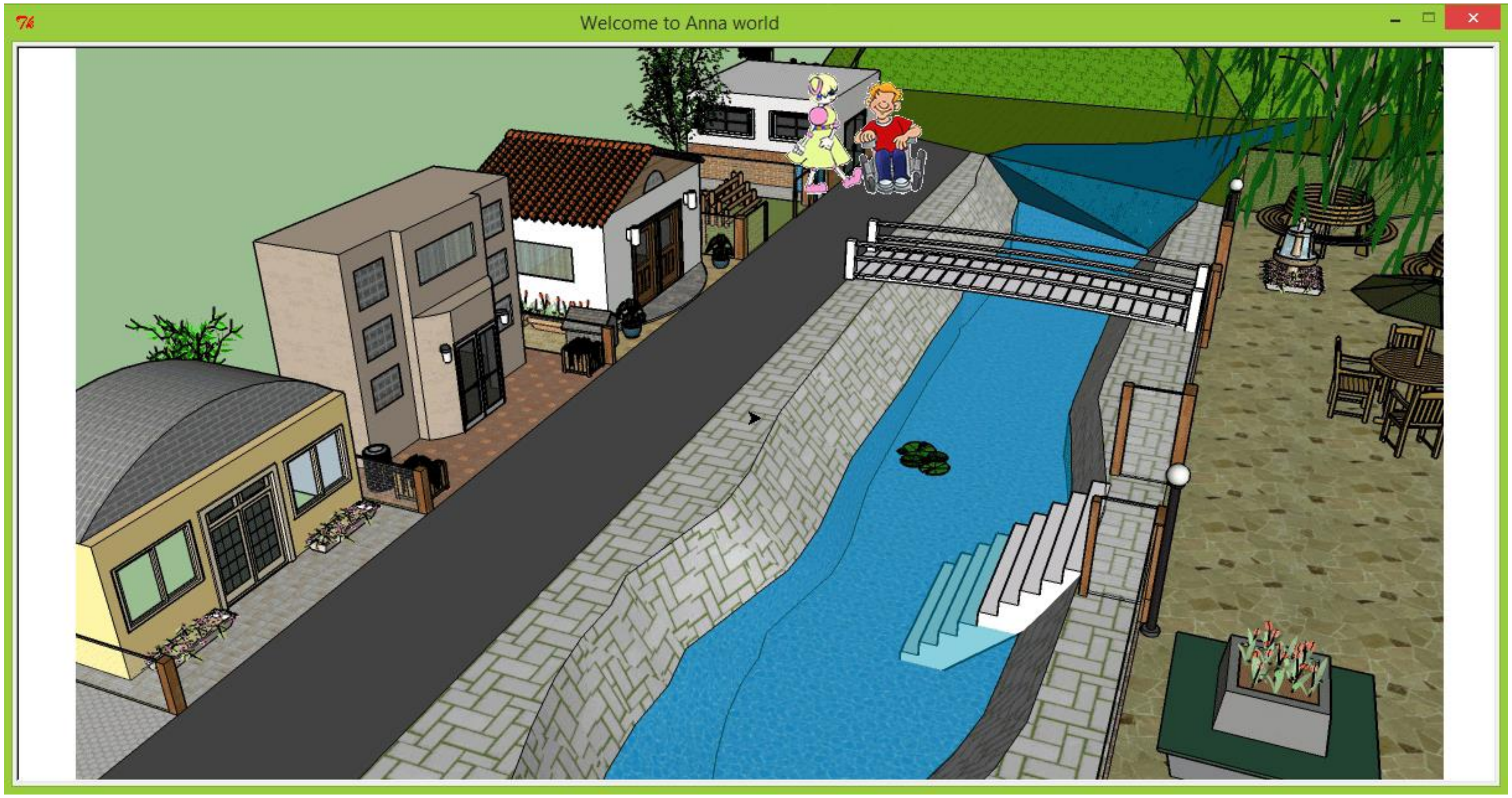
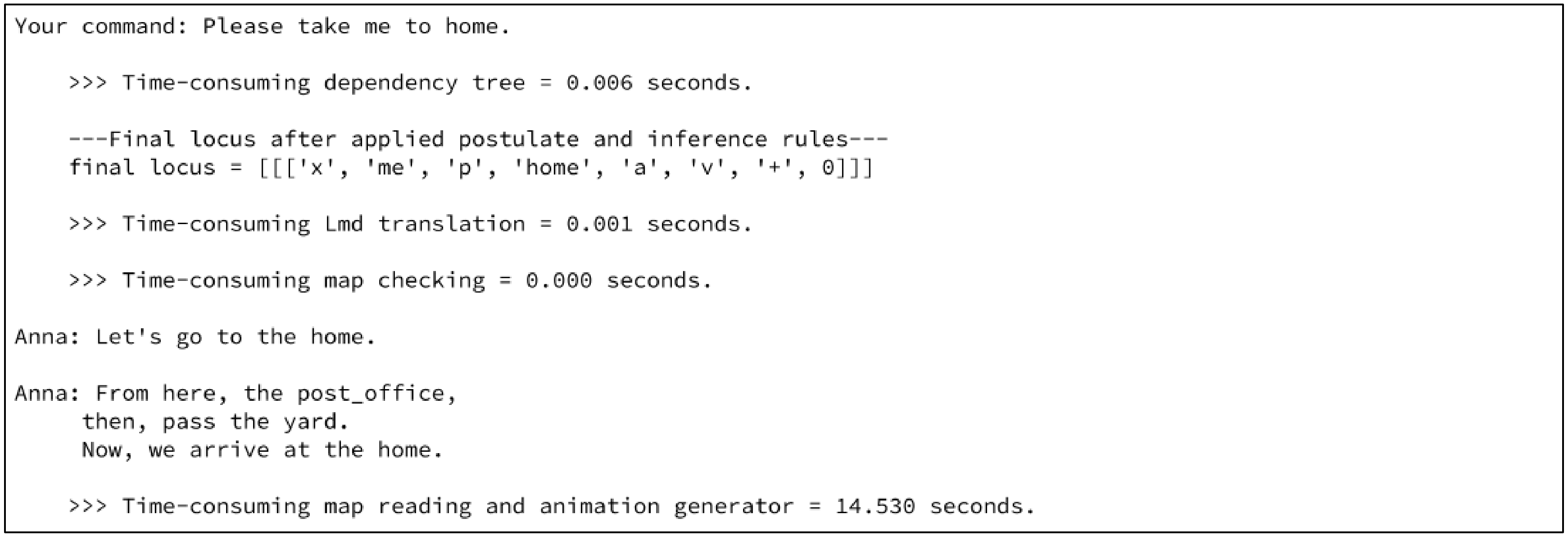


Fig. 5.20 Changement of animation screen that Taro and Anna move from house to post office.

Input sentence (as Taro): “Please take me to home.”

Output from the system: “Let’s go to the home. From here, the post office. Next, go to the yard. Now, we arrive at the home.” (As shown in Fig 5.21)

While Fig. 5.22 shows the changement of its animation screen.



```
Your command: Please take me to home.

    >>> Time-consuming dependency tree = 0.006 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['x', 'me', 'p', 'home', 'a', 'v', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: Let's go to the home.

Anna: From here, the post_office,
     then, pass the yard.
     Now, we arrive at the home.

    >>> Time-consuming map reading and animation generator = 14.530 seconds.
```

Fig. 5.21 Conversation screen when input is “Please take me to home”.

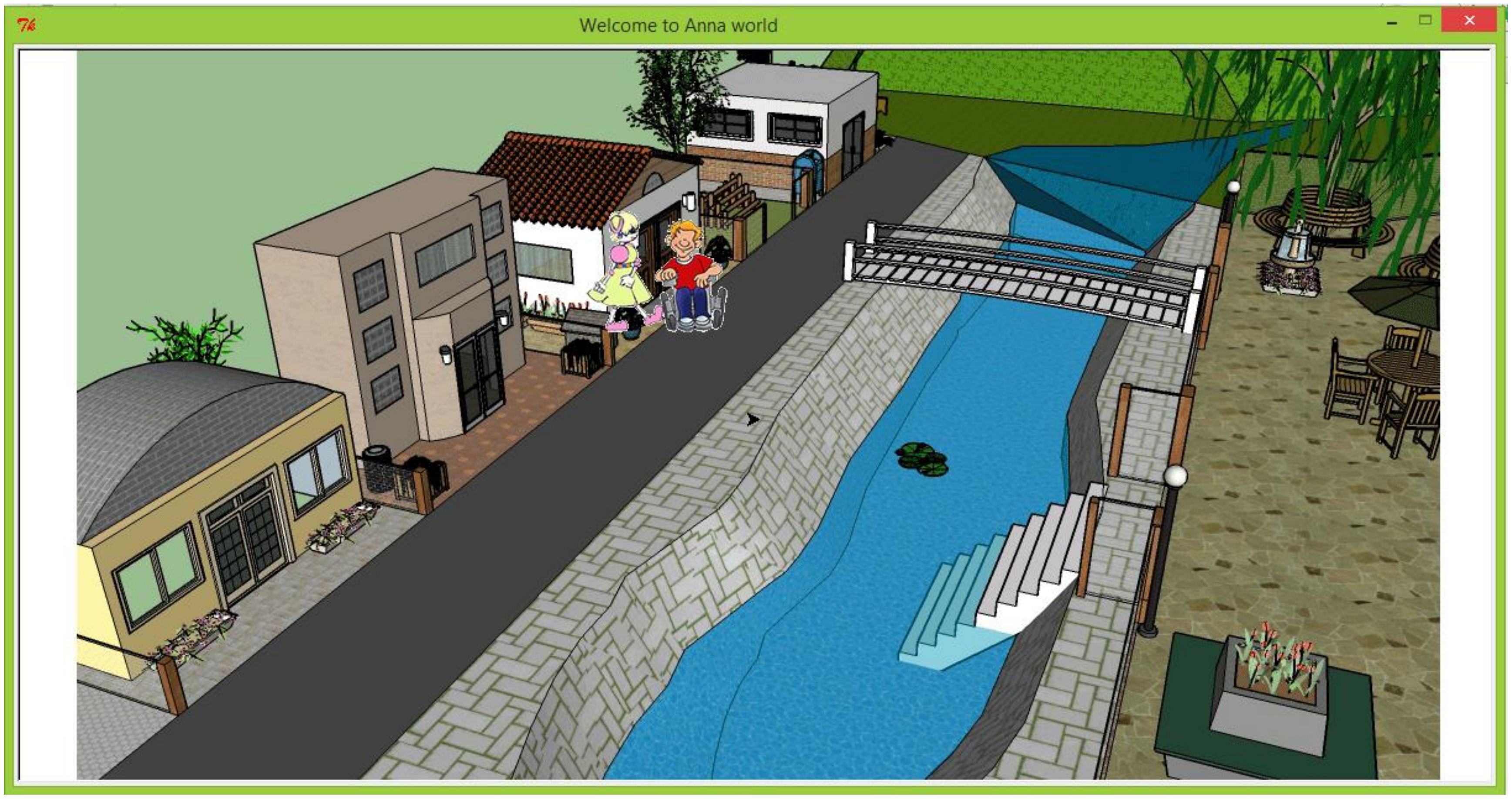
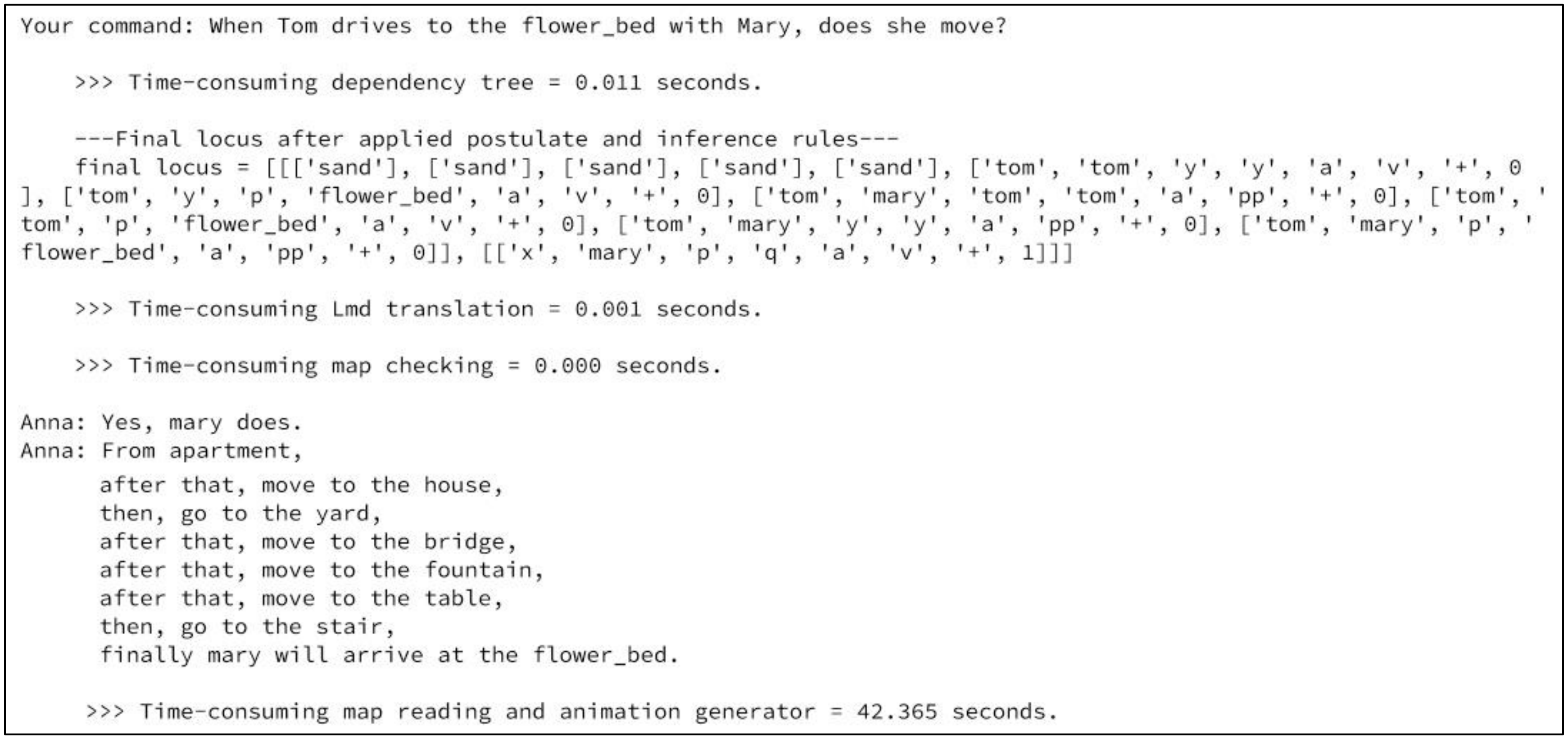


Fig. 5.22 Changement of animation screen that Taro and Anna move from post office to house.

Input sentence (as Taro): “When Tom drives to the flower bed with Mary, does she move?”

Output from the system: “Sure, Mary does.”, “From apartment, then go to the house. After that, move to the yard. Then, go to the bridge. After that, move to the fountain. Then, go to the table. Then, go to the stair. Finally, Mary will arrive at the flower bed.” (As shown in Fig 5.23)

While, Fig. 5.24 – 5.25 shows the changements of animation screen.



```
Your command: When Tom drives to the flower_bed with Mary, does she move?

    >>> Time-consuming dependency tree = 0.011 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand'], ['sand'], ['sand'], ['sand'], ['sand'], ['tom', 'tom', 'y', 'y', 'a', 'v', '+', 0
], ['tom', 'y', 'p', 'flower_bed', 'a', 'v', '+', 0], ['tom', 'mary', 'tom', 'tom', 'a', 'pp', '+', 0], ['tom', '
tom', 'p', 'flower_bed', 'a', 'v', '+', 0], ['tom', 'mary', 'y', 'y', 'a', 'pp', '+', 0], ['tom', 'mary', 'p', '
flower_bed', 'a', 'pp', '+', 0]], [['x', 'mary', 'p', 'q', 'a', 'v', '+', 1]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: Yes, mary does.
Anna: From apartment,
      after that, move to the house,
      then, go to the yard,
      after that, move to the bridge,
      after that, move to the fountain,
      after that, move to the table,
      then, go to the stair,
      finally mary will arrive at the flower_bed.

     >>> Time-consuming map reading and animation generator = 42.365 seconds.
```

Fig. 5.23 Conversation screen when input is “When Tom drives to the flower bed with Mary, does she move?”.

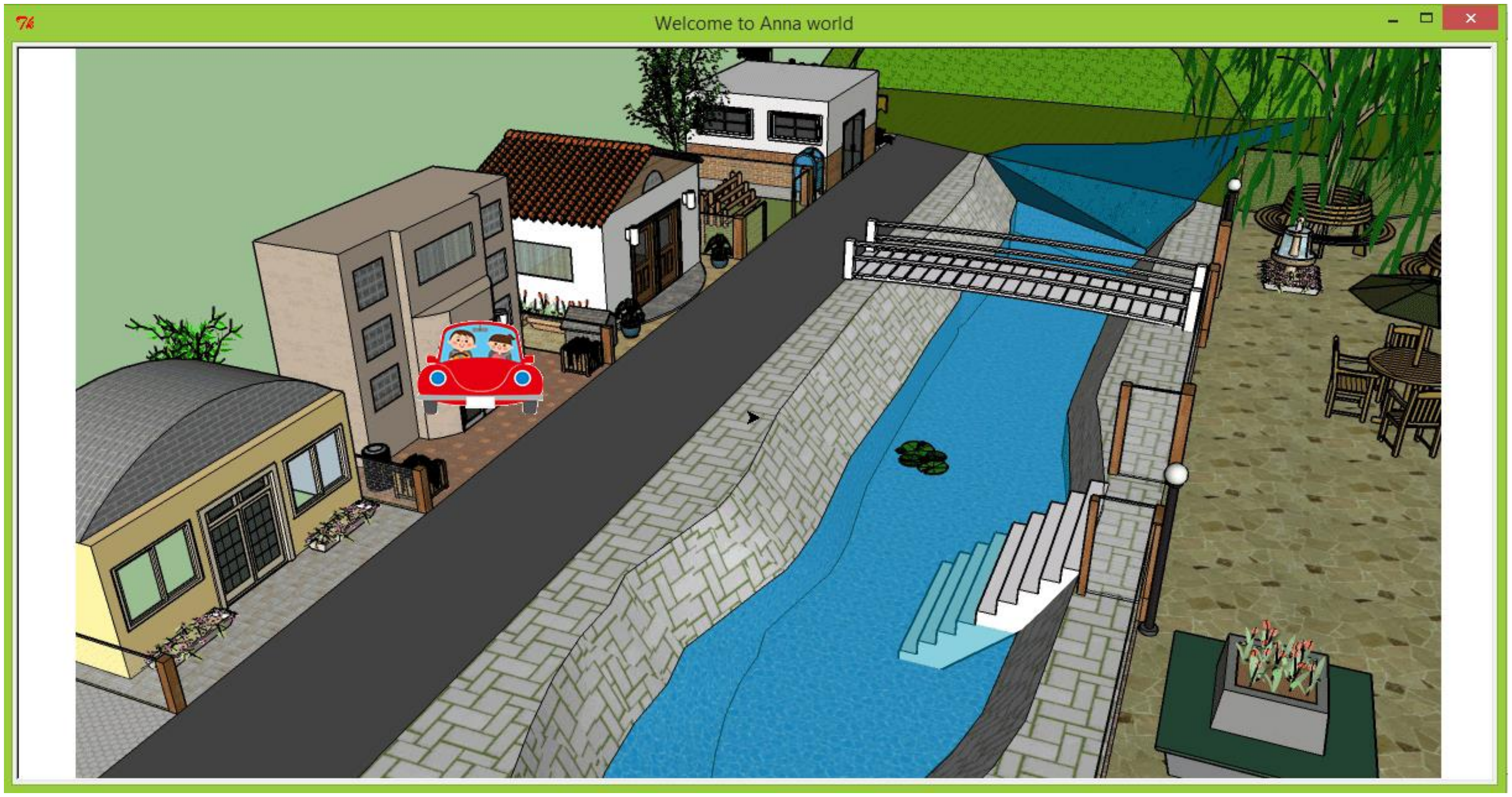


Fig. 5.24 Tom and Mary are their apartment, before moving to the flower bed.

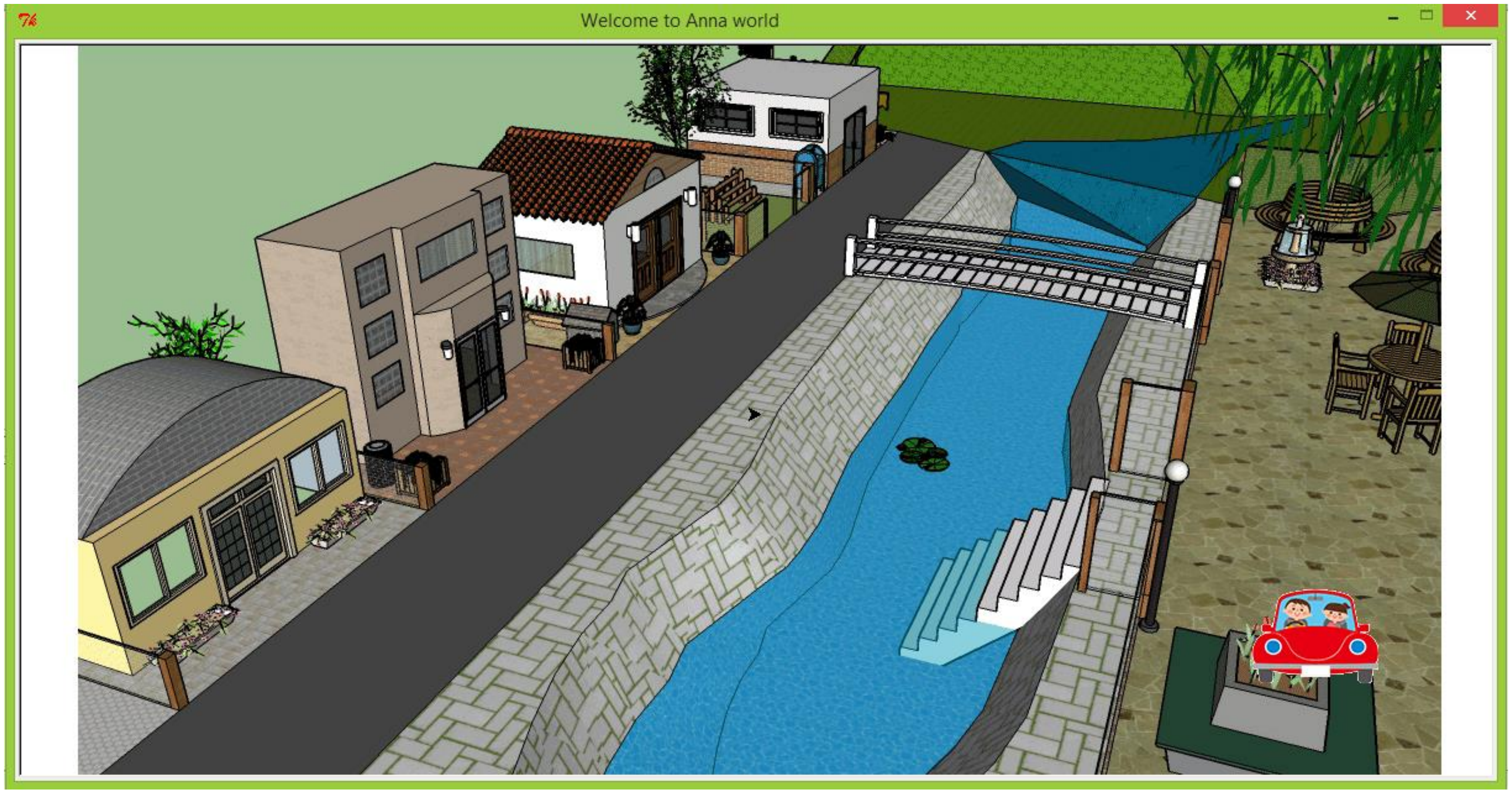
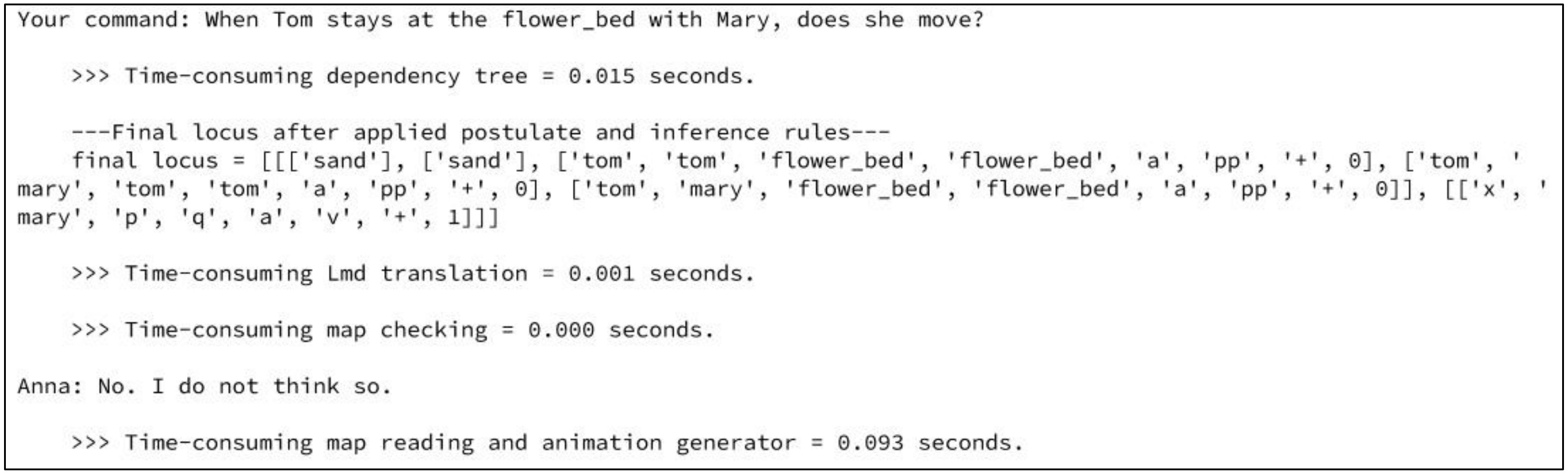


Fig. 5.25 Tom and Mary are the flower bed, after moving from the apartment.

Input sentence (as Taro): “When Tom stays at the flower bed with Mary, does she move?”

Output from the system: “No. I do not think so.” (As shown in Fig 5.26)

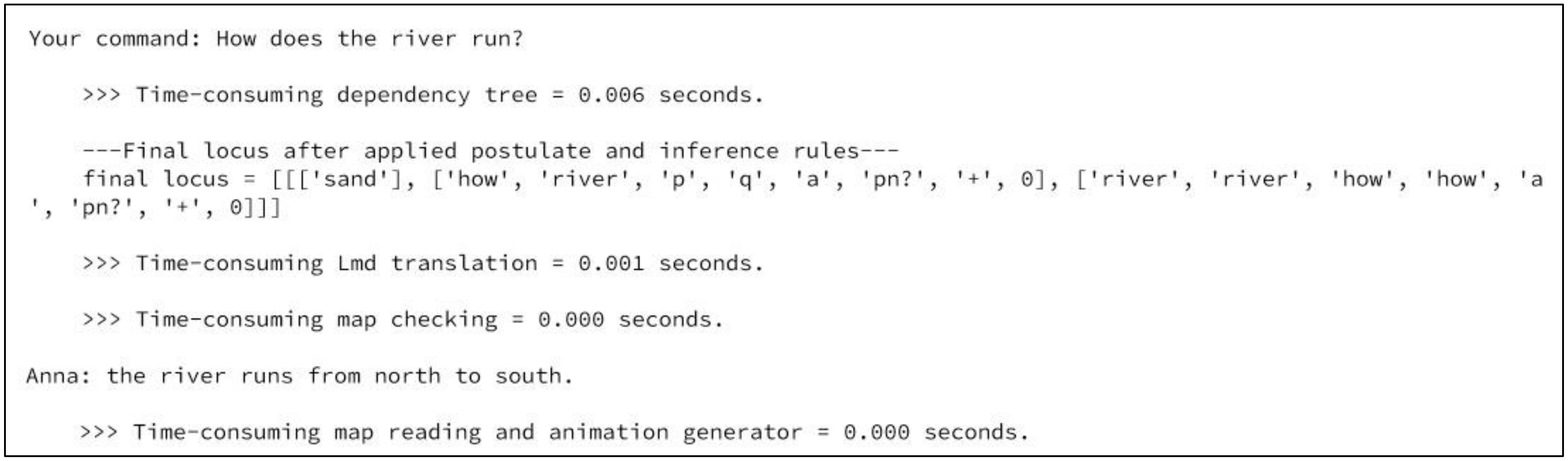


```
Your command: When Tom stays at the flower_bed with Mary, does she move?

    >>> Time-consuming dependency tree = 0.015 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand'], ['sand'], ['tom', 'tom', 'flower_bed', 'flower_bed', 'a', 'pp', '+', 0], ['tom', '
mary', 'tom', 'tom', 'a', 'pp', '+', 0], ['tom', 'mary', 'flower_bed', 'flower_bed', 'a', 'pp', '+', 0]], [['x', '
mary', 'p', 'q', 'a', 'v', '+', 1]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: No. I do not think so.

    >>> Time-consuming map reading and animation generator = 0.093 seconds.
```

Fig. 5.26 Conversation screen when input is “When Tom stays at the flower bed with Mary, does she move?”.

Input sentence (as Taro): "How does the river run?"

Output from the system: "The river runs from north to south." (As shown in Fig 5.27)

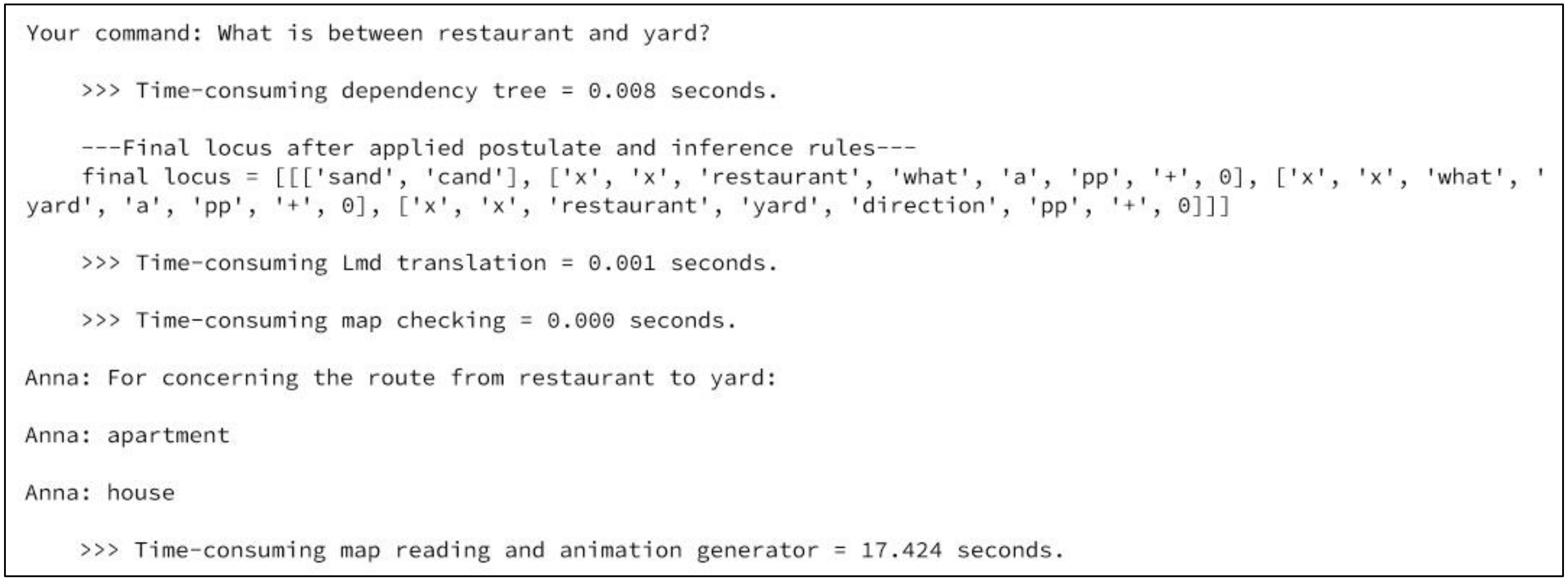


```
Your command: How does the river run?

    >>> Time-consuming dependency tree = 0.006 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand'], ['how', 'river', 'p', 'q', 'a', 'pn?', '+', 0], ['river', 'river', 'how', 'how', 'a
', 'pn?', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: the river runs from north to south.

    >>> Time-consuming map reading and animation generator = 0.000 seconds.
```

Fig. 5.27 Conversation screen when input is "How does the river run?".

Input sentence (as Taro): “What is between* restaurant and yard?”

Output from the system: “For concerning the route from restaurant to yard: apartment and house” (As shown in Fig 5.28)



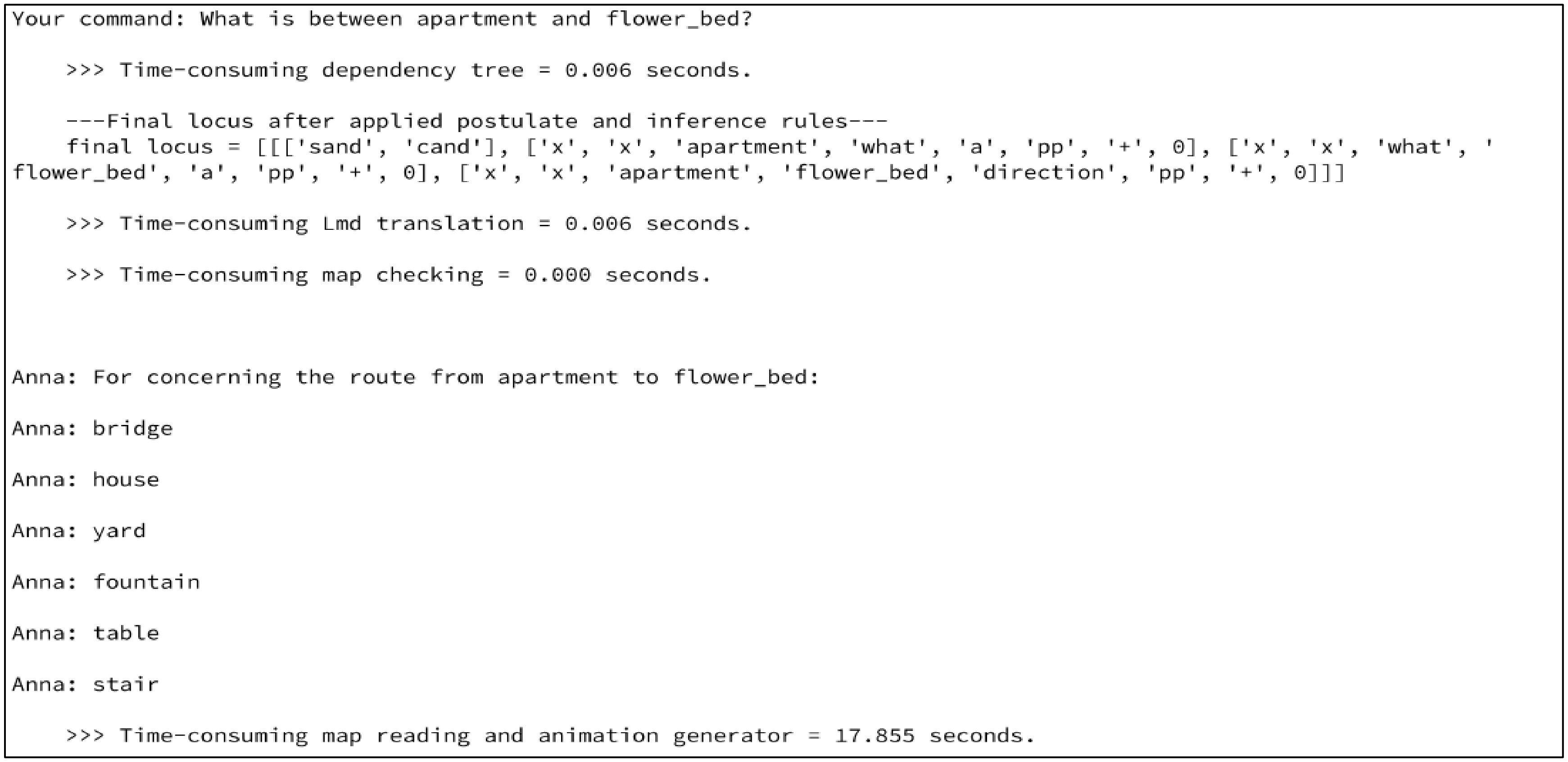
```
Your command: What is between restaurant and yard?

    >>> Time-consuming dependency tree = 0.008 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand', 'cand'], ['x', 'x', 'restaurant', 'what', 'a', 'pp', '+', 0], ['x', 'x', 'what', '
yard', 'a', 'pp', '+', 0], ['x', 'x', 'restaurant', 'yard', 'direction', 'pp', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: For concerning the route from restaurant to yard:

Anna: apartment

Anna: house

    >>> Time-consuming map reading and animation generator = 17.424 seconds.
```

Fig. 5.28 Conversation screen when input is “What is between restaurant and yard?”.

* In this work, there are two definition of “between” that is (1) concerning the route from one place and another place, and (2) concerning the coordinate(s) between two places.

Input sentence (as Taro): “What is between apartment and flower bed?**”

Output from the system: “For concerning the route from apartment to flower bed: bridge, house, yard, fountain, table, stair” (As shown in Fig 5.29)



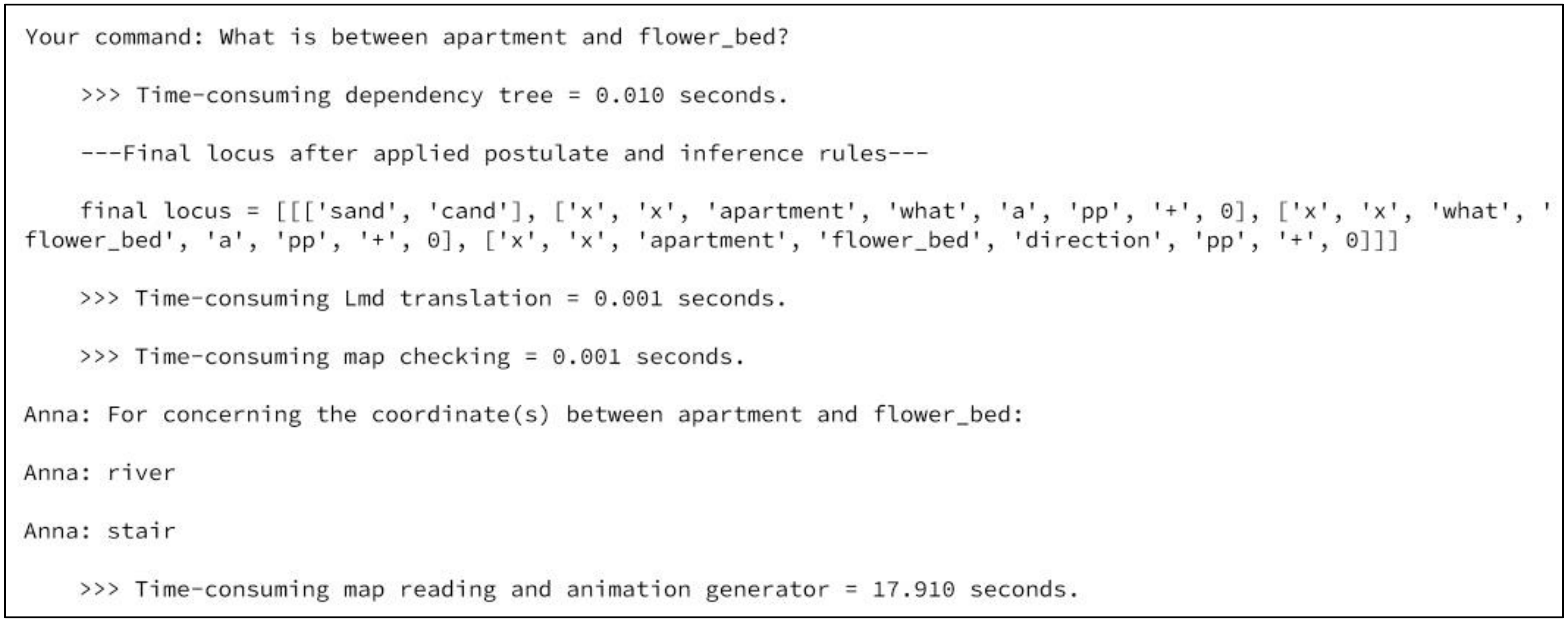
```
Your command: What is between apartment and flower_bed?

    >>> Time-consuming dependency tree = 0.006 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand', 'cand'], ['x', 'x', 'apartment', 'what', 'a', 'pp', '+', 0], ['x', 'x', 'what', '
flower_bed', 'a', 'pp', '+', 0], ['x', 'x', 'apartment', 'flower_bed', 'direction', 'pp', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.006 seconds.

    >>> Time-consuming map checking = 0.000 seconds.



Anna: For concerning the route from apartment to flower_bed:

Anna: bridge

Anna: house

Anna: yard

Anna: fountain

Anna: table

Anna: stair

    >>> Time-consuming map reading and animation generator = 17.855 seconds.
```

Fig. 5.29 Conversation screen when input is “What is between apartment and flower bed?**”.

** This example, the definition of “between” is concerning the route from one place and another place.

Input sentence (as Taro): "What is between apartment and flower bed?***"

Output from the system: "For concerning the coordinate(s) between apartment to flower bed: river and stair" (As shown in Fig 5.30)



```
Your command: What is between apartment and flower_bed?

    >>> Time-consuming dependency tree = 0.010 seconds.

    ---Final locus after applied postulate and inference rules---

    final locus = [[['sand', 'cand'], ['x', 'x', 'apartment', 'what', 'a', 'pp', '+', 0], ['x', 'x', 'what', '
flower_bed', 'a', 'pp', '+', 0], ['x', 'x', 'apartment', 'flower_bed', 'direction', 'pp', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.001 seconds.

    >>> Time-consuming map checking = 0.001 seconds.

Anna: For concerning the coordinate(s) between apartment and flower_bed:

Anna: river

Anna: stair

    >>> Time-consuming map reading and animation generator = 17.910 seconds.
```

Fig. 5.30 Conversation screen when input is "What is between apartment and flower bed?***".

*** This example, the definition of "between" is concerning the coordinate(s) between two points.

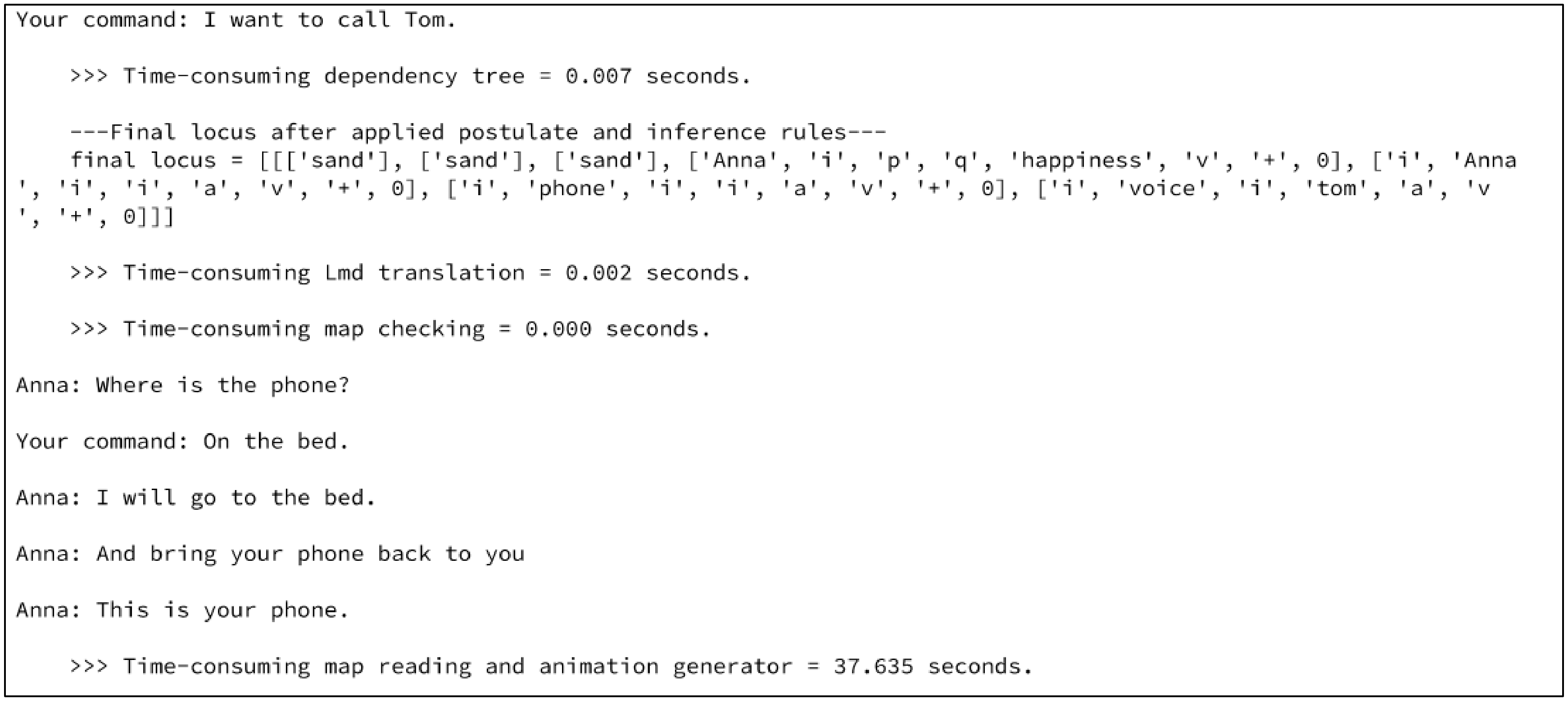
Input sentence (as Taro): “I want to call Tom.”

Output from the system: “Where is the phone?”

Input sentence (as Taro): “On the bed.”

Output from the system: “I will go to the bed, and bring your phone back to you. This is your phone.” (As shown in Fig 5.31).

While Fig. 5.32 – 5.34 show the changements of animation screen.



```
Your command: I want to call Tom.

    >>> Time-consuming dependency tree = 0.007 seconds.

    ---Final locus after applied postulate and inference rules---
    final locus = [[['sand'], ['sand'], ['sand'], ['Anna', 'i', 'p', 'q', 'happiness', 'v', '+', 0], ['i', 'Anna
', 'i', 'i', 'a', 'v', '+', 0], ['i', 'phone', 'i', 'i', 'a', 'v', '+', 0], ['i', 'voice', 'i', 'tom', 'a', 'v
', '+', 0]]]

    >>> Time-consuming Lmd translation = 0.002 seconds.

    >>> Time-consuming map checking = 0.000 seconds.

Anna: Where is the phone?

Your command: On the bed.

Anna: I will go to the bed.

Anna: And bring your phone back to you

Anna: This is your phone.

    >>> Time-consuming map reading and animation generator = 37.635 seconds.
```

Fig. 5.31 Conversation screen when input is “I want to call Tom.”.

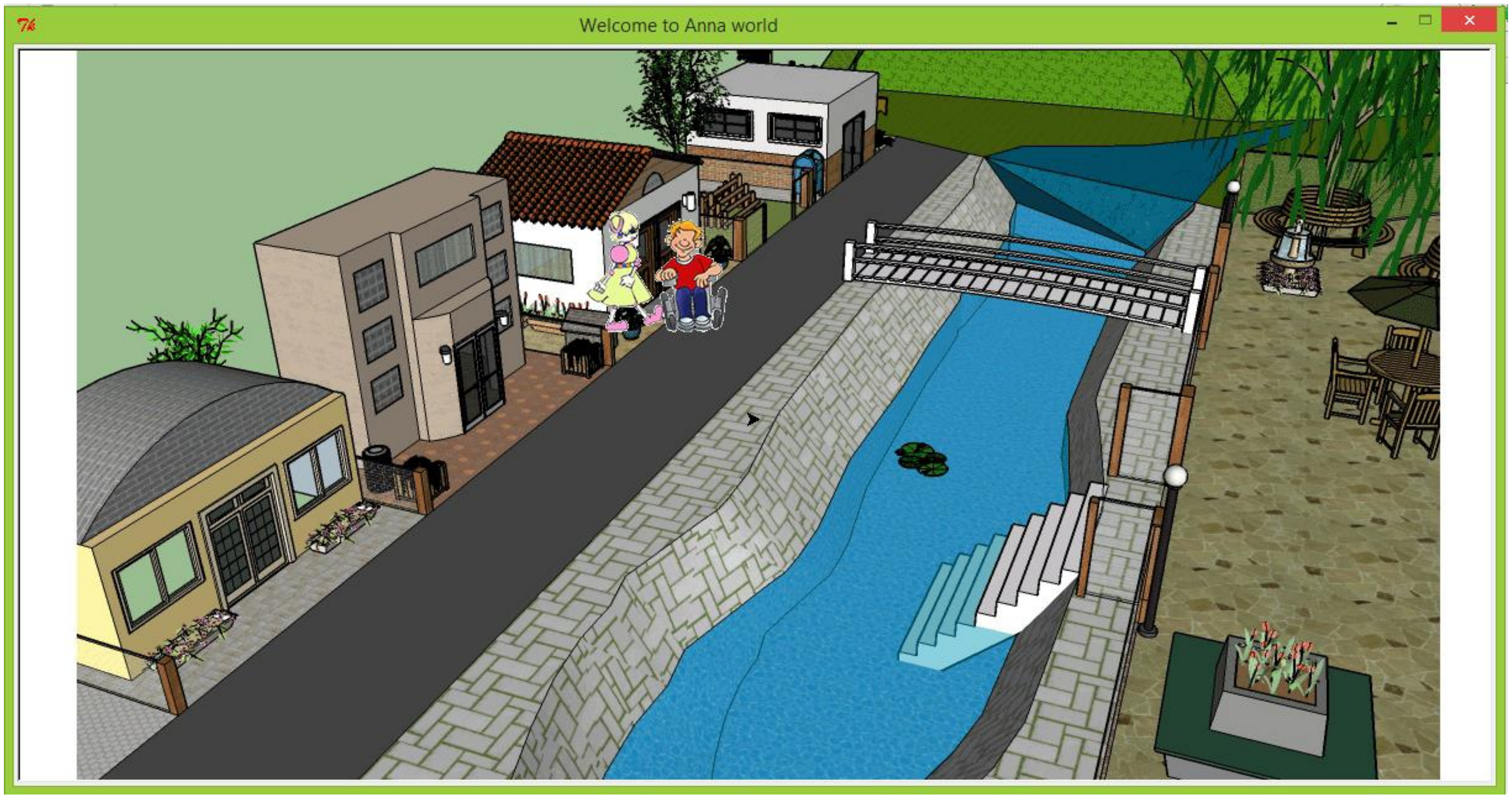


Fig. 5.32 Changement of animation screen after enter input sentence as “I want to call Tom.”.

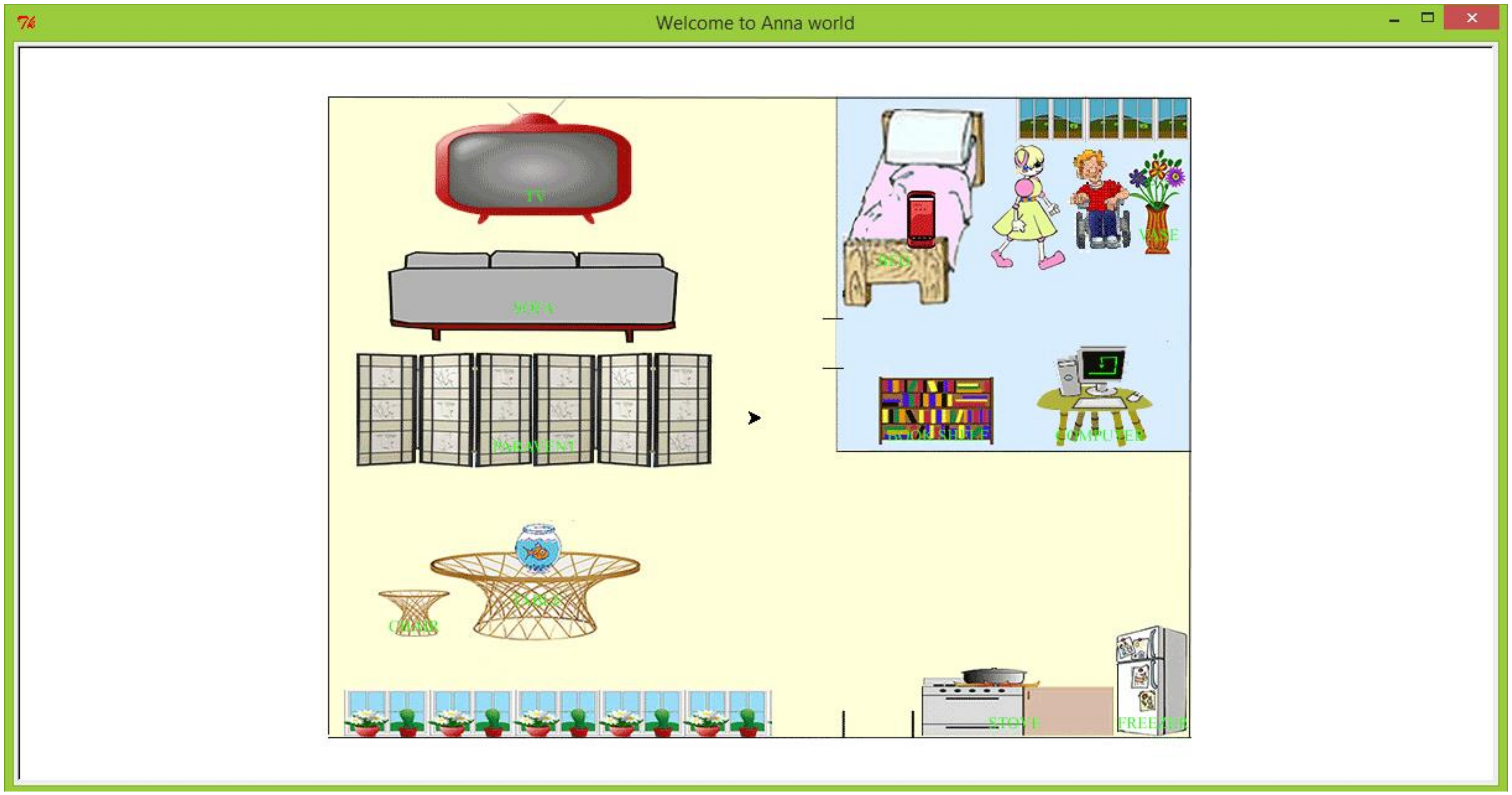


Fig. 5.33 Animation screen changes to house screen, and the phone is on the bed.

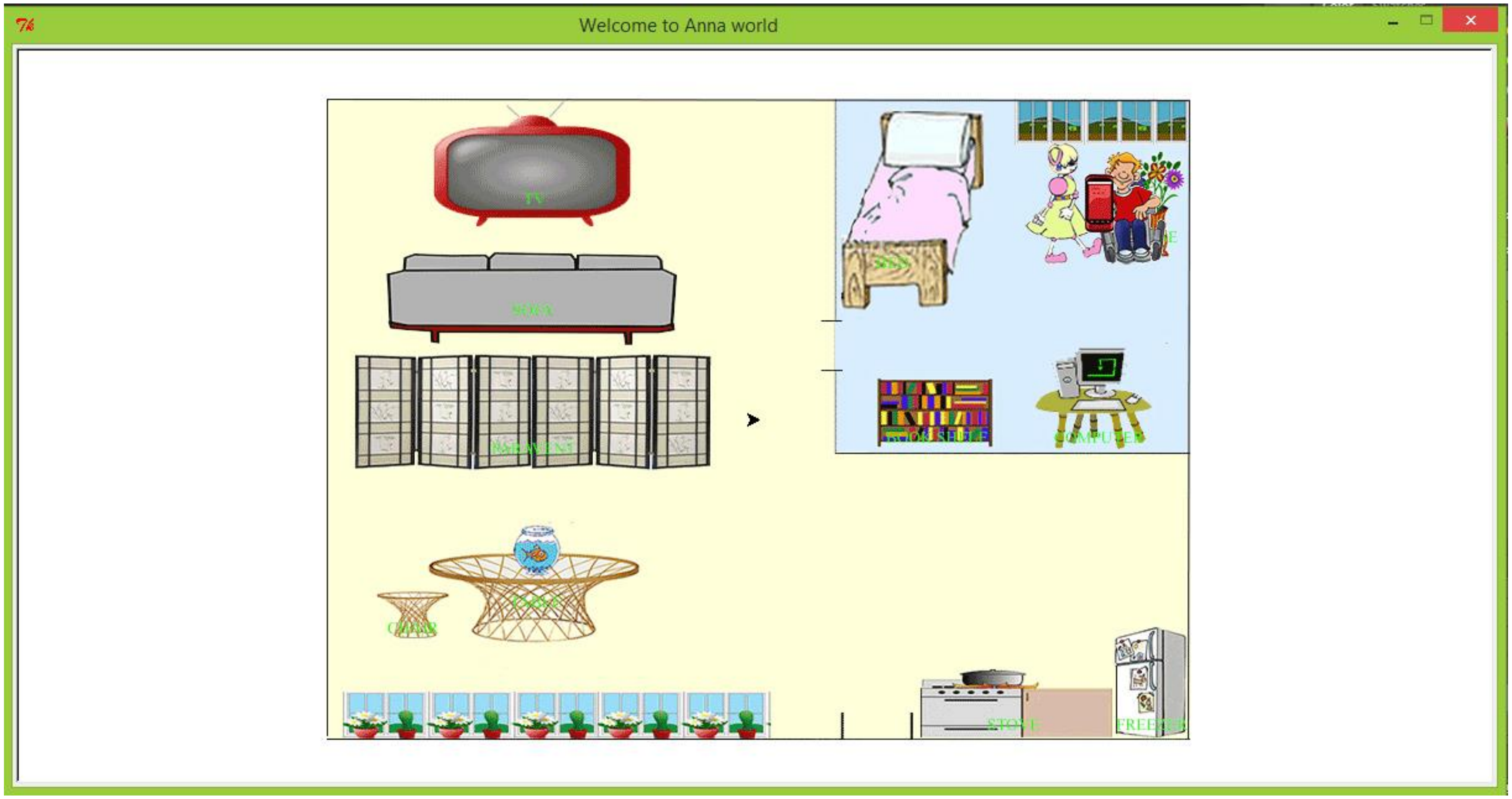


Fig. 5.34 Anna brings the phone to Taro.

CHAPTER 6

DISCUSSION AND CONCLUSION

This section is discussion and conclusion that summarizes some information of this work as the following details.

This work proposed Mental – Image Based Understanding (MBU), designed so as for people to search the image of the assertion (such as, “Tom was with the book in the bus running from Town to University”) for the specific image evoked by the question (e.g., “Did Tom carry the book from Town to University”, and so on) and simulated such human mental imagery in use of $\boldsymbol{L_{md}}$, resulting in a good success. Actually, people are considered to perform this in a top - down or predictive way where their awareness is triggered and driven by the event pattern included in the question (c.f., Fig. 2.5), which is difficult to be described as a simple way of deduction from the image of the assertion to that of the question here.

Conventional naïve semantics employ relative definitions of a word concept in context of the others. Quite distinguished from the conventional approaches, MIDST defines it language-freely with its systematic interpretation based on the mental image model grounded in sensory events. This fact is to contribute to compact description and computation of natural semantics especially for intuitive human - robot interaction beyond the conventional cognitive robotics which is considered to concern exclusively the knowledge representation and reasoning problems faced by an autonomous robot (or agent) in a dynamic and incompletely known world [6.1].

This is quite distinguished from conventional ones and shows a potential good enough to be a very powerful means for realizing awareness in computer and its understanding. To our best knowledge, there is no research similar to ours, namely, NLU based on the model of mental image processing. Therefore, we cannot present any quantitative comparisons with others while we have already commented on our qualitative advantage to conventional methodologies in the previous papers [5.1] - [5.2], [6.2] – [6.5]. At conclusion, MIDST could provide the MBU system with an effective methodology to return the correct and satisfied answers in question-answering. The system was designed to disambiguate an input sentence for its most plausible *semantic* interpretation by employing the mental images evoked by the entity names. Disambiguation is the most serious problem for any NLP system. Most of current approaches to it are based on the statistics about certain corpora of texts [6.6], [6.7] but they are what lead to the most plausible *syntactic* dependency but not to the most plausible *semantic* interpretation that is most essential to work robots appropriately by words.

One of the most remarkable points of our research is to aim at problem solving by human - robot cooperation especially through communication in human language where the core technologies are NLU and knowledge management based on $\boldsymbol{L_{md}}$. Conventional methodologies for problem solving inevitably employ simple state-space models or so as surveyed by Russel and Norvig [4.2] because of lacking KRLs such as $\boldsymbol{L_{md}}$ capable to formulate complex and dynamic problems in seamless connection with NLU.

The knowledge representation language, $\boldsymbol{L_{md}}$, quite distinctive from others, was invented in order to model mental imagery subjective to humans and it has been found that the word concepts concerning space can be formalized systematically in $\boldsymbol{L_{md}}$. In spite of tremendous work concerned, neither mental image nor NLU, even the meaning of *meaning* itself, has been given any definite definition yet. Distinguished from conventional NLU designed for QA between people and computers, the semantics for NL in cognitive robotics should be grounded on robotic sensors and actuators. For example, we must have robots behave in the environment according to people's commands such as "Put the book on the chair between the table and the desk", where of course the robot must judge whether or not the environment will permit its intended behavior.

Anna, a robot on imagination, was implemented in Python as an AI for managing conversation in NL. Her reasoning here is based on her conception in Natural Semantic (S_N) for humans and therefore she must interpret her solution in (Artificial Semantic) S_A for robots when she as a real robot acts it out in order to control her actuators/sensors properly.

That is, certain simulation in advance is needed based on the sensory information about the environment. If there are any obstacles, the robot must avoid or remove them on the way of its action. That is, the robot assesses the environment in advance to its action partially or totally based on inferential computation in certain internal representation of the environment. $\boldsymbol{L_{md}}$ has been proposed as a candidate formal language for systematization of such internal representation. The inferences in $\boldsymbol{L_{md}}$ are based on simple and general rules about atomic loci and therefore CMS works feasibly in Python except for computational cost in the animation generator.

Table 6.1 Time consuming in Conversation Management System (CMS)

Input/Stimulus sentence	**Time consuming (second)**[*]				
	Map loading	**Text into dependency tree**	**L_{md} traslation**	**Pattern matching and map reading**	**Animation genertation**
When Tom drives a car to the fountain with Mary, does she move?	14.892	0.103	0.003	0.000	40.514
When Tom stay with Mary at the fountain, does she move?	14.607	0.097	0.003	0.000	0.628
How can I go to the flower bed?	14.685	0.099	0.003	0.000	13.162
Please take me to the flower bed.	14.717	0.095	0.002	0.000	15.689
I want to call Tom.	14.825	0.075	0.012	0.000	33.330

[*] *This experiment run on Windows 7 Enterprise, Intel Celeron CPU 2.00GHz. So, time consuming may be diversify up to computer architecture and so forth.*

Table 6.1 shows computational cost in the system that is developed to support for 73 words of noun and pronoun, 22 words of verb, 10 words of preposition, 5 words of adverb and adjective (see APPENDIX C for detail). From the table, the data can summarize that most of time consuming in this system can be divided into two parts, i.e. animation process part and reasoning process part.

Animation process part consists of "map loading" and "animation generation" (2nd and 6th column of the table). In this portion, the system must be associate with computer[*] storage because too much information (e.g. town map picture, characters, etc.) will be loaded to generate graphic screen. Therefore, a period of time is used to prepare such basic elements. On the other hand, if taking a notice to reasoning processes that is "text into dependency tree", "L_{md} translation", and "pattern matching and map reading", we found that these processes spend very few time when compare to animation process. As a result, we can epitomize that MIDST is a methodology that can utilize for semantic interpretation that influence robot with capability of natural language understanding as well as humans based on a mental image model.

Moreover, we have analyzed a considerable number of spatial terms over various kinds of English words such as prepositions, verbs, adverbs, etc. categorized as Dimensions, Form and Motion in the class SPACE of the Roget's thesaurus, and found that almost all the concepts of spatial events can be defined in exclusive use of 5 kinds of attributes for FAOs, namely, Physical location (A_{12}), Direction (A_{13}), Trajectory (A_{15}), Mileage (A_{17}) and Topology (A_{44}). This implies that location-aware systems based on the formal system are very feasible in the respect of the size of knowledge to be installed. The inferences in $\boldsymbol{L_{md}}$ are based on simple and general rules about atomic loci and therefore CMS works feasibly in Python except for computational cost in the animation generator.

Recently, deep neural networks (DNNs) have been achieving remarkable performance on various pattern-recognition tasks, especially visual classification problems. Simultaneously, interesting differences have been pointed out between human vision and current DNNs arousing questions about the generality of DNN computer vision because there exist images that are completely unrecognizable to humans, but that DNNs believe to be recognizable objects with 99.99% confidence [6.8]. However, DNNs as is cannot accept any feedback from higher level cognition such as reasoning in order to calibrate misclassification because of lacking any immediate means to convert knowledge representation (as awareness of misclassification by reasoning) into weight sets for connectionism. This may lead to such a claim that artificial intelligence (AI) should be more cognitive [6.9]. We believe that this research as well can give a good suggestion about how machine learning should acquire knowledge to be available for higher level cognition such as abstract reasoning. Our future work will include systematic incorporation of machine learning into our conversation management system for automatic acquisition of natural concept and knowledge.

APPENDIX A

A.1 List of Attribute Constants

The attribute constants that used in this work, are related to human activities in daily life. M. Yokota [1.11] had classified it into five kinds of attribute that is sight, hearing, smell, taste, and feeling, exclusively extracted from Japanese and English words as shown in Table A.1.

Table A.1 Example of attribute constants in $\boldsymbol{L_{md}}^{*}$.

Attribute [Property][**]	**Linguistic expression for attribute values**
A. Attribute concerning the physical world.	
A_{01} World of existence [V]	He is in Tokyo. The accident happened in Osaka.
A_{02} Length [S]	The stick is 2 meters long.
A_{03} Height [S]	The tree is 2 meters high.
A_{04} Width [S]	The door is 2 meters wide.
A_{05} Thickness [S]	The book is 2 meters thick.
A_{06} Depth1 [S]	The swimming pool is 2 meters deep.
A_{07} Depth2 [S]	The cave is 100 meters deep.
A_{08} Diameter [S]	The hole is 2 meters across.
A_{09} Area [S]	The crop field is 10 square miles.
A_{10} Volume [S]	The box is 10 cubic meter.
A_{11} Shape [N]	The cake is round.
A_{12} Physical location [N]	Tom moved to Tokyo.
A_{13} Direction [N]	The box is to the left of the chair.
A_{14} Orientation [N]	The door faces to the south.
A_{15} Trajectory [N]	The plane circle in the sky.
A_{16} Velocity [S]	The boy runs very fast.
A_{17} Mileage [S]	The car ran ten miles.

Attribute [Property]**	Linguistic expression for attribute values
A_{18} Strength of effect [S]	He is very strong.
A_{19} Direction of effect [N]	He pulled the door.
A_{20} Density [S]	The milk is very dense.
A_{21} Hardness [S]	The candy is very soft.
A_{22} Elasticity [S]	The sheet is elastic.
A_{23} Toughness [S]	The glass is very brittle.
A_{24} Feeling [S]	The cloth is smooth.
A_{25} Moisture [S]	The paint is still wet.
A_{26} Viscosity [S]	The liquid is oily.
A_{27} Weight [S]	The metal is very light.
A_{28} Temperature [S]	It is hot today.
A_{29} Taste [N]	The grapes here are very sour.
A_{30} Odour [N]	The gas is pungent.
A_{31} Sound [N]	His voice is very loud.
A_{32} Color [N]	Tom painted the desk white.
A_{33} Internal sensation [N]	I am very tired.
A_{34} Time point [S]	It is ten o'clock.
A_{35} Duration [S]	He studies for two hours every day.
A_{36} Number [S]	There are many people.
A_{37} Order [S]	Tom sat next to Mary.
A_{38} Frequency [S]	He did it twice.
A_{39} Vitality [S]	The old man still alive.
A_{40} Sex [S]	The operator is female.
A_{41} Quality [V]	We make cheese from milk.
A_{42} Name [V]	The father named his baby Thomas.
A_{43} Conceptual category [V]	A whale is a mammal.

Attribute [Property][**]	**Linguistic expression for attribute values**
A_{44} Topology [N]	He is in the room.
A_{45} Angularity [S]	The knife is dull.
B. Attributes concerning the mental world.	
B_{01} Worth [V]	The house was damaged.
B_{02} Location of information [V]	We think that…
B_{03} Emotion [V]	I like him.
B_{04} Belief value [S]	I believe that…
B_{05} Truth value [S]	I realize that…
B_{06} Location of ownership [V]	I give him a book.
B_{07} Location of usership [V]	I borrowed a book from him.

[*] The original of this table comes from M. Yokota [1.11]

[**] S is "Scalar value", and N is "non-scalar value".

A.2 List of Standard Constants

Standard constants are necessary to represent values of attributes in Table A.1 that classified into six categories [1.11] as shown in Table A.2.

Table A.2 Example of standard constants in $\boldsymbol{L_{md}}$[*].

Categories of standards	**Remarks**
Rigid standard	Objective standards such as denoted by measuring *units* (meter, gram, etc.).
Species standard	The *attribute value ordinary* for a species. A *short train* is ordinarily longer than a *long pencil*.
Proportional standard	"*Oblong*" means that the width is greater than the height at a physical object.
Individual standard	*Much* money for one person can be too *little* for another.
Purposive standard	One room large enough for a person's *sleeping* must be too small for his *jogging*.
Declarative standard	Tom is taller *than Jim*. The origin of an order such as 'next' must be declared explicitly just as "next *to him*".

[*] The original of this table comes from M. Yokota [1.11]

A.3 List of Temporal Relations

M. Yokota [A.1] gave the definition of τ_i that provided by Table A.3**,** where the durations of χ_1 and χ_2 are $[t_{11}, t_{12}]$ and $[t_{21}, t_{22}]$, respectively. This table shows the complete list of temporal relations between two intervals, where 13 types of relations are discriminated by the suffix "i" ($-6 \le i \le 6$).

Table A.3 List of temporal relations*.

Temporal relations and definition of τ_i			**Allen's notation****
χ_1 +…………………………..+	$t_{11} = t_{21} \,\Lambda\, t_{12} = t_{22}$	$\tau_0(\chi_1, \chi_2)$	equals(χ_1, χ_2)
χ_2 +…………………………..+		$\tau_0(\chi_2, \chi_1)$	equals(χ_2, χ_1)
χ_1 +………...….+	$t_{12} = t_{21}$	$\tau_1(\chi_1, \chi_2)$	meets(χ_1, χ_2)
χ_2 +………...….+		$\tau_{-1}(\chi_2, \chi_1)$	met - by(χ_2, χ_1)
χ_1 +………...….+	$t_{11} = t_{21} \,\Lambda\, t_{12} < t_{22}$	$\tau_2(\chi_1, \chi_2)$	starts(χ_1, χ_2)
χ_2 +…………………………..+		$\tau_{-2}(\chi_2, \chi_1)$	started - by(χ_2, χ_1)
χ_1 +………...….+	$t_{11} > t_{21} \,\Lambda\, t_{12} < t_{22}$	$\tau_3(\chi_1, \chi_2)$	during(χ_1, χ_2)
χ_2 +…………………………..+		$\tau_{-3}(\chi_2, \chi_1)$	contains(χ_2, χ_1)
χ_1 +………...….+	$t_{11} > t_{21} \,\Lambda\, t_{12} = t_{22}$	$\tau_4(\chi_1, \chi_2)$	finishes(χ_1, χ_2)
χ_2 +…………………………..+		$\tau_{-4}(\chi_2, \chi_1)$	finished - by(χ_2, χ_1)
χ_1 +……..….+	$t_{12} < t_{21}$	$\tau_5(\chi_1, \chi_2)$	before(χ_1, χ_2)
χ_2 +……..….+		$\tau_{-5}(\chi_2, \chi_1)$	after(χ_2, χ_1)
χ_1 +……………..….+	$t_{11} < t_{21}$	$\tau_6(\chi_1, \chi_2)$	overlaps(χ_1, χ_2)
χ_2 +………….……..+	$\Lambda\, t_{21} < t_{12} \,\Lambda\, t_{12} < t_{22}$	$\tau_{-6}(\chi_2, \chi_1)$	overlepped - by(χ_2, χ_1)

* The original of this table comes from M. Yokota [A.1]

** This is in accordance with the conventional notation [2.2] which, to be strict, is for "temporal conjunctions (= Λ_i)" but not for pure "temporal relations (= τ_i)".

APPENDIX B

B.1 Overview of Psycholinguistic Experiment

This thesis focuses on “Mental Image Directed Semantic Theory” (MIDST) that imitates human thinking in a methodology based on formal logic, so called, “Mental-Image Based Understanding” (MBU). Here is described about psycholinguistic experiment on six human subjects in order to evaluate MBU. In this experiment, three types of stimulus sentence, i.e., SS+PrP, SS+PaP, and SS+C+SS as listed below were employed and the semantic definitions of the words involved were based on the Merriam-Webster dictionary.

The subject group consisted of various people from several countries, i.e. two Americans, one Chinese, two Thais, and one Japanese. In the survey, we applied all three-type stimulus sentences to the subjects, that is,

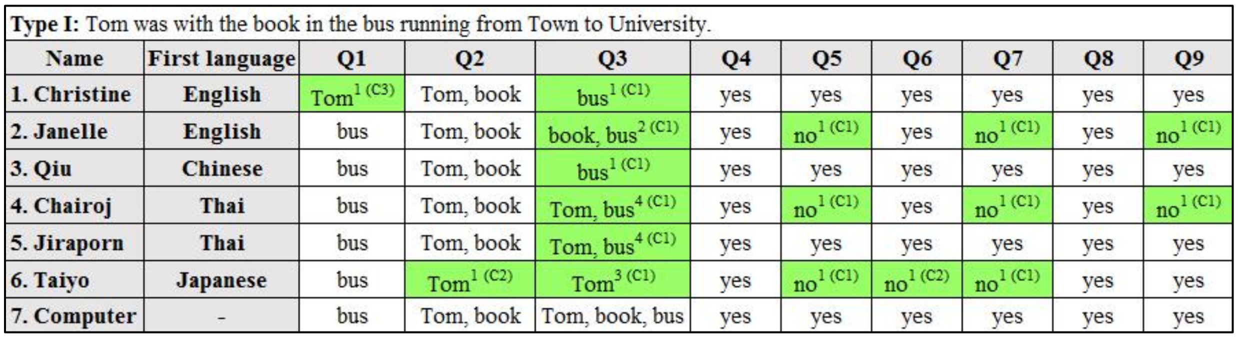
- [Type I] SS+PrP
  - Tom was with the book in the bus *running* from Town to University.
- [Type II] SS+PaP
  - Tom was with the book in the car *driven* from Town to University by Mary.
- [Type III] SS+C+SS
  - Tom kept the book in a box *before* he drove the car from Town to University with the box.

Here, SS, PrP, PaP, and C represent “Simple Sentence”, “Present Particle Construction”, “Past Particle Construction”, and “Conjunction”, respectively.

B.2 Type I sentence

Nine questions were asked to examinees and the results are shown as Table B.1.

Table B.1. The surveying of Type I stimulus sentence.



Type I: Tom was with the book in the bus running from Town to University.										
Name	**First language**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**
1. Christine	**English**	Tom$^{1\ (C3)}$	Tom, book	bus$^{1\ (C1)}$	yes	yes	yes	yes	yes	yes
2. Janelle	**English**	bus	Tom, book	book, bus$^{2\ (C1)}$	yes	no$^{1\ (C1)}$	yes	no$^{1\ (C1)}$	yes	no$^{1\ (C1)}$
3. Qiu	**Chinese**	bus	Tom, book	bus$^{1\ (C1)}$	yes	yes	yes	yes	yes	yes
4. Chairoj	**Thai**	bus	Tom, book	Tom, bus$^{4\ (C1)}$	yes	no$^{1\ (C1)}$	yes	no$^{1\ (C1)}$	yes	no$^{1\ (C1)}$
5. Jiraporn	**Thai**	bus	Tom, book	Tom, bus$^{4\ (C1)}$	yes	yes	yes	yes	yes	yes
6. Taiyo	**Japanese**	bus	Tom$^{1\ (C2)}$	Tom$^{3\ (C1)}$	yes	no$^{1\ (C1)}$	no$^{1\ (C2)}$	no$^{1\ (C1)}$	yes	yes
7. Computer	-	bus	Tom, book	Tom, book, bus	yes	yes	yes	yes	yes	yes

Questions:

Q1: What ran?

Q2: What was in the bus?

Q3: What traveled from Town to University?

Q4: Did the bus carry Tom from Town to University?

Q5: Did the bus move Tom from Town to University?

Q6: Did the bus carry the book from Town to University?

Q7: Did the bus move the book from Town to University?

Q8: Did Tom carry the book from Town to University?

Q9: Did Tom move the book from Town to University?

From Table B.1, the examinees all agreed that “the bus was running” in Q1, except Christine. Her answer was “Tom was running”. “Because Tom was a passenger and sitting in the bus which was moving from Town to University, moreover Tom was the most important character (or subject) of the sentence, so the answer should be Tom”, she said.

According to the experiment, we could interpret her reasoning process in ***L**$_{md}$* as follows:

$$L(Tom, Tom, Bus, Bus, \Lambda_t)\Pi\ L(Bus, Bus, Town, Univ., \Lambda_t)$$
$$\rightarrow L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.1)}$$

From (eq. B.1), Christine’s original image was $L(Tom, Tom, Bus, Bus, \Lambda_t)\Pi L(Bus, Bus, Town, Univ., \Lambda_t)$, meant that “Tom was in the bus”, and “the bus was running from Town to University”. Because of her own perspective to the stimulus sentence, the Postulate of Matter as Values (MV) was employed. So, the term $L(Tom, Tom, Town, Univ., \Lambda_t)$, "Tom was running from Town to University", was obtained.

Due to her rational process related to grammatical knowledge, so we grouped her motive in Category 3 (C3), "Difference in Reasoning", as noticeable in Table B.1 Below was shown people' rationalizing categories:

(C1) Difference of Personal Concepts

Each person had its own vocabularies' sense, so a same word could be written in various types of ***Lmd*** up to their concepts.

(C2) Difference of Personal Postulates

Because the judgment system of using postulate rules was different in each one, it might lead the dissimilarity of answers.

(C3) Difference in Reasoning

Because everyone was distinct, so the difference in reasoning might occur.

(C4) Difference of Personal Total Image

The interpretations of stimulus sentences were subjective diversely.

(C5) Mistake

The errors due to the original input sentences or examinees.

Following the above categories, let's consider Q2. In this question, Taiyo's answer was separately from others. "The book was in a bag, so it was similar that the book was a part of Tom. So, only Tom was in the bus.", Taiyo said. Owing to his testimony, the formal language ***Lmd*** or Locus Formula could be written as eq. B.2:

$$\begin{gathered} L(\_, Book, Bag, Bag, \Lambda_t) \Pi L(\_, Bag, -Transparent, -Transparent, Color) \\ \Pi L(Tom, Bag, Tom, Tom, \Lambda_t) \Pi L(Tom, Tom, Bus, Bus, \Lambda_t) \\ -/\rightarrow L(\_, Book, Bus, Bus, \Lambda_t) \end{gathered} \qquad \text{(eq. B.2)}$$

In this logical form eq. B.2, Taiyo didn't employ any postulate rule, then his answer didn't include the last term, "the book was in the bus" or $L(\_, Book, Bus, Bus, a)$. Because of this reason, we grouped his answer into C2.

Now go along with Q3, we could separate the responses into four - type. The first one was Christine's and Qiu's answers. They said that "Tom and the book were just stayed in the bus, so the bus was the only one thing that moving from Town to University". While the second group was Janelle's answer, she indicated that "The question was started by 'what' not 'who'", while, "The vocabulary 'travel' could be used for human agents only", Taiyo said. During Chairoj and Jiraporn gave their reason as "The book had no life, and it was being held by Tom. Then the book didn't travel from Town to University".

Since their senses were based on their own concepts, so we merged these all into C1, and their image could turn into Locus Formulas as below:

$$L(Tom, Book, Bus, Bus, \Lambda_t) \Pi L(Bus, Bus, Town, Univ., \Lambda_t) \\ -/\rightarrow L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.3)}$$

$$L(Tom, Book, Bus, Bus, \Lambda_t) \Pi L(Bus, Bus, Town, Univ., \Lambda_t) \\ \rightarrow L(Bus, Book, Town, Univ., \Lambda_t) \qquad \text{(eq. B.4)}$$

$$L(Tom, Book, Tom, Tom, \Lambda_t) \Pi L(Tom, Tom, Bus, Bus, \Lambda_t) \\ \Pi L(Bus, Bus, Town, Univ., \Lambda_t) \rightarrow L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.5)}$$

$$L(Tom, Tom, Bus, Bus, \Lambda_t) \Pi L(Bus, Bus, Town, Univ., \Lambda_t) \\ \Pi L(Tom, Book, Tom, Tom, \Lambda_t) \rightarrow L(Tom, Tom, Town, Univ., \Lambda_t) \\ -/\rightarrow L(Book, Book, Town, Univ., \Lambda_t) \qquad \text{(eq. B.6)}$$

Formula eq. B.3 – eq. B.6 came under Christine's and Qiu's, Janelle's, Taiyo's, and Chairoj's and Jiraporn's explanations respectively, where "–/→" referred to "does not imply" or "no use of postulate", while "→" meant "imply". Thus eq. B.3 and a part of eq. B.6 didn't employ likelihood answers, where eq. B.4, eq. B.5 and a part of eq. B.6 used of Postulate of Causal Chain (CC), MV, and MV serially.

Here please note that $L(\_, Book, Town, Univ., \Lambda_t)$ could be replaced to $L(Book, Book, Town, Univ., \Lambda_t)$ in eq. B.5 and eq. B.6 for Taiyo, Chairoj and Jiraporn cases. Due to the book was changed its location by someone or unknown factor ("_" in the $\boldsymbol{L_{md}}$).

In case of Q5, Q7, and Q9, Janelle and Chairoj answered "No", because their concepts of "carry" and "move" were disparate. "Carry: to bring or hold something by people' hands or machine, and used for changing the location in long distance", while "Move: changing the location of something by itself in short distance" Because of this reason, their answer in Q4 – Q9 were separated clearly due to the meaning of "carry" and "move", where their $\boldsymbol{L_{md}}$ of Q5, Q7, and Q9 could be clustered to C1 and interpreted like eq. B.7.

$$\begin{gathered} L(Tom, Book, Bus, Bus, \Lambda_t)\Pi L(Bus, Bus, Town, Univ., \Lambda_t) \\ -/\rightarrow L(Bus, Tom, Tom, Tom, \Lambda_t)\Pi L(Tom, Tom, Town, Univ., \Lambda_t) \\ -/\rightarrow L(Bus, Book, Book, Book, \Lambda_t)\Pi L(Book, Book, Town, Univ., \Lambda_t) \\ L(Tom, Book, Book, Book, \Lambda_t)\Pi L(Book, Book, Town, Univ., \Lambda_t) \end{gathered} \qquad \text{(eq. B.7)}$$

If compare with computer answers, the readers might suspect why the computer return "Yes" to all questions. As described before, the definition of vocabularies in this system were referable from Merriam - Webster Dictionary, where gave the meaning of "carry" and "move" as follows:

"Carry: to move (something) while holding and supporting it"

"Move: to cause (something or someone) to go from one place or position to another"

As from above, we could imply that "move" was a subset of "carry", so the answers of the system were "Yes" in all questions.

Anyway, the other interesting case was Taiyo. He gave the definition of "move" as "Thing must be changed its location by someone", so we didn't doubt why his answer in Q5 and Q7 were "No". Simultaneously, he didn't apply any postulate rules to his logical form in Q6, so the answer was "No" as well. Taiyo's imagery of Q5 – Q7 could be described as (8), and (9) reflected his image in Q9.

$$\begin{gathered} L(Tom, Book, Tom, Tom, \Lambda_t)\Pi L(Tom, Tom, Bus, Bus, \Lambda_t) \\ \Pi L(Bus, Bus, Town, Univ., \Lambda_t) \\ -/\rightarrow L(Bus, Tom, Town, Univ., \Lambda_t) \\ -/\rightarrow L(Bus, Book, Town, Univ., \Lambda_t) \end{gathered} \qquad \text{(eq. B.8)}$$

$$\begin{gathered} \underline{L(Tom, Book, Tom, Tom, \Lambda_t)\Pi L(Tom, Tom, Bus, Bus, \Lambda_t)} \\ \Pi L(Bus, Bus, Town, Univ., \Lambda_t) \\ \underline{\rightarrow L(Tom, Book, Bus, Bus, \Lambda_t)\Pi L(Bus, Bus, Town, Univ., \Lambda_t)} \\ \rightarrow L(Tom, Book, Town, Univ., \Lambda_t) \end{gathered} \qquad \text{(eq. B.9)}$$

The underlined parts in eq. B.9 were indicated the term using postulate rules. To dedicate MV to the first underlined couple, the term, $L(Tom, Book, Bus, Bus, \Lambda_t)$, was obtained. Then applied MV to underlined couple again, the target result, $L(Tom, Book, Town, Univ., \Lambda_t)$ was gained.

B.3 Type II sentence

The second type of examining input contained Simple Sentence and Past Particle Construction, and Table B.2 shown the outcome of the interview.

Table B.2 The surveying of Type II stimulus sentence.

Type II: Tom was with the book in the bus driven from Town to University by Mary.

Name	First language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
1. Christine	**English**	car	Mary, Tom, book	Mary, Tom, book, car	yes	yes	yes	yes	yes	yes	yes	yes	yes
2. Janelle	**English**	car	Mary, Tom, book	book, car$^{1\,(C1)}$	yes	no$^{1\,(C1)}$	yes	no$^{1\,(C1)}$	yes	no$^{1\,(C1)}$	no$^{1\,(C1)}$	no$^{1\,(C1)}$	no$^{1\,(C1)}$
3. Qiu	**Chinese**	car	Mary, Tom, book	Mary, Tom, book, car	yes	yes	yes	yes	yes	yes	yes	yes	yes
4. Chairoj	**Thai**	car	Mary, Tom, book	Mary, Tom, car$^{2\,(C1)}$	yes	no$^{1\,(C1)}$	yes	no$^{1\,(C1)}$	yes	no$^{1\,(C1)}$	no$^{1\,(C1)}$	no$^{1\,(C1)}$	yes
5. Jiraporn	**Thai**	car	Mary, Tom, book	Mary, Tom, car$^{2\,(C1)}$	yes	yes	yes	yes	yes	yes	no$^{1\,(C1)}$	no$^{1\,(C1)}$	no$^{1\,(C1)}$
6. Taiyo	**Japanese**	car	Mary, Tom, book	Mary, Tom$^{3\,(C1)}$	yes	yes	yes	no$^{1\,(C1)}$	yes	no$^{1\,(C4)}$	no$^{1\,(C1)}$	no$^{1\,(C1)}$	no$^{1\,(C1)}$
7. Computer	-	car	Mary, Tom, book	Mary, Tom, book, car	yes	yes	yes	yes	yes	yes	yes	yes	yes

This part summarizes the experiment result of Type II sentence composed of twelve questions as follows:

Q1: What was driven?

Q2: What was in the car?

Q3: What traveled from Town to University?

Q4: Did the car carry Tom from Town to University?

Q5: Did the car move Tom from Town to University?

Q6: Did the car carry the book from Town to University?

Q7: Did the car move the book from Town to University?

Q8: Did Tom carry the book from Town to University?

Q9: Did Tom move the book from Town to University?

Q10: Did Mary carry the car from Town to University?

Q11: Did Mary carry the book from Town to University?

Q12: Did Mary carry Tom from Town to University?

From the poll, we found that all examinees' answers were unidirectional in Q1 and Q2, but it was unlike for Q3. Here, we could classify their Q3's keys into three types that is Janelle, Chairoj and Jiraporn, and Taiyo groups.

At the first group, Janelle's answer went along with her reason in Q3 of Type I. Because she relied on the difference between 'what' and 'who', so her reasoning process could be described as follows:

$$\underline{L(Tom, Book, Tom, Tom, \Lambda_t) \Pi L(Tom, Tom, Car, Car, \Lambda_t)}$$
$$\Pi L(Mary, Mary, Car, Car, \Lambda_t)\ \Pi L(Mary, Car, Town, Univ., \Lambda_t)$$
$$\Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.10)}$$

Applied MV

$$\rightarrow \underline{L(Tom, Book, Car, Car, \Lambda_t}) \Pi L(Mary, Mary, Car, Car, \Lambda_t)$$
$$\Pi L(Mary, Car, Town, Univ., \Lambda_t) \Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)} \qquad \text{(eq. B.11)}$$

Applied CC

$$\rightarrow L(Car, Book, Town, Univ., \Lambda_t) \Pi L(Mary, Mary, Car, Car, \Lambda_t)$$
$$\Pi L(Mary, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.12)}$$

Apply simplification law of Π

$$\rightarrow L(Car, Book, Town, Univ., \Lambda_t) \qquad \text{(eq. B.13)}$$

Due to her concept of “what”, $What(x) = L(\_, x, -human, -human, \Lambda_t)$, so she could form her original image as eq. B.10, after that applied a postulate rule to eq. B.10 the expression eq. B.11 – eq. B.13 were derived. Then, applied the simplification law of Π to eq. B.12, the last expectation eq. B.13 was got.

As well as Chairoj and Jiraporn, and Taiyo, their reasons were similar in Type I, that is “the book was held by Tom”, and “‘travel’ could be used for human agents only” respectively.

Proceeding with above reason, Chairoj’s and Jiraporn’s reasoning about “travel” should be as $L(Human, Human, Veh, Veh, \Lambda_t) \Pi L(Veh, Veh, p, p \neq, \Lambda_t)$, so when we applied this concept to the stimulus sentence, we could get as follow:

$$L(Tom, Book, Car, Car, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t)$$
$$-/\rightarrow L(Book, Book, Town, Univ., \Lambda_t) \qquad \text{(eq. B.14)}$$

Due to their concept, their reasoning processes that lead to answers could be seen in eq. B.16 – eq. B.17, where eq. B.15 shown their original image.

$$L(Tom, Book, Tom, Tom, \Lambda_t) \Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)}$$
$$\Pi L(Mary, Mary, Car, Car, \Lambda_t) \Pi L(Mary, Car, Town, Univ., \Lambda_t)$$

$$\Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)} \qquad \text{(eq. B.15)}$$

Applied MV

$$\begin{gathered}\rightarrow L(Tom, Book, Tom, Tom, \Lambda_t)\Pi \underline{\underline{L(Tom, Tom, Town, Univ., \Lambda_t)}} \\ \Pi L(Mary, Mary, Car, Car, \Lambda_t)\Pi \underline{\underline{L(Mary, Car, Town, Univ., \Lambda_t)}} \\ \underline{\underline{\Pi L(Car, Car, Town, Univ., \Lambda_t)}} \qquad \text{(eq. B.16)}\end{gathered}$$

$$-/\rightarrow L(Book, Book, Town, Univ., \Lambda_t) \qquad \text{(eq. B.17)}$$

Because these two persons applied MV to the underlined part in eq. B.15, so they got eq. B.16 that the double underlined parts here referred to “Tom”, “Mary” and “car” were going from Town to University. Anyway, they applied the postulate rule once, but not again, so their answers did not include “the book was going from Town to University” as eq. B.17.

Next is the last of Q3 problem with Taiyo, due to Taiyo’s “travel” definition was “$Travel(x) = L(x, x, p, p \neq, \Lambda_t)\Pi L(\_, x, human, human, \Lambda_t)$” or “$Travel(x) = L(Human, Human, p, p \neq, \Lambda_t)$” for short, so we could represent his image as follows:

$$\begin{gathered}L(Tom, Book, Tom, Tom, \Lambda_t)\Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \\ \Pi L(Mary, Mary, Car, Car, \Lambda_t)\Pi L(Mary, Car, Town, Univ., \Lambda_t) \\ \Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)} \qquad \text{(eq. B.18)}\end{gathered}$$

$$\rightarrow L(Mary, Mary, Town, Univ., \Lambda_t)\Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.19)}$$

To employ MV with the emphasized section of eq. B.18 and followed his own “travel” description, the target outcome eq. B.19 was happened. Afterward, please considered Q5 – Q12, in this portion we would divide into two sub-group, i.e. Q5 - Q9 and Q10 – Q12.

In the first batch, there were three people whose their words disagreed with others: Janelle, Chairoj, and Taiyo. From the observation, these three persons’ thinking behavior could be represented by ***L*** $\boldsymbol{L_{md}}$ as $Move(x, y) = L(x, y, y, y, \Lambda_t)\ \Pi\ L(y, y, p, p \neq, \Lambda_t)$, where $L(x, y, y, y, \Lambda_t)$ implied “x controls y’ s location”, i.e. x could control y to stop. Therefore, to interpret their original mental images into formal language could be narrated as eq. B.20:

$$\begin{gathered}L(Tom, Book, Tom, Tom, \Lambda_t)\Pi L(Tom, Tom, Car, Car, \Lambda_t) \\ \Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.20)}\end{gathered}$$

$$-/\rightarrow \boldsymbol{L(Car, Tom, Tom, Tom, \Lambda_t)} \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.21)}$$

$$-/\rightarrow \boldsymbol{L(Car, Book, Book, Book, \Lambda_t)} \Pi \underline{\boldsymbol{L(Book, Book, Town, Univ., \Lambda_t)}} \qquad \text{(eq. B.22)}$$

$$-/\rightarrow \boldsymbol{L(Tom, Book, Book, Book, \Lambda_t)} \Pi \underline{\boldsymbol{L(Book, Book, Town, Univ., \Lambda_t)}} \qquad \text{(eq. B.23)}$$

Since their personal explanation of "move" combined with the usage of postulate rules, Just only $L(Tom, Tom, Town, Univ., \Lambda_t)$ of eq. B.21 could be generated by MV, while the remaindered of eq. B.21 – eq. B.23 (bold letters) could not derive from any postulate rules, especially the underlined part, i.e. the term would be never happened because the book could not control its location.

However, "because Tom was in the car and car was moving from Town to University, so the car was able to move Tom from Town to University. Conversely, owing to the book was just placed in the bus, and it had no life, so the car and Tom didn't move the book from Town to University", Taiyo said. (For more understanding please see Fig. B.1)

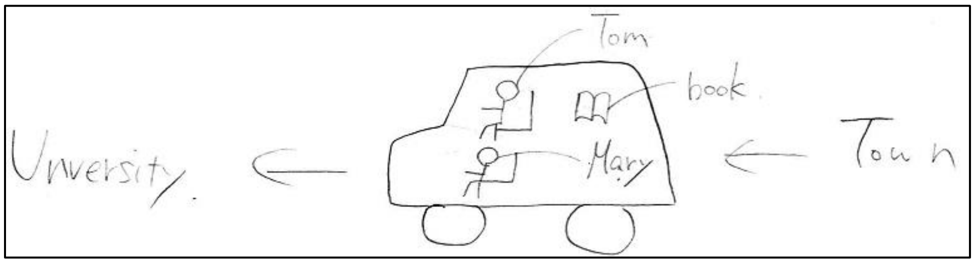


Fig. B.1. Mental image of Type II sentence, sketched by Taiyo.

Fig. B.1 shown Taiyo's imagery. He sketched this picture when we asked him for the experiment. Due to his image and interview (include other examinees) made us more understand each one intuitive learning process. Now consider the latter part, Q10 – Q12, the questions with the vocabulary of "carry". Here, the answers were similar in these three persons: Janelle, Jiraporn, and Taiyo. From the observation, we could summarize their "carry" definition as $Carry(x, y) = L(x, y, x, x, \Lambda_t) \Pi L(x, x, p, p \neq, \Lambda_t) \rightarrow L(x, y, p, p \neq, \Lambda_t) \Pi L(x, z, p, p \neq, \Lambda_t)$. Using this "carry" implication, we could represent their image as eq. B.24.

$$\begin{gathered}\underline{L(Mary, Mary, Car, Car, \Lambda_t)} \Pi L(Mary, Car, Town, Univ., \Lambda_t) \\ \Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)} \Pi L(Tom, Book, Tom, Tom, \Lambda_t) \\ \Pi L(Tom, Tom, Car, Car, \Lambda_t) \qquad \text{(eq. B.24)}\end{gathered}$$

$$-/\rightarrow \boldsymbol{L(Mary, Car, Mary, Mary, \Lambda_t)} \Pi L(Mary, Mary, Town, Univ., \Lambda_t) \qquad \text{(eq. B.25)}$$

$$-/\rightarrow \boldsymbol{L(Mary, Book, Mary, Mary,} \Lambda_t) \Pi L(Mary, Mary, Town, Univ., \Lambda_t) \qquad \text{(eq. B.26)}$$

$$-/\rightarrow \boldsymbol{L(Mary, Tom, Mary, Mary,} \Lambda_t) \Pi L(Mary, Mary, Town, Univ., \Lambda_t) \qquad \text{(eq. B.27)}$$

The non-italicized letters in formal languages eq. B.25 – eq. B.27 could be acquired from underlined terms in eq. B.24, while the italicizations were not happened from any kinds of postulate rule. Anyway, for more understanding we could summarize their justification that Mary was the driver, so she didn't carry Tom and the book by her hands, and the definition of 'carry' were different from "drive" in this case. That's why their answers to Q10 – Q12 were "No".

Just now, we gave precedence to the defenses of Janelle, Jiraporn, and Taiyo. Anyhow, Chairoj's opinion was quite interesting. Due to his answers to Q10 – Q12, we found that his meaning of "carry" was not similar to "drive" as same as those of Janelle, Jiraporn, and Taiyo, but the difference was due to his notions to Q11 and Q12. He gave the reason that "because Mary was the driver, so she couldn't hold the book by her hands. However, because Tom was a passenger, so it could imply that Mary carrired Tom from Town to University". Then, his definition of "carry" could be expressed as "$Carry(x, y) = L(x, y, p, p \neq, \Lambda_t) \Pi L(\_, z, y, y, \Lambda_t)$
$\rightarrow L(x, y, p, p \neq, \Lambda_t) \Pi L(x, z, p, p \neq, \Lambda_t)$". In accordance with (24), using Chiroj's ***Lmd*** of "carry", eq. B.28 – eq. B.30 could be derived as follows:

$$-/\rightarrow L(Mary, Mary, Town, Univ., \Lambda_t) \Pi \boldsymbol{L(Mary, Car, Town, Univ.,} \Lambda_t) \qquad \text{(eq. B.28)}$$

$$-/\rightarrow L(Mary, Mary, Town, Univ., \Lambda_t) \Pi \boldsymbol{L(Mary, Book, Town, Univ.,} \Lambda_t) \qquad \text{(eq. B.29)}$$

$$\rightarrow L(Mary, Mary, Town, Univ., \Lambda_t) \Pi L(Mary, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.30)}$$

Using MV to the underlined parts of eq. B.23, $L(Mary, Mary, Town, Univ., \Lambda_t)$, in eq. B.28 and eq. B.29 could be reached, but the italicized terms would not be got from any postulate rules. While the derivation of eq. B.29 could be shown in the following steps:

From eq. B.24, applied MV to underlined sections, then eq. B.31 was its return.

$$\rightarrow L(Mary, Mary, Town, Univ., \Lambda_t) \Pi \underline{L(Mary, Car, Town, Univ., \Lambda_t)}$$
$$\Pi L(Tom, Book, Tom, Tom, \Lambda_t) \Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \qquad \text{(eq. B.31)}$$

Applied CC to eq. B.31

$$\rightarrow L(Mary, Mary, Town, Univ., \Lambda_t) \Pi L(Mary, Tom, Town, Univ., \Lambda_t)$$

$$\Pi L(Tom, Book, Tom, Tom, \Lambda_t) \qquad \text{(eq. B.32)}$$

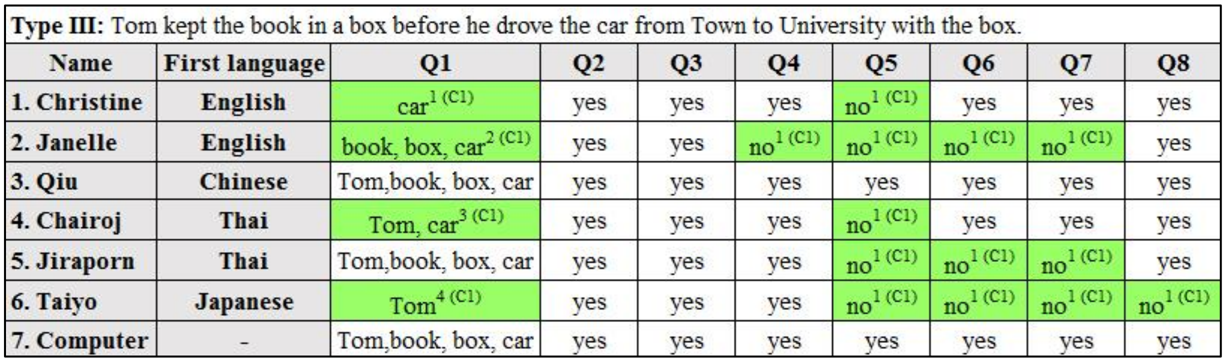
Applied simplification law of Π to eq. B.32, the target result, eq. B.33, was earned.

$$\rightarrow\ L(Mary, Mary, Town, Univ., \Lambda_t)\Pi L(Mary, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.33)}$$

B.4 Type III sentence

All above were the inspection of Type I and II, while below table indicated our experiment for another type of stimulus sentence that consisted of two simple sentence, i.e. Tom kept the book in a box before he drove the car from Town to University with the box.

Table B.3 The surveying of Type III stimulus sentence.



Type III: Tom kept the book in a box before he drove the car from Town to University with the box.

Name	First language	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
1. Christine	English	car$^{1\ (C1)}$	yes	yes	yes	no$^{1\ (C1)}$	yes	yes	yes
2. Janelle	English	book, box, car$^{2\ (C1)}$	yes	yes	no$^{1\ (C1)}$	no$^{1\ (C1)}$	no$^{1\ (C1)}$	no$^{1\ (C1)}$	yes
3. Qiu	Chinese	Tom,book, box, car	yes	yes	yes	yes	yes	yes	yes
4. Chairoj	Thai	Tom, car$^{3\ (C1)}$	yes	yes	yes	no$^{1\ (C1)}$	yes	yes	yes
5. Jiraporn	Thai	Tom,book, box, car	yes	yes	yes	no$^{1\ (C1)}$	no$^{1\ (C1)}$	no$^{1\ (C1)}$	yes
6. Taiyo	Japanese	Tom$^{4\ (C1)}$	yes	yes	yes	no$^{1\ (C1)}$	no$^{1\ (C1)}$	no$^{1\ (C1)}$	no$^{1\ (C1)}$
7. Computer	-	Tom,book, box, car	yes	yes	yes	yes	yes	yes	yes

Within this Type III sentence, the examinee would answer the following questions:

Q1: What traveled from Town to University?

Q2: Did the car carry Tom from Town to University?

Q3: Did the car carry the box from Town to University?

Q4: Did the car carry the book from Town to University?

Q5: Did Tom carry the car from Town to University?

Q6: Did Tom carry the box from Town to University?

Q7: Did Tom carry the book from Town to University?

Q8: Did the box carry the book from Town to University?

In Table B.3, we could separate the answers of Q1 into four groups. The first one was Christine's answer, then followed by Janelle, Chairoj, and Taiyo' answers serially. The expression eq. B.34 was duplicated Christine's image put together with her "travel" definition, $Travel(x) = L(x, x, p, p \neq, \Lambda_t)$, and no use of postulate rules. So it leaded to her answer at Q1 of this Type III sentence.

$$L(Tom, Book, Box, Box) \bullet L(Tom, Box, Car, Car)$$
$$\Pi L(Car, Car, Town, Univ., \Lambda_t) \quad \text{(eq. B.34)}$$

Similar with the same reason in Type I and II, Janelle was a loyal person on the difference between "what" and "who". Therefore her answers never included human when the questions were started with "what", and eq. B.35 shows her imagery in logical form.

$$L(Tom, Book, Box, Box) \bullet L(Tom, Box, Car, Car)$$
$$\Pi L(Car, Car, Town, Univ., \Lambda_t) \Pi L(Tom, Book, Box, Box)$$
$$\rightarrow L(Car, Box, Town, Univ., \Lambda_t) \Pi L(Car, Book, Town, Univ., \Lambda_t) \quad \text{(eq. B.35)}$$

As same as Christine's "travel" definition with no postulate rules, Chairoj applied it with his own belief that "Tom was the driver". So his thinking process could be described as:

$$L(Tom, Book, Box, Box) \bullet L(Tom, Book, Box, Box)$$
$$\Pi L(Tom, Tom, Car, Car, \Lambda_t) \Pi L(Tom, Car, Town, Univ., \Lambda_t) \quad \text{(eq. B.36)}$$

While Taiyo's reason in this Q1 was also followed by his logic as we had narrated in the earlier contents. Then we could reflex his mental image as eq. B.37.

$$L(Tom, Book, Box, Box) \bullet L(Tom, Box, Car, Car)$$
$$\underline{\Pi L(Tom, Tom, Car, Car, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t)}$$
$$\rightarrow L(Tom, Tom, Town, Univ., a) \quad \text{(eq. B.37)}$$

Because Taiyo selected to apply MV to his logic, the final answer, $L(Tom, Tom, Town, Univ., \Lambda_t)$ was gained. Furthermore, if noticed, we could see two kinds of sign in previous $\boldsymbol{L_{md}}$ in Type I - III. The "•"sign stood for "Consecutive AND (CAND)", while "Π" implied "Simultaneous AND" or "SAND". Because Type III stimulus sentence consisted of two events happened continuously, so we applied these signs to specify the circumstances.

Besides the disparity in Q1, if we considered Q3 – Q8 that contained the vocabulary "carry", we could group these diversities into four groups: Christine and Chairoj, Janelle, Jiraporn, and Taiyo group respectively. If we deliberated Christine and Chairoj group, we could found that their answers were "Yes" except in Q5. From the interview, they said that "the meaning of 'carry' was different from 'drive'", so we could expound their definition of "carry" and "drive" as follows:

$$Carry(x, y) = L(\_, y, x, x, \Lambda_t) \Pi L(\_, x, p, p \neq, \Lambda_t) \quad \text{(eq. B.38)}$$

$$Drive(x, y) = L(x, x, y, y, \Lambda_t) \Pi L(x, y, p, p \neq, \Lambda_t) \Pi L(y, y, p, p \neq, \Lambda_t) \quad \text{(eq. B.39)}$$

Along with eq. B.37, therefore Christine and Chairoj' image could be expressed as eq. B.40.

$$L(Tom, Book, Box, Box, \Lambda_t) \bullet (L(Tom, Box, Tom, Tom, \Lambda_t)$$
$$\Pi L(Tom, Tom, Car, Car, \Lambda_t) \Pi L(Tom, Car, Town, Univ., \Lambda_t)$$
$$\Pi L(Tom, Book, Box, Box, \Lambda_t) \Pi\ L(Car, Car, Town, Univ., \Lambda_t)) \qquad \text{(eq. B.40)}$$

Anyway eq. B.40 was just only their logical form of the stimulus sentence, without postulate rule it did not lead to the answers in Q3 – Q8. So we could reveal their answers as the following expressions.

Q3: From eq. B.40, we revised it by underlining some terms as shown in eq. B.40', after that applied MV to the underlined parts. Then employed CC to selection parts of eq. B.41, and finally with simplification law of • and Π, the final answer eq. B.42 was gained.

$$L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)}$$
$$\Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \Pi L(Tom, Car, Town, Univ., \Lambda_t)$$
$$\Pi L(Tom, Book, Box, Box, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t)) \qquad \text{(eq. B.40')}$$

$$\rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Car, Car, \Lambda_t)}$$
$$\Pi L(Tom, Car, Town, Univ., \Lambda_t) \Pi L(Tom, Book, Box, Box, \Lambda_t)$$
$$\Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \qquad \text{(eq. B.41)}$$

$$\rightarrow L(Car, Box, Car, Car, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.42)}$$

Please note that from the expression eq. B.43 we would like to use the abbreviations "MV", "CC", and "SL" after each expression to indicate that each one could be obtained by postulate of matters as values, postulate of shortcut in causal chain, and simplification law of • or Π respectively.

For Q4, the steps of $\boldsymbol{L_{md}}$ can be presented as the following:

$$L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)}$$
$$\Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \Pi L(Tom, Car, Town, Univ., \Lambda_t)$$
$$\Pi \underline{L(Tom, Book, Box, Box, \Lambda_t)} \Pi L(Car, Car, Town, Univ., \Lambda_t)) \qquad \text{(eq. B.43)}$$

Applied MV to eq. B.43

$$\rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Book, Car, Car, \Lambda_t)}$$
$$\Pi L(Tom, Car, Town, Univ., \Lambda_t) \Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \qquad \text{(eq. B.44)}$$

Applied CC and SL to eq. B.44

$$\rightarrow L(Car, Book, Car, Car, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.45)}$$

In Q5, they did not apply any postulate rule, therefore eq. B.46 would be never got, and they returned ‘No’ as their answers. Please remark that because of “no use of Postulate”, therefore, the bold letters with underline meant that the term would be never obtained.

$$-/\rightarrow \underline{\boldsymbol{L(Tom, Car, Tom, Tom, \Lambda_t)}} \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B. 46)}$$

For eq. B. 47 – eq. B. 54 are steps of ***L***$\boldsymbol{L_{md}}$ of Q6 – Q8.

Q6:

$$\begin{gathered} L(Tom, Book, Box, Box, \Lambda_t) \bullet (L(Tom, Box, Tom, Tom, \Lambda_t) \\ \Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \Pi L(Tom, Car, Town, Univ., \Lambda_t) \\ \Pi L(Tom, Book, Box, Box, \Lambda_t) \Pi \underline{L(Car, Car, Town, Univ., \Lambda_t))} \end{gathered} \qquad \text{(eq. B.47)}$$

Applied MV to eq. B.47

$$\begin{gathered} \rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)} \\ \Pi \underline{L(Tom, Tom, Town, Univ., \Lambda_t)} \Pi L(Tom, Car, Town, Univ., \Lambda_t) \\ \Pi L(Tom, Book, Box, Box, \Lambda_t)) \end{gathered} \qquad \text{(eq. B.48)}$$

Applied SL to eq. B.48

$$\rightarrow L(Tom, Box, Tom, Tom, \Lambda_t) \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.49)}$$

Q7:

$$\begin{gathered} L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)} \\ \Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \Pi L(Tom, Car, Town, Univ., \Lambda_t) \\ \underline{\Pi L(Tom, Book, Box, Box, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t))} \end{gathered} \qquad \text{(eq. B.50)}$$

Applied MV twice and SL to eq. B.50

$$\rightarrow L(Tom, Book, Tom, Tom, \Lambda_t) \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.51)}$$

Q8:

$$L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t})$$
$$\Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t})\Pi L(Tom, Car, Town, Univ., \Lambda_t)$$
$$\Pi L(Tom, Book, Box, Box, \Lambda_t)\Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \qquad \text{(eq. B.52)}$$

Applied MV twice to eq. B.52

$$\rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Town, Univ., \Lambda_t})$$
$$\Pi L(Tom, Car, Town, Univ., \Lambda_t)\Pi \underline{L(Tom, Book, Box, Box, \Lambda_t)}) \qquad \text{(eq. B.53)}$$

Applied SL to eq. B. 53

$$\rightarrow L(\_, Book, Box, Box, \Lambda_t)\Pi L(\_, Box, Town, Univ., \Lambda_t) \qquad \text{(eq. B.54)}$$

Above was Christine's and Chairoj's cases for Q3 – Q8 in Type III. Afterward we liked to continue with Janelle, Jiraporn, and Taiyo' problems, their original images were also same with Christine and Chairoj in eq. B.40, but their answers (Q5– Q7) did not apply any postulate rules. Moreover they said that the box (with the book inside) was just placed in the car, so Tom didn't carry the box and the book directly. So their decisions were "No" in Q5 – Q7 items.

The following we would like to show you about their reasoning process by starting from Janelle's method, after that followed by Jiraporn, and Taiyo respectively.

[Janelle's case]:

Q3:

$$L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t})$$
$$\Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t})\Pi L(Tom, Car, Town, Univ., \Lambda_t)$$
$$\Pi L(Tom, Book, Box, Box, \Lambda_t)\Pi L(Car, Car, Town, Univ., \Lambda_t)) \qquad \text{(eq. B.55)}$$

Applied MV to eq. B.55

$$\rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Car, Car, \Lambda_t})$$
$$\Pi L(Tom, Car, Town, Univ., \Lambda_t)\Pi(Tom, Book, Box, Box, \Lambda_t)$$
$$\Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \qquad \text{(eq. B.56)}$$

Applied CC and SL to eq. B.56

$$\rightarrow L(Car, Box, Car, Car, \Lambda_t)\Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.57)}$$

For Q4 – Q7, the bold letters with underline meant that the term would be never obtained because of no use of postulate.

$$-/\!\to \underline{\boldsymbol{L(Car, Book, Car, Car}, \Lambda_{\boldsymbol{t}})}\Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.58)}$$

$$-/\!\to \underline{\boldsymbol{L(Tom, Car, Tom, Tom}, \Lambda_{\boldsymbol{t}})}\Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.59)}$$

$$-/\!\to \underline{\boldsymbol{L(Tom, Box, Tom, Tom}, \Lambda_{\boldsymbol{t}})}\Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.60)}$$

$$-/\!\to \underline{\boldsymbol{L(Tom, Book, Tom, Tom}, \Lambda_{\boldsymbol{t}})}\Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.61)}$$

Q8:

$$\begin{gathered} L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)} \\ \Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)}\Pi L(Tom, Car, Town, Univ., \Lambda_t) \\ \Pi L(Tom, Book, Box, Box, \Lambda_t)\Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \end{gathered} \qquad \text{(eq. B.62)}$$

Applied MV twice to eq. B.62

$$\begin{gathered} \to L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Town, Univ., \Lambda_t)} \\ \Pi L(Tom, Car, Town, Univ., \Lambda_t)\Pi \underline{L(Tom, Book, Box, Box, \Lambda_t)}) \end{gathered} \qquad \text{(eq. B.63)}$$

Applied SL to eq. B.63

$$\to L(\_, Book, Box, Box, \Lambda_t)\Pi L(\_, Box, Town, Univ., \Lambda_t) \qquad \text{(eq. B.64)}$$

[Jiraporn's case]:

Q3:

$$\begin{gathered} L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)} \\ \Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)}\Pi L(Tom, Car, Town, Univ., \Lambda_t) \\ \Pi L(Tom, Book, Box, Box, \Lambda_t)\Pi L(Car, Car, Town, Univ., \Lambda_t)) \end{gathered} \qquad \text{(eq. B.65)}$$

Applied MV to eq. B.65

$$\begin{gathered} \to L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Car, Car, \Lambda_t)} \\ \Pi L(Tom, Car, Town, Univ., \Lambda_t)\Pi L(Tom, Book, Box, Box, \Lambda_t) \\ \Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \end{gathered} \qquad \text{(eq. B.66)}$$

Applied CC and SL to eq. B.66

$$\rightarrow L(Car, Box, Car, Car, \Lambda_t)\mathit{\Pi} L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.67)}$$

Q4:

$$\begin{gathered} L(Tom, Book, Box, Box, \Lambda_t) \bullet(\underline{L(Tom, Box, Tom, Tom, \Lambda_t)} \\ \mathit{\Pi}\underline{L(Tom, Tom, Car, Car, \Lambda_t)}\mathit{\Pi} L(Tom, Car, Town, Univ., \Lambda_t) \\ \mathit{\Pi}\underline{L(Tom, Book, Box, Box, \Lambda_t)}\mathit{\Pi} L(Car, Car, Town, Univ., \Lambda_t)) \end{gathered} \qquad \text{(eq. B.68)}$$

Applied MV to eq. B.68

$$\begin{gathered} \rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet(\underline{L(Tom, Book, Car, Car, \Lambda_t)} \\ \mathit{\Pi} L(Tom, Car, Town, Univ., \Lambda_t)\mathit{\Pi}\underline{L(Car, Car, Town, Univ., \Lambda_t)}) \end{gathered} \qquad \text{(eq. B.69)}$$

Applied CC and SL to eq. B.69

$$\rightarrow L(Car, Book, Car, Car, \Lambda_t)\mathit{\Pi} L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.70)}$$

For Q5 – Q7, because of no use of postulate, the bold with underline terms will not be gained.

$$-/\!\!\rightarrow \underline{\boldsymbol{L(Tom, Car, Tom, Tom, \Lambda_t)}}\mathit{\Pi} L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.71)}$$

$$-/\!\!\rightarrow \underline{\boldsymbol{L(Tom, Box, Tom, Tom, \Lambda_t)}}\mathit{\Pi} L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.72)}$$

$$-/\!\!\rightarrow \underline{\boldsymbol{L(Tom, Book, Tom, Tom, \Lambda_t)}}\mathit{\Pi} L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.73)}$$

Q8:

$$\begin{gathered} L(Tom, Book, Box, Box, \Lambda_t) \bullet(\underline{L(Tom, Box, Tom, Tom, \Lambda_t)} \\ \mathit{\Pi}\underline{L(Tom, Tom, Car, Car, \Lambda_t)}\mathit{\Pi} L(Tom, Car, Town, Univ., \Lambda_t) \\ \mathit{\Pi} L(Tom, Book, Box, Box, \Lambda_t)\mathit{\Pi}\underline{L(Car, Car, Town, Univ., \Lambda_t)}) \end{gathered} \qquad \text{(eq. B.74)}$$

Applied MV twice to eq. B.74

$$\begin{gathered} \rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet(\underline{L(Tom, Box, Town, Univ., \Lambda_t)} \\ \mathit{\Pi} L(Tom, Car, Town, Univ., \Lambda_t)\mathit{\Pi}\underline{L(Tom, Book, Box, Box, \Lambda_t)}) \end{gathered} \qquad \text{(eq. B.75)}$$

Applied SL to eq. B.75

$$\rightarrow L(\_, Book, Box, Box, \Lambda_t)\mathit{\Pi} L(\_, Box, Town, Univ., \Lambda_t) \qquad \text{(eq. B.76)}$$

[Taiyo’s case]:

Q3:

$$L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)}$$
$$\Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \Pi L(Tom, Car, Town, Univ., \Lambda_t)$$
$$\Pi L(Tom, Book, Box, Box, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t)) \qquad \text{(eq. B.77)}$$

Applied MV to eq. B.77

$$\rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Car, Car, \Lambda_t)}$$
$$\Pi L(Tom, Car, Town, Univ., \Lambda_t) \Pi L(Tom, Book, Box, Box, \Lambda_t)$$
$$\Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \qquad \text{(eq. B.78)}$$

Applied CC and SL to eq. B.78

$$\rightarrow L(Car, Box, Car, Car, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.79)}$$

Q4:

$$L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Box, Tom, Tom, \Lambda_t)}$$
$$\Pi \underline{L(Tom, Tom, Car, Car, \Lambda_t)} \Pi L(Tom, Car, Town, Univ., \Lambda_t)$$
$$\Pi \underline{L(Tom, Book, Box, Box, \Lambda_t)} \Pi L(Car, Car, Town, Univ., \Lambda_t)) \qquad \text{(eq. B.80)}$$

Applied MV to eq. B.80

$$\rightarrow L(Tom, Book, Box, Box, \Lambda_t) \bullet (\underline{L(Tom, Book, Car, Car, \Lambda_t)}$$
$$\Pi L(Tom, Car, Town, Univ., \Lambda_t) \Pi \underline{L(Car, Car, Town, Univ., \Lambda_t)}) \qquad \text{(eq. B.81)}$$

Applied CC and SL to eq. B.81

$$\rightarrow L(Car, Book, Car, Car, \Lambda_t) \Pi L(Car, Car, Town, Univ., \Lambda_t) \qquad \text{(eq. B.82)}$$

For Q5 – Q7, because of no use of postulate, the following equations (eq. B. 83 – eq. B.86), therefore, bold with underline terms will not be gained.

$$-/\rightarrow \underline{\boldsymbol{L(Tom, Car, Tom, Tom, \Lambda_t)}} \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.83)}$$

$$-/\rightarrow \underline{\boldsymbol{L(Tom, Box, Tom, Tom, \Lambda_t)}} \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.84)}$$

$$-/\rightarrow \underline{\boldsymbol{L(Tom, Book, Tom, Tom, \Lambda_t)}} \Pi L(Tom, Tom, Town, Univ., \Lambda_t) \qquad \text{(eq. B.85)}$$

$$-/\rightarrow L(\_, Book, Box, Box, \Lambda_t) \Pi L(\_, Box, Town, Univ., \Lambda_t) \qquad \text{(eq. B.86)}$$

This work aimed to approve the ability of Mental Image Directed Semantic Theory (MIDST) in order to use in real condition. In the test, we received the cooperation from six persons, Christine and Janelle (American), Qiu (Chinese), Chairoj and Jiraporn (Thai), and Taiyo (Japanese). To collect the data, we designed the psycholinguistic experiment which contained three parts, in each part composed of an example (stimulus) sentence and sketch area for draw out their image, after that they answered the questions of that stimulus sentence. Please note that all words in our computer application were based on Merriam - Webster Dictionary. Anyway, when we cited to other related works, we found that there were none or few works that tried to affirm their hypothesis or theories were workable in real situation.

From the experiment, we could say that nobody gave the wrong answers, because everyone had his/her own reason up to outside environments. So their understanding processes might vary person by person as we could see. In this experiment, we discovered many unexpected things as follows:

1. Most of the examinees' vocabularies did not depend on the meaning in English dictionary, but the most influence in each person was their native language. For example:

    - "Travel could be used for people only", Taiyo said. As his testimony, we could confirm that the word 'travel' could employ to human only in Japanese.
    - As same as Christine case, when we analyzed her answers, we found that her answer quite unique in some questions. So we interviewed her and know that although she is a native speaker, but she told us that her mother spoke Filipino to her sometime. So we could assume that Filipino language had an influence on her decision.
    - For Janelle, she was the person who really relied on grammatical knowledge of Wh-question. Her answers to 'what' questions never included people, because she believed that 'what' and 'who' did not relate to each other.
    - Chairoj and Jiraporn, although they are Thai people, we found that their thinking were not similar. We noticed that Chairoj's definition of 'carry' and 'move' are different, but it was not happened in Jiraporn case. While Qiu, an only Chinese examinee, her answer was the most resemblant to Computer's returns.

2. Moreover we found that most of people though that the definition of "carry" was different from "drive". They gave the meaning of "carry" as "Holding something (and changing its location) by people' hands", while "drive" referred to "Controlling a vehicle from one place to another place". Anyhow, when we looked for the meaning of these two words in the dictionary, we could conclude that. "drive was a subclass of carry", but in people sense, "drive" was absolutely different from

"carry". Not only the meaning of "carry" and "drive", but the definition between "move" and "carry" were also different as we could found in previous details.

3. Due to their sketches, we found that not everyone would interpret the stimulus sentence in the same way. As you could see, Fig. B.1 shown Taiyo's paraphrase in Type II sentence, while other ones could transform the input sentences in the same way.

4. In addition, we found that Janelle, Jiraporn, Taiyo, and Chairoj (some cases) did not employ postulate rules. They returned the answers depended on the pictures they had imaged, as we could see in some answers for example: 'Tom did not carry the box/book' in Type III, or 'the book did not travel from Town to University' in Type II. Here we could summarize that these people seldom applied postulate rules, they selected to believe something what they had seen. Therefore, Tom did not carry the box/book although he was the driver in Type III. Moreover, because the book was held by Tom, and it has no life, so the book should not travel in Type II (as same as in Type I and III as well).

APPENDIX C

This section presents semantic definitions of words each of which here is limited to temporal change events.

C.1 List of words in CMS

Nouns and Pronouns

Table C.1 List of nouns and pronouns

Word	L_{md}
shop_keeper	$+L[shop\_keeper, shop\_keeper, p, q, Physical\ location]$
bank_clerk	$+L[bank\_clerk, bank\_clerk, p, q, Physical\ location]$
Tom	$+L[tom, tom, p, q, Physical\ location]$
Mary	$+L[mary, mary, p, q, Physical\ location]$
he	$+L[he, he, p, q, Physical\ location]$
she	$+L[she, she, p, q, Physical\ location]$
I	$+L[i, i, p, q, Physical\ location]$
me	$+L[me, me, p, q, Physical\ location]$
you	$+L[you, you, p, q, Physical\ location]$
it	$+L[it, it, p, q, Physical\ location]$
everything	$+L[everything, everything, p, q, Physical\ location]$
phone	$-L[phone, phone, p, q, Physical\ location]$
book	$-L[book, book, p, q, Physical\ location]$
box	$-L[box, box, p, q, Physical\ location]$
bus	$+L[bus, bus, p, q, Physical\ location]$
car	$+L[car, car, p, q, Physical\ location]$
flower_shop	$-L[flower\_shop, flower\_shop, p, q, Physical\ location]$
drug_store	$-L[drug\_store, drug\_store, p, q, Physical\ location]$
hospital	$-L[hospital, hospital, p, q, Physical\ location]$

Word	L_{md}
post_office	$-L[post\_office, post\_office, p, q, Physical\ location]$
swimming_pool	$-L[swimming\_pool, swimming\_pool, p, q, Physical\ location]$
playground	$-L[playground, playground, p, q, Physical\ location]$
super_market	$-L[super\_market, super\_market, p, q, Physical\ location]$
pet_shop	$-L[pet\_shop, pet\_shop, p, q, Physical\ location]$
bank	$-L[bank, bank, p, q, Physical\ location]$
home	$-L[home, home, p, q, Physical\ location]$
house	$-L[house, house, p, q, Physical\ location]$
apartment	$-L[apartment, apartment, p, q, Physical\ location]$
restaurant	$-L[restaurant, restaurant, p, q, Physical\ location]$
zoo	$-L[zoo, zoo, p, q, Physical\ location]$
yard	$-L[yard, yard, p, q, Physical\ location]$
bridge	$-L[bridge, bridge, p, q, Physical\ location]$
stair	$-L[stair, stair, p, q, Physical\ location]$
flower_bed	$-L[flower\_bed, flower\_bed, p, q, Physical\ location]$
fountain	$-L[fountain, fountain, p, q, Physical\ location]$
road	$-L[road, road, p, q, Physical\ location]$
street	$-L[street, street, p, q, Physical\ location]$
river	$-L[river, river, p, q, Physical\ location]$
town	$-L[town, town, p, q, Physical\ location]$
university	$-L[university, university, p, q, Physical\ location]$
school	$-L[school, school, p, q, Physical\ location]$
TV	$-L[tv, tv, p, q, Physical\ location]$
bed	$-L[bed, bed, p, q, Physical\ location]$
table	$-L[table, table, p, q, Physical\ location]$
chair	$-L[chair, chair, p, q, Physical\ location]$

Word	L_{md}
refrigerator	$-L[refrigerator, refrigerator, p, q, Physical\ location]$
sofa	$-L[sofa, sofa, p, q, Physical\ location]$
paravent	$-L[paravent, paravent, p, q, Physical\ location]$
computer	$-L[computer, computer, p, q, Physical\ location]$
computer_chair	$-L[computer\_chair, computer\_chaie, p, q, Physical\ location]$
closet	$-L[closet, closet, p, q, Physical\ location]$
vest	$-L[vest, vest, p, q, Physical\ location]$
stove	$-L[stove, stove, p, q, Physical\ location]$
book_shelf	$-L[book\_shelf, book\_shelf, p, q, Physical\ location]$
tulip	$-L[tulip, tulip, p, q, Physical\ location]$
carnation	$-L[canation, canation, p, q, Physical\ location]$
paracetamol	$-L[paracetamol, paracetamol, p, q, Physical\ location]$
letter	$-L[letter, letter, p, q, Physical\ location]$
voice	$-L[voice, voice, p, q, Physical\ location]$
money	$-L[money, money, p, q, Physical\ location]$
medicine	$-L[medicine, medicine, p, q, Physical\ location]$
drug	$-L[drug, drug, p, q, Physical\ location]$
goldfish	$+L[goldfish, goldfish, p, q, Physical\ location]$
fish	$+L[fish, fish, p, q, Physical\ location]$
dog	$+L[dog, dog, p, q, Physical\ location]$
elephant	$+L[elephant, elephant, p, q, Physical\ location]$
water	$+L[water, water, p, q, Physical\ location]$
who	$+L[x, x, who, who, Physical\ location?]$
what	$+L[x, x, what, what, Physical\ location?]$
where	$+L[x, x, where, where, Physical\ location?]$
when	$+L[x, x, when, when, Physical\ location?]$

Word	L_{md}
how	$+(L[how, x, p, q, Physical\ location?]$ $\Pi L[x, x, how, how, Physical\ location?])$

Verbs

Table C.2 List of verbs*

Word	L_{md}
be	$+L[be, be, p, q, Physical\ location]$
do	$+L[do, do, p, q, Physical\ location]$
can	$+L[can, can, p, q, Physical\ location]$
run	$+L[runner, runner, p, q, Physical\ location]$
travel	$+L[traveler, traveler, p, q, Physical\ location]$
go	$+L[go, go, p, q, Physical\ location]$
love	$+(L[x, beloved, p, q, happiness]\Pi L[beloved, x, p, q, happiness])$
like	$+L[liked, liked, p, q, happiness]$
paint	$+L[x, paint, z, z, color]$
buy	$+(L[[seller, buyer], z, seller, buyer, Physical\ location]$ $\Pi L[[seller, buyer], money, buyer, seller, Physical\ location])$
give	$+L[x, z, x, give(= receiver), Physical\ location]$
withdraw	$+L[[x, withdrawer], money, x, withdrawer, Physical\ location]$
call	$+(L[caller, phone, caller, caller, Physical\ location]$ $\Pi L[caller, voice, caller, receiver, Physical\ location])$
carry_1	$+(L[carrier, carried, carrier, carrier, Physical\ location]$ $\Pi L[carrier, carrier, p, q, Physical\ location])$
carry_2	$+(L[carrier, carrier, p, q, Physical\ location]$ $\Pi L[carrier, carried, p, q, Physical\ location])$
drive	$+(L[driver, driver, vehicle, vehicle, Physical\ location]$ $\Pi L[driver, vehicle, p, q, Physical\ location])$
move	$+L[mover, moved, p, q, Physical\ location]$
keep	$+L[keeper, kept, p, q, Physical\ location]$
want	$+(L[desirer, wanted, desirer, desirer, Physical\ location]$ $\Pi L[wanted, desirer, p, q, happiness])$

Word	L_{md}
take	$+L[taker, taken, p, q, Physical\ location]$
return	$+(L[x, x, return, q, Physical\ location]$ $\Pi L[x, x, q, return, Physical\ location])$
fetch	$+(L[x, x, fetch, q, Physical\ location]$ $\Pi L[x, x, q, fetch, Physical\ location])$

* This table, for ease to understand, the author had used words instead of mathematical variables such as "x", "y", and "z".

Prepositions

Table C.3 List of prepositions

Word	L_{md}
with	$+L[x, with, x, x, Physical\ location]$
at	$+L[x, x, at, at, Physical\ location]$
in	$+L[x, x, in, in, Physical\ location]$
on	$+L[x, x, on, on, Physical\ location]$
by	$+L[by, y, p, q, Physical\ location]$
to	$+L[x, x, p, to, Physical\ location]$
from	$+L[x, x, from, q, Physical\ location]$
along	$+L[x, x, along, along, direction]$
north	$+L[x, x, north, north, direction]$
south	$+L[x, x, south, south, direction]$

Adverbs and Adjectives

Table C.4 List of adverbs and adjectives

Word	$\boldsymbol{L_{md}}$
please	$+L[x, x, pleased, pleased, happiness]$
fine	$+L[x, x, fine, fine, happiness]$
good	$+L[x, x, good, good, happiness]$
red	$+L[x, x, red, red, color]$
gold	$+L[x, x, gold, gold, color]$

APPENDIX D

This section presents program codes of this work that consist of:

- main.py
- setpositionofplaces.py
- connectionposition.py
- mindistance.py
- my_dictionary.py
- extractphrase.py
- inference.py
- move_anna_taro.py
- nlexpression_to_lmd.py
- call_phone.py
- home_placeposition.py
- home_connectionposition.py
- closest_place.py
- closepoint.py
- sex_detection.py
- question_answer.py
- pattern_matching.py

D.1 main.py

```
import turtle
import nltk
import random
import copy
import time
from turtle import *
from PIL import Image
from random import randint

from setpositionofplaces import SetPositionofPlaces
from connectionposition import ConnectionPosition
from mindistance import MinDistance
from my_dictionary import my_dictionary
from extractphrase import ExtractPhrases
from inference import Inference
from move_anna_taro import MoveAnnaTaro
from nlexpression_to_lmd import NLExpressionToLMD
from call_phone import CallPhone
from home_placeposition import HomePlacePosition
from home_connectionposition import HomeConnectionPosition
from closest_place import ClosePlace
from closepoint import ClosePoint
from sex_detection import SexDetection
from question_answer import QuestionAnswering
from pattern_matching import PatternMatching

#===== Load background picture =====
start = time.time()
screen = turtle.getscreen()
screen.setup(1200,600)
screen.bgpic("Townmap.gif")
screen.title("Welcome to Anna world")

#===== Set position of the shops and places =====
place_position = {}
place_position = SetPositionofPlaces()
connection_position = {}
connection_position = ConnectionPosition()
close_point = {}
close_point = ClosePoint()


#===== Create Anna, the home robot =====

anna_shape = 'Anna.gif'
screen.addshape(anna_shape)
anna = turtle.Turtle()
anna.penup()
anna.goto(-80.00,120.00)
anna.shape(anna_shape)
anna.hideturtle()

#===== Create Taro, the old man =====
taro_shape = 'Taro.gif'
screen.addshape(taro_shape)
taro = turtle.Turtle()
taro.penup()
taro.goto(-80.00,120.00)
taro.shape(taro_shape)
taro.hideturtle()
```

```
#===== Create Anna_Taro, for moving together =====
anna_taro_shape = 'Anna_Taro.gif'
screen.addshape(anna_taro_shape)
anna_taro =  turtle.Turtle()
anna_taro.penup()
anna_taro.goto(-80.00,120.00)
anna_taro.shape(anna_taro_shape)

#===== Create Phone turtle =====
telephone_shape = 'Phone.gif'
screen.addshape(telephone_shape)
telephone = turtle.Turtle()
telephone.penup()
telephone.goto(130.00, 160.00)
telephone.shape(telephone_shape)
telephone.hideturtle()

#===== Create Anna_Phone turtle =====
anna_phone_shape = 'Anna_Phone.gif'
screen.addshape(anna_phone_shape)
anna_phone = turtle.Turtle()
anna_phone.penup()
anna_phone.goto(130.00, 160.00)
anna_phone.shape(anna_phone_shape)
anna_phone.hideturtle()

#===== Create Tom_Mary turtle =====
tom_mary_shape = 'tom_mary.gif'
screen.addshape(tom_mary_shape)
tom_mary = turtle.Turtle()
tom_mary.penup()
tom_mary.goto(-225.00,40.00)
tom_mary.shape(tom_mary_shape)
tom_mary.hideturtle()
end = time.time()
print ("\n\t\033[0;33m>>> Time-consuming map loading = %.3f seconds.\033[0;m" % (end
 - start))



print ("************* Welcome to Anna's world **************")
print ("*** To exit the program, please press 'q' or 'Q' ***\n")
text = raw_input("\nYour command: ")
last_locus = ''
while (text != 'Q') and (text != 'q'):

    #===============
=====================================================================================
===============

    start = time.time()

    #=== Call my_dictionary function ===
    sentence = my_dictionary(text)
    sentence_new = sentence[0]
    cj_w = sentence[1]   # cj_w = conjunction_word
    A_1 = sentence[2]    # A_1 = Raw input sentence
    A_2 = sentence[3]    # A_2 = Word types from A_1
    remain_word = sentence[4]    # remain_word = Cut off useless words in A_1
    remain_all_DT = sentence[5] # remain_all_DT = word types of 'remain_word'
    index_of_keyword = sentence[6]  # index_of_keyword = index of words in A_1
    remain_phrase = sentence[7] # remain_phrase = the phrases the are extracted from
input sentence
    remain_DT = sentence[8]     # remain_DT = word types of phrases in 'ramain_phrase
'

    end = time.time()
    print ("\n\t\033[0;33m>>> Time-consuming dependency tree = %.3f seconds.\033[0;m
" % (end - start))
    #===== End of calling my_dictionary function =====

    #===== If 'When' is the first word of input sentence, but it does not act for
question sentence. =====
    start = time.time()
    for w in sentence_new:
        if ('when' in w[0]) and ('PN' in w[1]):     # In case of "When he...  (Not
question, but affirmative sentence)"
            w.pop(0)
            for ww in w:
                if (',' in ww):
                    i = w.index(ww)
                if (('do' in ww) or ('does' in ww) or ('will' in ww)) and (w.index(ww
) == i+1):
                    sentence_new.append(w[:i])
                    sentence_new.append(w[i+1:])
                    sentence_new.remove(w)
    #===== End of the process =====


    for w in remain_phrase:
        if (w[0] == ['when']) and ((w[1] != ['do']) or (w[1] != ['does']) or (w[1] !=
 ['will'])):
            w.pop(0)
            for ww in w:
                if (',' in ww):
                    i = w.index(ww)
                if (('do' in ww) or ('does' in ww) or ('will' in ww)) and (w.index(ww
) == i+1):
                    #print 'hiiiii'
                    remain_phrase.append(w[:i+1])
                    remain_phrase.append(w[i+1:])
                    remain_phrase.remove(w)
            for ww in w:
                if (',' in ww):
                    ww.remove(',')
    #===== End of add new (end of 'when' detection) =====

    #===== If there are duplicate phrases, make the difference =====
    i = 1
    while i <= len(remain_phrase)-1:    # In case of the input consists of two or
more sentences.
        for w in remain_phrase[i]:
            if w in remain_phrase[i-1]:
                for ww in w:
                    k = w.index(ww)
                    s = ww
                    w.remove(ww)
                    ww = s + str(i)
                    w.insert(k,ww)
            else:
                for ww in w:
                    k = w.index(ww)
                    s = ww
                    w.remove(ww)
                    ww = s + str(i)
                    w.insert(k,ww)
        i+=1

    i = 0; j = 1; k = 0; temp = ''
    for w in remain_phrase:
        while i <= len(w)-1:
            temp = w[i][0]
            while j <= len(w)-1:
                if (temp in w[j]) and (w[i] != w[j]) and (i<j):
                    if len(w[j]) == 1:
                        s = ww + str(i)
                        w[j].insert(1,s)
                        w[j].pop(0)
                    else:
                        for ww in w[j]:
                            if ww == temp:
                                s = ww
                                k = w[j].index(ww)
                                w[j].pop(k)
                                ww = s + str(i)
                                w[j].insert(k,ww)
                j+=1; s = ''
            i+=1; j = 1
    #===== Finish to make the difference to duplicate phrases =====
    #=======================================================
=============================================================

    #=======================================================
=============================================================
    #===== Start the translation process (NL sentence --> Logical sentence) =====

    #===== Change natural expression to be logical terms =====
    temp2 = NLExpressionToLMD(remain_phrase)
    #===== Finish to change natural expression to be logical terms =====

    #===== Delete unnecessary square brackets =====
    temp = []; locus = []
    for w in temp2:
        for ww in w:
            if len(ww) == 1:
                for www in ww:
                    temp.append(www)
            else:
                temp.append(ww)
        locus.append(temp)
        temp = []

    # ===== In case of the input sentence consists of conjunction, i.e.'before' or '
after'. =====
    i = 100; j = 0; a = 0; b = 0; temp = []
    for w in remain_phrase:
        for ww in w:
            if 'before' in ww:
                i = w.index(ww)
                j = remain_phrase.index(w)
                a+=1
            elif 'after' in ww:
                i = w.index(ww)
                b+=1

    k=0; l = 0      # Start switching process for loci (in case of 'before' or 'after
')
    if i != 100:
        for w in locus:
            if locus.index(w) == j:
                for ww in w:
                    if w.index(ww) == i:
                        l = len(w)
                        if a != 0:      # in case of conjunction is 'before'
                            temp = ['cand']
                            ww.pop(1)
                            ww.insert(1,temp)
                        elif b != 0:    # in case of conjunction is 'after'
                            temp = ['cand']
                            w[0].pop(1)
                            w[0].insert(1,temp)

                            k = 0
                            for x in w[i:]:
                                w.insert(k,x)
                                k+=1
                            del w[l:]
    # ===== End of swith input sentence in case of the conjunction ('before' or 'afte
r') case =====

    #=====================
=====================================================================================
========
    #===== Start combination process =====
    #--- Combine locus terms of phrases into one ---
    temp = ""; temp1 = ""; temp2 = ""; i = 0
    for w in locus:
        for ww in w:
            #print (len(ww),ww)
            if len(ww) != 4:
                for www in ww:
                    #print ww
                    if ww.index(www) == 0:
                        #print www[3][1]
                        if www[3][1] != 'none':
                            temp = www[3][1]    # Keep the dependant variable
                        else:
                            temp = www[3][0]
                        temp1 = ww[1][0][:-1]    # Keep the object name
                        temp2 = ww[0][0] + temp1
                        for x in www[2]:
                            #print (x,temp1)
                            for xx in x:
                                if xx == temp:
                                    i = x.index(xx)
                                    x.pop(i)
                                    x.insert(i,temp1)    # Replace dependant variable
by object name
                        ww[0].pop(0)
                        ww[0].insert(0,temp2)
                        www[3].pop(1)
                        www[3].insert(1,temp1)
                ww.pop(1)
    temp = ""; temp1 = ""; temp2 = ""; i = 0
    #===== Delete useless brackets =====
    i = 0; temp = []
    for w in locus:
        for ww in w:
            if len(ww) == 1:
                temp = ww[0]
                i = w.index(ww)
                w.pop(i)
                w.insert(i,temp)
                temp = []
    #===== Finish to delete useless brackets =====

    #===== Delete useless term in locus, and remain_phrase (i.e. 'to' and 'be' and '
please' term) =====
    for w in locus:          # delete useless term in locus
        for ww in w:
            if (len(ww) == 4) and ((ww[0] == 'be_') or (ww[0] == 'to_') or (ww[0
] == 'please_') or (ww[0] == 'will_')
                                      or ww[0] == 'do_'):
                w.remove(ww)

    for w in remain_phrase:      # delete useless term in remain_pharse
        for ww in w:
            if len(ww) == 1:
                for www in ww:
                    if (not www[-1].isdigit()):
                        if (www == 'be') or (www == 'to') or (www == 'please') or (ww
w == 'will'):
                            w.remove(ww)
                    else:
                        if (www[:-1] == 'be') or (www[:-1] == 'to') or (www[:-1] == '
please') or (www[:-1] == 'will'):
                            w.remove(ww)
    #===== Finish to delete useless terms =====


    # ===== Start to combine 'to' and 'from' terms =====
    i = 100;     j = 100;    temp = '';    temp1 = '';        temp2 = ''
    for w in locus:
        for ww in w:
            if ('from_' in ww[0]) and (ww[3][0] == 'x'):
                i = w.index(ww)
                temp = ww[0]  # Keep the entity name
                temp1 = ww[3][1]  # Keep the dependant variable
            elif ('to_') and (ww[3][0] == 'x') and (w.index(ww) == i + 1):
                for x in ww[2]:
                    for xx in x:
                        if xx == 'p':
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j, temp1)
                temp2 = temp + '_' + ww[0]
                ww.pop(0)
                ww.insert(0, temp2)
                temp2 = temp1 + '_' + ww[3][1]
                ww[3].pop(1)
                ww[3].insert(1, temp2)
                w.pop(i)
                i = 100
    # ===== End of combination 'to' and 'from' terms =====
    #===== Combine first two terms (in case of the first term is 'pn' or 'n') =====
    temp = ""; temp1 = "";temp2 = ""; tempp = ""; i = 0
    for w in locus:
        for ww in w:
            if (w.index(ww) == 0) and (len(ww) == 4) and (len(ww[2]) == 1) \
                    and ((ww[2][0][5] == 'pn') or (ww[2][0][5] == 'n') or (ww[2][0][5
] == 'pn?')):
                temp = ww[0][:-1]   # Keep the entity name
            elif (w.index(ww) == 0) and (len(ww) == 4) and (len(ww[2]) != 1) \
                    and ((ww[2][0][5] == 'pn') or (ww[2][0][5] == 'n') or (ww[2][0][5
] == 'pn?')):
                tempp = ww[0]  # Keep the entity name
                temp1 = ww[3][0]
            elif (w.index(ww) == 1) and (temp != "") and ('where' not in text):
                temp1 = ww[3][0]    # Keep the governor variable
                for x in ww[2]:
                    for xx in x:
                        if xx == temp1:
                            i = x.index(xx)
                            x.pop(i)
                            x.insert(i,temp)
                temp2 = temp +'_' + ww[0]
                ww.pop(0)
                ww.insert(0,temp2)
                ww[3].pop(0)
                ww[3].insert(0,temp)
                if ww[1] != w[0][1]:
                    ww.pop(1)
                    ww.insert(1,w[0][1])
                w.pop(0)
                temp = ""

            elif (w.index(ww) == 1) and (temp != "") and ('where' in text):
                temp1 = ww[3][0]    # Keep the governor variable
                for x in ww[2]:
                    for xx in x:
                        if (xx == 'p') or (xx == 'q'):
                            i = x.index(xx)
                            x.pop(i)
                            x.insert(i,temp)
                temp2 = temp +'_' + ww[0]
                ww.pop(0)
                ww.insert(0,temp2)
                ww[3].pop(0)
                ww[3].insert(0,temp)
                if ww[1] != w[0][1]:
                    ww.pop(1)
                    ww.insert(1,w[0][1])
                w.pop(0)
                temp = ""

            elif (w.index(ww) == 1) and (tempp != ""):
                temp2 = ww[3][1]    # Keep the governor variable
                for x in w[0][2]:
                    for xx in x:
                        if xx == temp1:
                            i = x.index(xx)
                            x.pop(i)
                            x.insert(i,temp2)
                w[0].pop(0)
                w[0].insert(0,tempp + ww[0])
                w[0][3].pop(0)
                w[0][3].insert(0,temp2)
                w.pop(1)
                tempp = ""
    #===== Finish to combine first two terms =====

    #===== Start to combine other terms =====
    i = 100; temp = ''; temp1 = ''; temp2 = ''
    for w in locus:
        for ww in w:
            #print ww
            if (ww[3][0] != 'x') and (not 'keep' in ww[0]):      # Check that the locu
s is contained of governor, or not.
                i = w.index(ww)
                temp = ww[3][0]      # Keep the governor variable
                temp1 = ww[0]
            elif (ww[3][0] != 'x') and ('keep' in ww[0]):
                i = w.index(ww)
                temp = ww[3][1]      # keep the governor variable
                temp1 = ww[0]
            else:    # Check if the next term is lack of governor or not
                if w.index(ww) == i+1:
                    temp2 = ww[3][0]     # Keep the governor variable of recent locus.
                    for x in ww[2]:
                        for xx in x:
                            if xx == temp2:
                                i = x.index(xx)
                                x.pop(i)
                                x.insert(i,temp)
                    ww[3].pop(0)
                    ww[3].insert(0,temp)
                    temp2 = temp1 + "_" + ww[0]
                    ww.pop(0)
                    ww.insert(0,temp2)
                    i = 100

    #===== Combine 'V' and 'AJ' terms =====
    temp = ""; temp1 = ""; temp2 = ""; tempp = ""; i = 0; j = 0
    for w in locus:
        for ww in w:
            if (len(ww) == 4) and (ww[2][0][5] == 'v'):
                temp = ww[0][:-1]  # Keep the entity name
                temp1 = ww[3][1]     # Keep the dependant variable in first locus term
                j = w.index(ww)
            elif (w.index(ww) == j+1) and (temp != "") and ('where' not in text) and
(ww[2][0][5] == 'aj'):
                temp2 = ww[3][1]  # Keep the dependant variable
                for x in w[j][2]:
                    for xx in x:
                        if xx == temp1:
                            i = x.index(xx)
                            x.pop(i)
                            x.insert(i, temp2)
                w[j][3].pop(1)
                w[j][3].insert(1, temp2)
                temp2 = temp + ww[0]
                w[j].pop(0)
                w[j].insert(0, temp2)
    #===== End of combination of 'V' and 'AJ' terms =====
    # ===== Combine 'AJ' and 'PP' terms =====
    i = 100; temp = ''; temp1 = ''; temp2 = ''; j = 0
    for w in locus:
        for ww in w:
            if ww[2][0][5] == 'aj':
                i = w.index(ww)
                temp = ww[3][0]     # Keep the governor variable
                temp2 = ww[0]
            elif (ww[2][0][5] == 'pp') and (w.index(ww) == i + 1) and (ww[3][0] == 'x
'):
                temp1 = ww[3][0]
                for x in ww[2]:
                    for xx in x:
                        if (xx == temp1):
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j, temp)
                ww[3].pop(0)
                ww[3].insert(0, temp)
                temp1 = temp2 + "_" + ww[0]
                ww.pop(0)
                ww.insert(0, temp1)
                i = 100
    # ===== End of combination 'AJ' and 'PP' terms =====

    #===== Combine 'PP' and 'AJ' terms =====
    i = 100; temp = ''; temp1 = ''; temp2 = ''
    for w in locus:
        for ww in w:
            if ww[2][0][5] == 'pp':
                i = w.index(ww)
                temp = ww[3][0]
            elif (ww[2][0][5] == 'aj') and (w.index(ww) == i + 1):
                temp1 = ww[3][1]
                for x in w[i][2]:
                    for xx in x:
                        if (x.index(xx) == 2) and (xx == 'p'):
                            x.pop(2)
                            x.insert(2, temp1)
                        elif (x.index(xx) == 3) and ((xx == 'q') or (xx == 'p')):
                            x.pop(3)
                            x.insert(3, temp1)
                w[i][3].pop(1)
                w[i][3].insert(1,temp1)
                temp1 = temp + "_" +ww[0]
                w[i].pop(0)
                w[i].insert(0, temp1)
                i = 100
    #===== End of combination 'PP' and 'AJ' terms =====

    #===== Combine 'V' and 'PP' terms =====
    i = 100; temp = ''; temp1 = ''; temp2 = ''
    for w in locus:
        for ww in w:
            if ww[2][0][5] == 'v':
                i= w.index(ww)
                temp = ww[0]
            elif (ww[2][0][5] == 'pp') and (w.index(ww) == i+1):
                for x in ww[2]:
                    temp1 = x[2]        # Keep 'from' variable
                    temp2 = x[3]        # Keep 'to' variable
                for x in w[i][2]:
                    for xx in x:
                        if (x.index(xx) == 2) and (xx == 'p'):
                            x.pop(2)
                            x.insert(2,temp1)
                        elif (x.index(xx) == 3) and (xx == 'q'):
                            x.pop(3)
                            x.insert(3,temp2)
                temp1 = temp + ww[0]
                w[i].pop(0)
                w[i].insert(0,temp1)
                w.pop(w.index(ww))
                i = 100
    #===== End of combine 'V' and 'PP' terms =====

    #===== Combination of 'PP' and 'V' (in case of the governor of 'V' is 'x') =====
    i = 100; j =100; temp = ""; temp1 = ""; temp2 = ""; count = 100
    for w in locus:
        for ww in w:
            if (ww[2][0][5] == 'pp') and (not 'with' in ww[0]):
                i= w.index(ww)
                temp = ww[0]
                temp1 = ww[3][1]        # Keep the dependant of a term
                count = locus.index(w)
            elif (ww[2][0][5] == 'v') and (ww[3][0] == 'x') and (w.index(ww) == i+1)
and (count == locus.index(w)):
                temp2 = ww[3][0]        # Keep the governor of recent term
                for x in ww[2]:
                    for xx in x:
                        if xx == temp2:
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j,temp1)        # Replaced governor term by depe
ndant of last term (temp1)
                ww[3].pop(0)
                ww[3].insert(0,temp1)
                temp2 = temp + '_' + ww[0]
                ww.pop(0)
                ww.insert(0,temp2)
                i = 100
    #===== End of combination of 'PP' and 'V' =====

    #===== Combine 'PN' and 'V' terms =====
    i = 100; j = 100; temp = ""; temp1 = ""; temp2 = ""
    for w in locus:
        for ww in w:
            if ((ww[2][0][5] == 'pn') or (ww[2][0][5] == 'pn?') or (ww[2][0][5] == 'n
')) and (ww[3][0] == 'x'):
                #print ww
                i = w.index(ww)
                temp = ww[0][:-1]
                temp1 = ww[1]
            elif (ww[2][0][5] == 'v') and (ww[3][0] == 'x') and (w.index(ww) == i+1):
                #print ww
                temp2 = ww[3][0]
                for x in ww[2]:
                    for xx in x:
                        if xx == temp2:
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j,temp)
                if ww[1] != temp1:
                    ww.pop(1)
                    ww.insert(1,temp1)
                ww[3].pop(0)
                ww[3].insert(0,temp)
                temp2 = temp +'_'+ww[0]
                ww.pop(0)
                ww.insert(0,temp2)
                w.pop(i)
                i = 100
    #===== End of combination of 'PN' and 'V' terms =====

    # ===== Combine 'PN' and 'PP' terms =====
    i = 100; j = 100; temp = ""; temp1 = ""; temp2 = ""
    for w in locus:
        for ww in w:
            if ((ww[2][0][5] == 'pn') or (ww[2][0][5] == 'pn?') or (ww[2][0][5] == 'n
')) and (ww[3][0] == 'x'):
                # print ww
                i = w.index(ww)
                if ww[0][-1] == '_':
                    temp = ww[0][:-1]
                else:
                    temp = ww[0]
                temp1 = ww[1]
            elif (ww[2][0][5] == 'pp') and (ww[3][0] == 'x') and (w.index(ww) == i +
1):
                temp2 = ww[3][0]
                for x in ww[2]:
                    for xx in x:
                        if xx == temp2:
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j, temp)
                if ww[1] != temp1:
                    ww.pop(1)
                    ww.insert(1, temp1)
                ww[3].pop(0)
                ww[3].insert(0, temp)
                temp2 = temp + '_' + ww[0]
                ww.pop(0)
                ww.insert(0, temp2)
                w.pop(i)
                i = 100
    # ===== End of combination of 'PN' and 'PP' terms =====

    # ===== Combine 'PP' and 'V' terms =====
    i = 100;     j = 100;     temp = "";     temp1 = "";     temp2 = ""
    for w in locus:
        for ww in w:
            if (ww[2][0][5] == 'pp') and (ww[3][0] != 'x'):
                i = w.index(ww)
                if (i+1 <= len(w)-1) and ('drive' in w[i+1][0]):
                    temp = ww[3][0]
                else:
                    temp = ww[3][1]
                temp1 = ww[1]
            elif (ww[2][0][5] == 'v') and (ww[3][0] == 'x') and (w.index(ww) == i + 1
):
                temp2 = ww[3][0]
                for x in ww[2]:
                    for xx in x:
                        if xx == temp2:
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j, temp)
                ww[3].pop(0)
                ww[3].insert(0, temp)
                temp2 = temp + '_' + ww[0]
                ww.pop(0)
                ww.insert(0, temp2)
                i = 100
    # ===== End of combination of 'PP' and 'V' terms =====

    # ===== Combine 'V' and 'PP' terms =====
    i = 100; j = 100; temp = ''; temp1 = ''; temp2 = ''
    for w in locus:
        for ww in w:
            if (ww[2][0][5] == 'v') and (ww[3][0] != 'x'):
                i = w.index(ww)
                temp = ww[3][0]
                temp2 = ww[0]
            elif (ww[2][0][5] == 'pp') and (ww[3][0] == 'x') and (w.index(ww) == i +
1):
                temp1 = ww[3][0]
                for x in ww[2]:
                    for xx in x:
                        if xx == temp1:
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j, temp)
                temp1 = temp2 + ww[0]
                ww.pop(0)
                ww.insert(0, temp1)
                ww[3].pop(0)
                ww[3].insert(0,temp)
                i = 100
    # ===== End of combine 'V' and 'PP' terms =====

    #===== Combine 'V' and 'V' terms =====
    i = 100; j = 100; temp = ""; temp1 = ""; temp2 = ""
    for w in locus:
        for ww in w:
            if (ww[2][0][5] == 'v') and (ww[3][0] == 'x') and (ww[3][1] != 'y'):
                i = w.index(ww)
                temp = ww[3][1]
                temp1 = ww[0]
            elif (ww[2][0][5] == 'v') and (ww[3][0] == 'x') and (ww[3][1] == 'y') and
 (w.index(ww) == i+1):
                temp2 = ww[3][1]
                for x in ww[2]:
                    for xx in x:
                        if xx == temp2:
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j,temp)
                temp2 = temp1 + ww[0]
                ww.pop(0)
                ww.insert(0,temp2)
                ww[3].pop(1)
                ww[3].insert(1,temp)
                i = 100
    #===== End of 'V' and 'V' combination =====

    #===== Combine 'AJ' and 'V' terms =====
    i = 100; j = 100; temp = ""; temp1 = ""; temp2 = ""
    for w in locus:
        for ww in w:
            if (ww[2][0][5] == 'aj'):
                i = w.index(ww)
                temp = ww[3][1]
                temp1 = ww[0]
            elif (ww[2][0][5] == 'v') and (ww[3][0] == 'x') and (w.index(ww) == i + 1
):
                temp2 = ww[3][0]
                for x in ww[2]:
                    for xx in x:
                        if xx == temp2:
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j, temp)
                temp2 = temp1 + '_' +ww[0]
                ww.pop(0)
                ww.insert(0, temp2)
                ww[3].pop(0)
                ww[3].insert(0, temp)
                i = 100
    #===== End of combination of 'AJ' and 'V' terms =====

    # ===== Combine 'PP' and 'PP' terms =====
    i = 100; temp = ''; temp1 = ''; temp2 = ''; j = 0
    for w in locus:
        for ww in w:
            if (ww[2][0][5] == 'pp') and (w.index(ww) != i+1):
                i = w.index(ww)
                temp = ww[3][0]  # Keep the governor variable
                temp2 = ww[0]
            elif (ww[2][0][5] == 'pp') and (w.index(ww) == i + 1) and (ww[3][0] == 'x
'):
                temp1 = ww[3][0]
                for x in ww[2]:
                    for xx in x:
                        if (xx == temp1):
                            j = x.index(xx)
                            x.pop(j)
                            x.insert(j, temp)
                ww[3].pop(0)
                ww[3].insert(0, temp)
                temp1 = temp2 + "_" + ww[0]
                ww.pop(0)
                ww.insert(0, temp1)
                i = 100
    # ===== End of combination 'PP' and 'PP' terms =====

    #===== Delete useless term =====
    for w in locus:
        if len(w) > 1:
            for ww in w:
                if len(ww[2]) == 1:
                    for www in ww[2]:
                        if (www[0] == www[1]) and (www[2] == 'p') and (www[3] == 'q
'):
                            w.remove(ww)
```

```
#===== End of deletion of useless term =====

#===== Summarise final locus formula =====
final_locus = []; i = 0; temp = []; j = 100
for w in locus:
    if (len(w) == 1) and (len(w[0][2]) == 1):
        temp.append(copy.deepcopy(w[0][2]))
        final_locus.append(temp)
        temp = []
    elif (len(w) == 1) and (len(w[0][2]) > 1):
        temp.append(copy.deepcopy(w[0][1]))
        temp.append(copy.deepcopy(w[0][2]))
        final_locus.append(temp)
        temp = []
    else:
        for ww in w:
            j = w.index(ww)
            if (j != 0):
                temp.append(copy.deepcopy(ww[1]))
            elif (j == 0) and (len(ww[2]) > 1):
                temp.append(copy.deepcopy(ww[1]))
            elif (j == 0) and (len(ww[2]) == 1) and (len(w[j + 1][2]) > 1):
                temp.append(copy.deepcopy(ww[1]))
        for ww in w:
            temp.append(copy.deepcopy(ww[2]))
        final_locus.append(temp)
        temp = []; i = 0
#===== End of summarization =====

#===== Delete unnecessary parentheses =====
temp = ""; i = 100; j = 100; temp1 = ""
for w in final_locus:
    for ww in w:
        if (ww != ['sand']) and (ww != ['cand']):
            if len(ww) == 1:
                i = w.index(ww)
                temp = copy.deepcopy(ww)
                w.pop(i)
                for x in temp:
                    w.insert(i,x)
            else:
                j = w.index(ww)
                temp1 = copy.deepcopy(ww)
    if (temp1 != "") and (j != 100):
        w.pop(j)
        for x in temp1:
            w.insert(j,x)
            j+=1
        temp1 = ""; j =100
#===== End of deletion of unnecessary parentheses =====

#===== Call Sex detection function =====
temp = ''
for w in final_locus:
    for ww in w:
        if 'she' in ww:
            gender = 'female'
            word = 'she'
            temp = SexDetection(gender,final_locus,word)
        elif 'he' in ww:
            gender = 'male'
```

```
                word = 'he'
                temp = SexDetection(gender,final_locus,word)
    if temp != "":
        final_locus = []
        final_locus = temp

    i = 0; j = 0; k = 0
    for w in final_locus:
        for ww in w:
            if (ww == ['sand']) and (ww == ['cand']):
                i += 1
            if (len(ww) != 1) and (len(ww) != 8):
                k = len(ww)
                j = w.index(ww)
                for www in ww:
                    w.insert(j,www)
                w.remove(ww)
        if (j != 0) and (k != 0):
            i = 1
            while i < k:
                w.insert(j - 1, ['sand'])
                i += 1

    #===== In case of ' I want to call Tom', change 'y' to be 'Anna' =====
    if ('want' in text) and ('call' in text):
        for w in final_locus:
            for ww in w:
                if (ww != ['sand']) and (ww != ['cand']) and (len(ww) == 8):
                    for x in ww:
                        if x =='y':
                            i = ww.index(x)
                            ww.pop(i)
                            ww.insert(i,'Anna')

    #===== Call Inference rules =====
    final_locus = Inference(final_locus)
    print ("\n\t---Final locus after applied postulate and inference rules---")
    print ("\tfinal locus = %s" % final_locus)

    end = time.time()
    print ("\n\t\033[0;33m>>> Time-consuming Lmd translation = %.3f seconds.\033[0;m"
% (end - start))
    #===== End of calling a function =====


    #===== Q&A =====
    #===== Call Pattern Matching function =====
    name = ""; i = 0
    start = time.time()
    if '?' in text:
        ans = PatternMatching(final_locus,text)
        if ans != []:
            for w in ans:        # Check for the place in place_position
                if w in place_position:
                    i+=1
            for w in ans:
                if w.endswith("_"):
                    name = w[:-1]
            if (i == 0) or (name == ""):
                num = randint(1, 2)
                if num == 1:
                    print ('\n\033[1;31mAnna: No, %s does not.\033[1;m'%name)
                else:
                    print ('\n\033[1;31mAnna: I do not think so.\033[1;m')
    end = time.time()
    print ("\n\t\033[0;33m>>> Time-consuming pattern matching = %.3f seconds.\033[0;m
" % (end - start))
    #===== End of pattern matching function =====

    i = 0
    starting_point = ""; goal_point = ""
    for w in final_locus:
        for ww in w:
            if (ww != ['sand']) and (ww != ['cand']):
                for x in place_position:
                    if ww[2] == x:
                        starting_point = x
                        i+=1
                    elif ww[3] == x:
                        goal_point = x
                        i+=1
                    elif (ww[2] == 'where') or (ww[3] == 'where'):
                        if ww[0] == x:
                            starting_point = x
                            i+=1
                        elif ww[1] == x:
                            goal_point = x
                            i+=1
                    elif (('how' in text) or ('How' in text)) and ('?' in text):
                        if (ww[2] == x):
                            starting_point = x
                            i+=1
                        elif (ww[3] == x):
                            goal_point = x
                            i+=1
                        elif (ww[2] == 'how') and (ww[3] == 'how'):
                            i+=1

    if (i == 0) and ((text != 'Thank you.') and (text != 'thank you.') and (text != '
Thank you') and (text != 'thank you')) and ('call' not in text):
        print "\n\033[1;31mAnna: Do not have the place in our system.\033[1;m"
    elif (i == 0) and ('call' in text):
        i+=1


    #===== Animation generator =====
    route = []
    if i != 0:
        text = text.lower()
        start = time.time()
        #===== For 'when "x" drives a car to "q" with "y", does she/he move?' =====
        if ('tom' and 'mary') and ('drive' or 'drove') in text:
            if (i > 0) and (name != ''):
                num = randint(1, 2)
                if num == 1:
                    print ('\n\033[1;31mAnna: Yes, %s does.\033[1;m' % name)
                else:
                    print ('\n\033[1;31mAnna: Sure, %s does.\033[1;m' % name)

            anna_taro.hideturtle()
            for w in final_locus:
                for ww in w:
                    if (ww != ['sand']) and (ww != 'cand'):
                        if ww[2] in place_position:
                            tom_mary.goto(place_position[ww[2]])
                            tom_mary.showturtle()
                        elif ww[3] in place_position:
                            tom_mary.showturtle()
                            route = MinDistance(tom_mary.pos(), place_position[ww[3]]
)

            route_new = []
            if route != []:
                for w in route:
                    if w not in route_new:
                        route_new.append(w)

                num = 0
                for w in route_new:
                    for x in close_point:
                        if w == close_point[x]:
                            if (x != 'bridge_1') and (x != 'home'):
                                if route_new.index(w) == 1:
                                    print "\033[1;31mAnna: From %s,\033[1;m"%x
                                elif route_new.index(w) == len(route_new)-2:
                                    print "\t\033[1;31m finally %s will arrive at the
 %s.\033[1;m"%(name,x)
                                else:
                                    num = randint(1,2)
                                    if num == 1:
                                        print "\t\033[1;31m then, go to the %s,\033[1
;m"%x
                                    else: print "\t\033[1;31m after that, move to the
 %s,\033[1;m"%x

                for w in route_new:
                    if 'flower_bed' in text:
                        if w == (320.00,-90.00):
                            route_new.remove((320.00,-90.00))

                for w in route_new:
                    tom_mary.goto(w)
            else: print ("\n\033[1;31mAnna: Sorry!! cannot go\033[1;m")
        #=====================================================================

        #===== For 'when Tom is at home with Mary, does she move?' =====
        if (('at' and 'move' and '?') in text) and (i > 0):
            for w in final_locus:
                for ww in w:
                    if (ww != ['sand']) and (ww != ['cand']):
                        if ww[2] in place_position:
                            if ('tom' and 'mary') in text:
                                tom_mary.hideturtle()
                                tom_mary.goto(place_position[ww[2]])
                                tom_mary.showturtle()
                            elif ('Taro' and 'Anna') in text:
                                anna_taro.hideturtle()
                                taro.goto(place_position[ww[2]])
                                anna.goto(place_position[ww[2]])
                                anna_taro.goto(place_position[ww[2]])
                                anna_taro.showturtle()
                        elif ww[3] in place_position:
                            if ('tom' and 'mary') in text:
                                tom_mary.hideturtle()
                                tom_mary.goto(place_position[ww[3]])
                                tom_mary.showturtle()
                            elif ('Taro' and 'Anna') in text:
                                anna_taro.hideturtle()
                                taro.goto(place_position[ww[3]])
                                anna.goto(place_position[ww[3]])
                                anna_taro.goto(place_position[ww[3]])
                                anna_taro.showturtle()

            count = 0
            for w in final_locus:
                for ww in w:
                    if (ww!=['sand']) and (ww!=['cand']):
                        if (ww[2] == ww[3]) and (ww[2] in place_position):
                            if ('tom' and 'mary') in text:
                                if tom_mary.pos() == place_position[ww[2]]:
                                    count+=1
                            elif ('taro' and 'anna') in text:
                                if anna_taro.pos() == place_position[ww[2]]:
                                    count+=1
            if count > 0:
                num = randint(1,2)
                if num == 1:
                    print ("\n\033[1;31mAnna: No, %s does not.\033[1;m" % name)
                else:
                    print ("\n\033[1;31mAnna: No. I do not think so.\033[1;m")
        #=====================================================================================
==================================



        #===== For 'please take me to "q".', 'I want to go to "q".' =====
        elif ((('please' in text) or ('Please' in text)) and ('take' in text) and (go
al_point != '')) or (('want' in text) and ('go' in text) and (goal_point != '')):
            tom_mary.hideturtle()
            if anna.pos() != taro.pos():
                print "\n\033[1;31mAnna: I am going back to see you.\033[1;m"
                route = MinDistance(anna.pos(),taro.pos())

                for w in route:
                    anna.goto(w)

            if anna.pos() == taro.pos():
                anna_taro.showturtle()
                taro.hideturtle()
                anna.hideturtle()
                print "\n\033[1;31mAnna: Let's go to the %s.\033[1;m"%goal_point
                route = MinDistance(anna_taro.pos(),place_position[goal_point])

                route_new = []
                for w in route:
                    if w not in route_new:
                        route_new.append(w)

                num = 0
                for w in route_new:
                    for x in close_point:
                        if w == close_point[x]:
                            if (x != 'bridge_1') and (x != 'home'):
                                if route_new.index(w) == 1:
                                    print "\n\033[1;31mAnna: From here, the %s,\033[1
```

```
;m"%x
                                elif route_new.index(w) == len(route_new) -2:
                                    print "\t\033[1;31m Now, we arrive at the %s.\033
[1;m" %goal_point
                                else:
                                    num = randint(1, 2)
                                    if num == 1:
                                        print "\t\033[1;31m next, go to the %s.\033[1
;m" %x
                                    else:
                                        print "\t\033[1;31m then, pass the %s.\033[1;
m" %x

                for w in route_new:
                    if 'flower_bed' in text:
                        if w == (320.00, -90.00):
                            route_new.remove((320.00, -90.00))
                for w in route_new:
                    anna_taro.goto(w)
                    anna.goto(w)
                    taro.goto(w)
        #==============================================================================
==========

        #===== For 'please go to "q".' =====
        elif (('please' in text) or ('Please' in text)) and ('go' in text) and (
goal_point != ''):
            tom_mary.hideturtle()
            anna_taro.hideturtle()
            taro.showturtle()
            anna.showturtle()

            print "\n\033[1;31mAnna: I will go to the %s.\033[1;m" % goal_point
            route = MinDistance(anna.pos(), place_position[goal_point])

            route_new = []
            for w in route:
                if w not in route_new:
                    route_new.append(w)

            num = 0; temp = ''
            for w in route_new:
                for x in close_point:
                    if w == close_point[x]:
                        if (x != 'bridge_1') and (x != 'home'):
                            if route_new.index(w) == 1:
                                print "\n\033[1;31mAnna: From here, the %s,\033[1;m
" % x
                            elif route_new.index(w) == len(route_new) - 2:
                                print "\t\033[1;31m then, I will pass the %s.\033[1;m
" %temp
                                print "\t\033[1;31m Finally, I arrive at the %s.\033[
1;m" % goal_point
                            else:
                                temp = temp + " " + x + ","

            for w in route_new:
                if 'flower_bed' in text:
                    if w == (320.00, -90.00):
                        route_new.remove((320.00, -90.00))
            for w in route_new:
                anna.goto(w)
        #===========================================================================

        #===== For 'where is "p"?' =====
        elif (('where' in text) or ('Where' in text)) and ('?' in text):
            print text
            for w in final_locus:
                for ww in w:
                    if ww[0] in place_position:
                        dis = ClosePlace(place_position[ww[0]])
                        temp = place_position[ww[0]]
                        temp1 = place_position[dis[0]]
                        if ('yard' in text) or ('home' in text) or ('apartment' in te
xt) or ('house' in text):
                            num = randint(1, 4)
                            if num == 1:
                                print ("\n\033[1;31mAnna: The %s is between the\033[1
;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0]) + (
                                " and %s." % dis[1])
                            elif num == 2:
                                print("\n\033[1;31mAnna: The %s is close to the\033[1
;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                            elif num == 3:
                                print("\n\033[1;31mAnna: The %s is next to the\033[1;
m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                            elif num == 4:
                                if (temp[0] > temp1[0]) and (temp[1] > temp1[1]):
                                    print("\n\033[1;31mAnna: The %s is to the right o
f the\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                                else:
                                    print("\n\033[1;31mAnna: The %s is to the left of
 the\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])

                        elif ('post_office' in text) or ('restaurant' in text):
                            num = randint(1, 3)
                            if num == 1:
                                print("\n\033[1;31mAnna: The %s is close to the\033[1
;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                            elif num == 2:
                                print("\n\033[1;31mAnna: The %s is next to the\033[1;
m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                            elif num == 3:
                                if (temp[0] > temp1[0]) and (temp[1] > temp1[1]):
                                    print("\n\033[1;31mAnna: The %s is to the right
of the\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                                else:
                                    print("\n\033[1;31mAnna: The %s is to the left of
 the\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])

                        elif ('bridge' in text):
                            num = randint(1, 2)
                            if num == 1:
                                print("\n\033[1;31mAnna: The %s is near the\033[1;m
" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[2])
                            elif num == 2:
                                print("\n\033[1;31mAnna: The %s is opposite the\033[1
;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[4])

                        elif ('fountain' in text) or ('table' in text):
                            num = randint(1, 2)
                            if num == 1:
                                print("\n\033[1;31mAnna: The %s is near the\033[1;m"
 % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                            elif num == 2:
                                if (temp[0] > temp1[0]) and (temp[1] < temp1[1]):
                                    print("\n\033[1;31mAnna: The %s is to the right
of the\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])
                                elif (temp[0] < temp1[0]) and (temp[1] > temp1[1]):
                                    print("\n\033[1;31mAnna: The %s is to the left of
 the\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[0])

                        elif ('river' in text):
                            print("\n\033[1;31mAnna: The %s is along the road.\033[1;
m" % ww[0])

                        elif ('stair' in text) or ('flower_bed' in text):
                            if ('flower_bed' in text):
                                print("\n\033[1;31mAnna: The %s is to the right of th
e\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[1])
                            elif ('stair' in text):
                                print("\n\033[1;31mAnna: The %s is to the left of the
\033[1;m" % ww[0]) + ("\033[1;31m %s.\033[1;m" % dis[2])

                        else:
                            print("\n\033[1;31mAnna: I have no data about it\033[1;m
")
        #============================================================================
===============

        #===== For 'How...?' =====
        if (('how' in text) or ('How' in text)) and ('?' in text):
            if ('can' in text) or ('could' in text):
                if goal_point != '':
                    if starting_point == '':
                        route = MinDistance(taro.pos(),place_position[goal_point])
                    else:
                        route = MinDistance(place_position[starting_point],place_pos
ition[goal_point])

                    if route != []:
                        route_new = []
                        for w in route:
                            if w not in route_new:
                                route_new.append(w)

                        i = 0
                        for w in route_new:
                            for x in close_point:
                                if w == close_point[x]:
                                    if i == 0:
                                        print ("\n\033[1;31mAnna: From here, %s.\033[
1;m"%x)

                                    elif (x != 'bridge_1') and (x!='home') and (i!=0)
:

                                        num = randint(1,2)
                                        if num == 1:
                                            print ("\t\033[1;31m Next, go to the %s.\
033[1;m"%x)

                                        else:
                                            print ("\t\033[1;31m Then, walk to the %s
.\033[1;m"%x)
                                    i+=1
                else: print ("\n\033[1;31mAnna: I do not know how to go there. Sorry!
!\033[1;m")


            elif ('do' in text) or ('does' in text) or ('did' in text):
                temp = []; direction = []
                if (goal_point != '') or ('river' in text) or ('road' in text):
                    if 'river' in text:
                        for w in close_point:
                            if 'river' in w:
                                temp.append(w)
                        temp.sort()

                        for w in temp:
                            direction.append(w[6:])
                        print ("\n\033[1;31mAnna: the river runs from %s to %s.\033[1
;m" %(direction[0],direction[1]))

                    elif 'road' in text:
                        num = randint(1,2)
                        if num == 1:
                            for w in close_point:
                                if ('road' == w):
                                    for x in place_position:
                                        if close_point[w] == place_position[x]:
                                            print ("\n\033[1;31mAnna: The %s runs
along the %s.\033[1;m" %(w,x))

                        else:
                            for w in close_point:
                                if ('road' in w) and ('road' != w):
                                    temp.append(w)
                            temp.sort()
                            for w in temp:
                                direction.append(w[5:])
                            print ("\n\033[1;31mAnna: The road runs from %s to %s\033
[1;m" %(direction[0],direction[1]))

                    else:
                        num = randint(1,2)
                        if num == 1:
                            print ("\n\033[1;31mAnna: By foot.\033[1;m")
                        else:
                            print ("\n\033[1;31mAnna: By car.\033[1;m")

                else: print ("\n\033[1;31mAnna: Sorry! I do not have any information
about it.\033[1;m")

            elif ("are" in text) and ('you' in text):
                print ("\n\033[1;31mAnna: I am fine, thank you!\033[1;m")

    #===== In case of 'I want to call Tom' =====
        elif ('call' in text):
            tom_mary.hideturtle()
            anna_taro.showturtle()
            if telephone.pos() != taro.pos():
                if anna.position() != taro.pos():  # in case of Anna is not with Taro
                    route = MinDistance(anna.pos(), taro.pos())
                    for i in route:
                        anna.goto(i)
                        anna_taro.goto(i)
                    anna.hideturtle()
                    taro.hideturtle()
                    anna_taro.showturtle()
                if anna_taro.pos() != (275.00, 175.00):  # in case if Anna and Taro
are not at home.
                    route = MinDistance(anna_taro.pos(), place_position['home'])
                    for i in route:
                        anna.goto(i)
                        taro.goto(i)
                        anna_taro.goto(i)

                temp = CallPhone(final_locus)
                route = temp[0]
                temp2 = temp[1]
                temp3 = temp[2]
                home_place_position = HomePlacePosition()

                if route[0][2][0][2] in HomePlacePosition():
                    screen.bgpic('home_2.gif')
                    telephone.goto(home_place_position[route[0][2][0][2]])
                    telephone.showturtle()
                    anna_taro.hideturtle()
                    anna_taro.goto(275.00,175.00)
                    anna.goto(275.00,175.00)
                    taro.goto(275.00,175.00)
                    anna.showturtle()
                    taro.showturtle()
                    print ("\nAnna: I will go to the %s." % route[0][2][0][2])
                    print ("\nAnna: And bring your phone back to you")
                    screen.delay(50)
                    anna.goto(home_place_position[route[0][2][0][2]])
                    anna_phone.goto(home_place_position[route[0][2][0][2]])

                    if taro.pos() != telephone.pos():  # Check if the phone is not wi
th Taro.
                        # screen.delay(50)
                        anna.hideturtle()
                        telephone.hideturtle()
                        anna_phone.showturtle()
                        anna.goto(taro.pos())
                        telephone.goto(taro.pos())
                        anna_phone.goto(taro.pos())
                        anna_phone.hideturtle()
                        taro.hideturtle()
                        anna_taro.showturtle()
                        telephone.showturtle()
                        print("\nAnna: This is your phone.")

                    temp2 = []
                    for w in route:
                        temp2.append(w)

                    if taro.pos() == telephone.pos():
                        for w in temp2:
                            w.pop(0)
                            w.insert(0, 'anna')
                            w.pop(3)
                            w.insert(3, 'taro')
                anna_start_position = anna.pos()
                taro_start_position = taro.pos()
                taro_anna_start_position = anna_taro.pos()
```

```
            else:
                print("\nAnna: the phone is with you, you can use it now.")

        end = time.time()
        print ("\n\t\033[0;33m>>> Time-consuming animation generator = %.3f seconds.\
033[0;m" % (end - start))
    #===== End of 'I want to call Tom' =====

    elif (text == 'Thank you.') or (text == 'thank you.') or (text == 'Thank you') or
 (text == 'thank you'):
        print("\n\033[1;31mAnna: You are welcome.\033[1;m")

    else: print "\n\033[1;31m--- No animation displayed ---\033[1;m"
    #===== End of Animation generator =====
    #===== End of Q&A =====
    #===== End of combination process =====
    #=================================================================================
===================================

    text = raw_input("\nYour command: ")

if (text == 'q') or (text == 'Q'):
    exit()

#Wait for user to close display window.
turtle.mainloop()
    #=== End of while loop ===
```

D.2 setpositionofplaces.py

```
#===== Create turtles as shop position =====
import turtle
from turtle import *
#import sys

def SetPositionofPlaces():

    place_position = {}

    #===== Set position of places =====
    post_office = turtle.Turtle()
    post_office.penup()
    post_office.color('yellow')
    post_office.turtlesize(0.01)
    post_office.setposition(80.00,230.00)
    place_position['post_office'] = post_office.pos()

    yard = turtle.Turtle()
    yard.penup()
    yard.color('yellow')
    yard.turtlesize(0.01)
    yard.setposition(-20.00,175.00)
    place_position['yard'] = yard.pos()

    home = turtle.Turtle()
    home.penup()
    home.color('yellow')
    home.turtlesize(0.01)
    home.setposition(-80.00,120.00)
    place_position['home'] = home.pos()

    house = turtle.Turtle()
    house.penup()
    house.color('yellow')
    house.turtlesize(0.01)
    house.setposition(-80.00,120.00)
    place_position['house'] = house.pos()

    apartment = turtle.Turtle()
    apartment.penup()
    apartment.color('yellow')
    apartment.turtlesize(0.01)
    apartment.setposition(-225.00,40.00)
    place_position['apartment'] = apartment.pos()

    restaurant = turtle.Turtle()
    restaurant.penup()
    restaurant.color('yellow')
    restaurant.turtlesize(0.01)
    restaurant.setposition(-400.00,-80.00)
    place_position['restaurant'] = restaurant.pos()

    bridge = turtle.Turtle()
    bridge.penup()
    bridge.color('yellow')
    bridge.turtlesize(0.01)
    bridge.setposition(220.00,115.00)
    place_position['bridge'] = bridge.pos()

    stair = turtle.Turtle()
    stair.penup()
    stair.color('yellow')
    stair.turtlesize(0.01)
    stair.setposition(220.00,-105.00)
    place_position['stair'] = stair.pos()

    fountain = turtle.Turtle()
    fountain.penup()
    fountain.color('yellow')
    fountain.turtlesize(0.01)
    fountain.setposition(435.00,130.00)
    place_position['fountain'] = fountain.pos()

    table = turtle.Turtle()
    table.penup()
    table.color('yellow')
    table.turtlesize(0.01)
    table.setposition(500.00,20.00)
    place_position['table'] = table.pos()

    river = turtle.Turtle()
    river.penup()
    river.color('yellow')
    river.turtlesize(0.01)
    river.setposition(220.00,0.00)
    place_position['river'] = river.pos()

    road = turtle.Turtle()
    road.penup()
    road.color('yellow')
    road.turtlesize(0.01)
    road.setposition(-80.00,0.00)
    place_position['road'] = road.pos()

    flower_bed = turtle.Turtle()
    flower_bed.penup()
    flower_bed.color('yellow')
    flower_bed.turtlesize(0.01)
    flower_bed.setposition(450.00,-180.00)
    place_position['flower_bed'] = flower_bed.pos()

    return place_position
```

D.3 connectionposition.py

```
#====== Create turtles as shop position =====
import turtle
from turtle import *

def ConnectionPosition():

    connection_position = {}

    #===== Set connection position =====
    connection01 = turtle.Turtle()
    connection01.turtlesize(0.01)
    connection01.color('green')
    connection01.penup()
    connection01.setpos(120.00,200.00)
    connection_position['connection01'] = connection01.pos()

    connection02 = turtle.Turtle()
    connection02.turtlesize(0.01)
    connection02.color('green')
    connection02.penup()
    connection02.setpos(50.00,150.00)
    connection_position['connection02'] = connection02.pos()

    connection03 = turtle.Turtle()
    connection03.turtlesize(0.01)
    connection03.color('green')
    connection03.penup()
    connection03.setpos(85.00,130.00)
    connection_position['connection03'] = connection03.pos()

    connection04 = turtle.Turtle()
    connection04.turtlesize(0.01)
    connection04.color('green')
    connection04.penup()
    connection04.setpos(-20.00,90.00)
    connection_position['connection04'] = connection04.pos()

    connection05 = turtle.Turtle()
    connection05.turtlesize(0.01)
    connection05.color('green')
    connection05.penup()
    connection05.setpos(-120.00,0.00)
    connection_position['connection05'] = connection05.pos()

    connection06 = turtle.Turtle()
    connection06.turtlesize(0.01)
    connection06.color('green')
    connection06.penup()
    connection06.setpos(-270.00,-140.00)
    connection_position['connection06'] = connection06.pos()

    connection07 = turtle.Turtle()
    connection07.turtlesize(0.01)
    connection07.color('green')
    connection07.penup()
    connection07.setpos(370.00,90.00)
    connection_position['connection07'] = connection07.pos()

    connection08 = turtle.Turtle()
    connection08.turtlesize(0.01)
    connection08.color('green')
    connection08.penup()
    connection08.setpos(420.00,90.00)
    connection_position['connection08'] = connection08.pos()

    connection09 = turtle.Turtle()
    connection09.turtlesize(0.01)
    connection09.color('green')
    connection09.penup()
    connection09.setpos(420.00,20.00)
    connection_position['connection09'] = connection09.pos()

    connection10 = turtle.Turtle()
    connection10.turtlesize(0.01)
    connection10.color('green')
    connection10.penup()
    connection10.setpos(420.00,-90.00)
    connection_position['connection10'] = connection10.pos()

    connection11 = turtle.Turtle()
    connection11.turtlesize(0.01)
    connection11.color('green')
    connection11.penup()
    connection11.setpos(320.00,-90.00)
    connection_position['connection11'] = connection11.pos()

    connection12 = turtle.Turtle()
    connection12.turtlesize(0.01)
    connection12.color('green')
    connection12.penup()
    connection12.setpos(140.00,-140.00)
    connection_position['connection12'] = connection12.pos()

    connection13 = turtle.Turtle()
    connection13.turtlesize(0.01)
    connection13.color('green')
    connection13.penup()
    connection13.setpos(420.00, -120.00)
    connection_position['connection13'] = connection13.pos()

    return (connection_position)
```

D.4 mindistance.py

```
import turtle
from turtle import *
from setpositionofplaces import SetPositionofPlaces
from connectionposition import ConnectionPosition
from closepoint import ClosePoint
import math

def MinDistance(anna_current_position, goal_position):

    #===== Get the place positions from SetPositionofPlaces function =====
    place_position = SetPositionofPlaces()
    connection_position = ConnectionPosition()
    close_point = ClosePoint()

    route = []; temp = ''
    for w in place_position:
        if anna_current_position == place_position[w]:
            temp = w
            route.append(anna_current_position)     # Keep the starting point in '
route'

    for w in close_point:
        if temp == w:
            route.append(close_point[w])     # Keep the closest point of start point
in 'route'

    #--- Find the possible route ---
    possible_route = []     # Keep the possible point that Anna will walk.
    if (anna_current_position[0] < 140) and (goal_position[0] < 140) and (
anna_current_position[0] > goal_position[0]):
        for i in connection_position:
            temp = connection_position[i]
            if (temp[0] <= 120) and (temp[0] != 85) and (temp[0] > goal_position[0])
and (temp[0] < anna_current_position[0]):
                if not temp in possible_route:
                    possible_route.append(temp)

    elif (anna_current_position[0] < 140) and (goal_position[0] < 140) and (
anna_current_position[0] < goal_position[0]):
        for i in connection_position:
            temp = connection_position[i]
            if (temp[0] <= 120) and (temp[0] != 85) and (temp[0] >
anna_current_position[0]) and (temp[0] <= goal_position[0]):
                if not temp in possible_route:
                    possible_route.append(temp)

    elif (anna_current_position[0] < 140) and (goal_position[0] > 140):
        goal_1 = 'bridge'
        for i in connection_position:
            temp = connection_position[i]
            if (temp[0] <= 85) and (temp[0] > anna_current_position[0]):
                if temp not in possible_route:
                    possible_route.append(temp)
        if (goal_1 in place_position) and (goal_1 not in possible_route):
            possible_route.append(place_position[goal_1])
        for w in close_point:
            if goal_1 in w:
                if close_point[w] not in possible_route:
                    possible_route.append(close_point[w])
        for i in connection_position:
            temp = connection_position[i]
            if (temp[0] > 140) and (temp[1] <= 90) and (goal_position[1] > 90) and (
goal_position[1] < temp[1]):
                if temp not in possible_route:
                    possible_route.append(temp)
            elif (temp[0] > 140) and (temp[1] <= 90) and (goal_position[1] < 90) and
(goal_position[1] < temp[1]):
                if temp not in possible_route:
                    possible_route.append(temp)

    elif (anna_current_position[0] > 140) and (goal_position[0] < 140):
        goal_1 = 'bridge'
        for i in connection_position:
            temp = connection_position[i]
            if (anna_current_position[1] > 90) and (temp[0] > 370) and (temp[1] >= 90
):
                if temp not in possible_route:
                    possible_route.append(temp)
            elif (anna_current_position[1] < 90) and (temp[0] > 140) and (temp[1] >=
anna_current_position[1]):
                if temp not in possible_route:
                    possible_route.append(temp)
        if (goal_1 in place_position) and (goal_1 not in possible_route):
            possible_route.append(place_position[goal_1])
        for w in close_point:
            if goal_1 in w:
                if close_point[w] not in possible_route:
                    possible_route.append(close_point[w])
        for i in connection_position:
            temp = connection_position[i]
            if (goal_position[0] >= -20) and (temp[0] >= 50) and (temp[0] <
goal_position[0]):    #(temp[0] <= 120):
                if not temp in possible_route:
                    possible_route.append(temp)
            elif (goal_position[0] < -20) and (temp[0] <= 50) and (temp[0] >
goal_position[0]):
                if not temp in possible_route:
                    possible_route.append(temp)

    elif (anna_current_position[0] > 140) and (goal_position[0] > 140):
        for i in connection_position:
            temp = connection_position[i]
            if (temp[0] == 420):
                if (anna_current_position[1] < goal_position[1]):
                    if (temp[1] > anna_current_position[1]) and (temp[1] <=
goal_position[1]):
                        if not temp in possible_route:
                            possible_route.append(temp)
                else:
                    if (temp[1] < anna_current_position[1]) and (temp[1] >=
goal_position[1]):
                        if not temp in possible_route:
                            possible_route.append(temp)
    #--- End of finding the possible route ---

    #--- Sort the route from nearest to outermost (From Anna_current_position to
Goal_position) ---
    i = 1
    temp = []
    while len(possible_route) != 0:
        p1 = possible_route[0]
        distance = math.hypot(anna_current_position[0]-p1[0], anna_current_position[1
]-p1[1])
        while i < len(possible_route):
            p1 = possible_route[i]
            if math.hypot(anna_current_position[0]-p1[0], anna_current_position[1]-p1
[1]) < distance:
                distance = math.hypot(anna_current_position[0]-p1[0], anna_current_po
sition[1]-p1[1])
                temp = possible_route[i]
            i+=1
        if temp == []:
            route.append(possible_route[0])
            anna_current_position = possible_route[0]
            possible_route.remove(possible_route[0])
        else:
            route.append(temp)
            anna_current_position = temp
            possible_route.remove(temp)
        i = 1
        temp = []
    #--- End of sorting the route ---

    for w in place_position:
        if goal_position == place_position[w]:
            temp = w
    for w in close_point:
        if temp == w:
            if close_point[w] not in route:
                route.append(close_point[w])
            route.append(goal_position)
    return (route)
```

D.5 my_dictionary.py

```
import nltk
import en
import copy
from nltk.corpus import wordnet
from nltk.tree import *
from extractphrase import ExtractPhrases
from nltk.stem.wordnet import WordNetLemmatizer

#My dictionary is used to match word and its word type. Moreover, it also extract
phrases from clauses.
#Then send the result back to main function.
def my_dictionary(text):
    my_word = [('avoid','v1'), ('become','v1'), ('believe','v1'), ('bring','v1'), ('
buy','v1'), ('carry','v1'),('call','v1'),
            ('catch','v1'), ('come','v1'), ('cry','v1'), ('drive','v1'), ('feel','v1'
), ('fetch','v1'), ('get','v1'), ('give','v1'),
            ('go','v1'), ('hand','v1'), ('hit','v7'), ('hear','v1'), ('hold','v1'),('
help','v1'),
            ('hurry','v1'), ('kick','v1'),('keep','v1'), ('know','v1'), ('like','v1')
, ('love','v1'), ('make','v1'), ('meet','v1'),
            ('move','v1'), ('pick','v1'), ('remain','v1'), ('receive','v1'), ('
release','v1'), ('return','v1'), ('rise','v1'),
            ('run','v5'), ('see','v1'), ('sell','v1'), ('separate','v1'), ('sink','v1
'),('stay','v1'), ('sympathize','v1'),
            ('touch','v1'), ('take','v1'), ('tell','v1'), ('throw','v1'), ('want','v1
'), ('increase','v1'), ('decrease','v1'),
            ('withdraw','v1'),('thank','v1'),('need','v1'),('live','v1'),('return','
v1'),

            ('became','v2'), ('believed','v6'), ('brought','v6'), ('bought','v6'), ('
carried','v6'), ('caught','v6'), ('came','v2'),
            ('cried','v6'), ('drove','v2'), ('felt','v6'), ('fetched','v6'), ('got','
v6'), ('gave','v2'), ('gotten','v4'), ('went','v2'), ('gone','v4'),
            ('handed','v6'), ('heard','v6'), ('held','v6'), ('hurried','v6'), ('
kicked','v6'),('kept','v6'), ('knew','v2'), ('known','v4'),
            ('liked','v6'), ('loved','v6'), ('made','v6'), ('met','v6'), ('moved','v6
'), ('picked','v6'), ('remained','v6'),
            ('received','v6'), ('released','v6'), ('returned','v6'), ('rose','v2'), (
'risen','v4'), ('ran','v2'), ('saw','v2'),
            ('seen','v4'), ('sold','v6'), ('separated','v6'), ('sank','v2'), ('sunk',
'v4'), ('sympathized','v6'), ('touched','v6'),
            ('took','v2'), ('taken','v4'), ('told','v6'), ('threw','v2'), ('thrown','
v4'),('travel','v1'),('travelled','v6'),
            ('wanted','v6'),('increased','v6'), ('decreased','v6'),('driven','v4'),('
lived','v6'),

            ('do','vx'), ('did','vx'), ('done','v4'), ('be','vb'), ('is','vb'), ('am'
,'vb'), ('are','vb'), ('was','vb'), ('were','vb'),
            ('been','v4'), ('can','vx'), ('could','vx'), ('will','vx'), ('would','vx'
),('have','vx'), ('has','vx'), ('had','vx'),
            ('shall', 'vx'), ('should', 'vx'),

            ('robot','n'), ('human','n'), ('boy','n'), ('girl','n'), ('guy','n'), ('
man','n'), ('woman','n'), ('baby','n'),
            ('book','n'), ('table','n'), ('desk','n'), ('chair','n'), ('computer','n'
), ('cat','n'), ('rat','n'), ('dog','n'), ('fish','n'),
            ('bus','n'), ('school','n'), ('box','n'), ('pillar','n'), ('house','n'),
('home','n'), ('room','n'), ('left','n'),
            ('right','n'), ('front','n'), ('town','n'), ('university','n'), ('car','n
'),('phone','n'),('bed','n'),('bridge','n'),
            ('book_shelf','n'),('super_market','n'),('pet_shop','n'), ('bank','n'),('
voice','n'),('money','n'),('headache','n'),
            ('medicine','n'),('drug','n'),('drug_store','n'),('goldfish','n'),('water
','n'),('apartment','n'),('yard','n'),
            ('refrigerator','n'),('flower_shop','n'),('tulip','n'),('carnation','n'),
('paracetamol','n'),('hospital','n'),
            ('post_office','n'),('flower_bed','n'),('letter','n'),('swimming_pool','n
'),('playground','n'),('zoo','n'),('elephant','n'),('tv','n'),
            ('sofa','n'),('paravent','n'),('computer','n'),('computer_chair','n'),('
closet','n'),('vest','n'),('stove','n'),
            ('road','n'),('river','n'),('street','n'),('north','n'),('south','n'),('
restaurant','n'),('fountain','n'),('stair','n'),

            ('robby','pn'), ('tom','pn'), ('jim','pn'), ('mary','pn'), ('smith','pn')
, ('dan','pn'), ('brown','pn'), ('mr.','pn'),
            ('mz.','pn'),('miss','pn'), ('mrs.','pn'), ('ms.','pn'), ('fukuoka','pn')
, ('japan','pn'), ('bangkok','pn'), ('thailand','pn'),
            ('tokyo','pn'), ('osaka','pn'), ('story','pn'), ('city','pn'), ('i','pn')
, ('taro', 'pn'), ('anna','pn'),
            ('you','pn'), ('he','pn'), ('she','pn'), ('it','pn'), ('we','pn'), ('they
','pn'), ('my','pn'), ('me','pn'), ('mine','pn'),
            ('myself','pn'), ('your','pn'),('yours','pn'), ('yourself','pn'), ('him',
'pn'), ('his','pn'), ('himself','pn'), ('her','pn'),
            ('hers','pn'), ('herself','pn'), ('its','pn'), ('itself','pn'),('our','pn
'), ('us','pn'), ('ourselves','pn'), ('their','pn'),
            ('them','pn'), ('themselves','pn'),('shop_keeper','pn'),('bank_clerk','pn
'),('that','pn'), ('this','pn'),('everything','pn'),

            ('at','p'), ('into','p'), ('with','p'), ('within','p'), ('from','p'), ('
to','p'), ('along','p'),('across','p'),
            ('between','p'), ('among','p'), ('above','p'), ('under','p'), ('behind','
p'), ('beside','p'), ('through','p'), ('around','p'),
            ('in','p'), ('on','p'), ('of','p'),('by','p'), ('near','p'),('between','p
'),

            ('the','de'), ('a','de'), ('an','de'),

            ('red','aj'), ('green','aj'), ('blue','aj'), ('yellow','aj'), ('big','aj'
), ('round','aj'), ('triangle','aj'), ('cubic','aj'),
            ('hexahedral','aj'), ('high','aj'), ('low','aj'), ('long','aj'), ('short'
,'aj'), ('wide','aj'), ('narrow','aj'), ('deep','aj'), ('heavy','aj'),
            ('shallow','aj'), ('angry','aj'), ('jealous','aj'), ('happy','aj'), ('
unhappy','aj'), ('sad','aj'), ('small','aj'), ('little','aj'),
            ('fine','aj'), ('good','aj'),('gold','aj'),('sure','aj'),('hungry','aj'),

            ('very','ad'), ('not','ad'), ('no','ad'), ('yes','ad'), ('fast','ad'), ('
slowly','ad'), ('far','ad'), ('away','ad'), ('out','ad'),
            ('up','ad'), ('down','ad'), ('forward','ad'), ('backward','ad'), ('
leftward','ad'), ('rightward','ad'), ('quickly','ad'),
            ('upward','ad'), ('downward','ad'), ('forward','ad'), ('yesterday','ad'),
 ('today','ad'), ('tonight','ad'), ('everyday','ad'),
            ('tomorrow','ad'), ('there','ad'), ('here','ad'), ('please','ad'),

            ('and','cj'), ('or','cj'), ('not','cj'), ('if','cj'), ('because','cj'), (
'before','cj'), ('after','cj'), ('while','cj'), ('until','cj'),
            ('then','cj'),('.','mark'), ('?','mark'), ('!','mark'), (',','mark'),
            ('which','pn'), ('who','pn'), ('whom','cj'), ('what','pn'), ('why','pn'),
 ('where','pn'), ('when','cj'), ('whose','pn'), ('how','pn')]


    #===== Strat the Greeting message detection process =====
    '''greeting = []
    temp = ''
    #while (text == 'How are you?') or (text == 'How are you today?') or (text == '
how are you?') or (text == 'how is everything?')\
    #         or (text == 'How is everything?') or (text == 'Thank you.') or (text
 == 'thank you.'):
    while (text == 'Thank you.') or (text == 'thank you.') or (text == 'Thank you')
or (text == 'thank you'):
        print("\nAnna: You are welcome.")'''
    #===== end of Greeting message detection process =====


    text_1 = nltk.word_tokenize(text)           #word segment
    i = 0
    for w in text_1:
        if (w == 'some') or (w == 'any') or (w == 'several'):
            text_1.pop(i)
        for x in my_word:
            if (w == x[0]) and ('v' in x[1]):
                text_1[i] = en.verb.present(text_1[i])
            elif (w.endswith('ing') and w != 'morning' and w != 'evening') or (w.en
dswith('s')) or (w.endswith('es')):
                text_1[i] = en.verb.present(text_1[i])
        i+=1

    a = len(text_1)-1                           #word counting in text_1

    text_3 = []                                 #chang the words are be cut from
uppercase (text_1) to be lowercase (text_3)
    for a in text_1:
        text_2 = a.lower()
        text_3.append(text_2)

    #................ Check the input words with my dictionary......................
    b = 0
    c = 0
    text_4 = []
    while b <= len(text_3)-1:
        flag = 0
        while c <= len(my_word)-1:
            d = text_3[b]
            if(text_3[b] == my_word[c][0]):     #match type of words (text_3) using
dictionary (my_word)
                flag = 1                        #if flag = 1, have the word in
dictionary, but if flag = 0, don't have in dictionary
                if(my_word[c][1] == 'v2')or(my_word[c][1] == 'v4')or(my_word[c][1] ==
 'v5')or(my_word[c][1] == 'v6')or(my_word[c][1] == 'v7'):
                    text_2 = en.verb.present(my_word[c][0])     #Change the word to
be present tense if it is past tense.
                    text_4.append(text_2)
                else:
                    text_4.append(my_word[c][0])
            c+=1
        c=0
        b=b+1
        if flag == 0:
            text_4.append(d)

    #...................... Define input word type with my dictionary
...................
    b = 0
    c = 0
    text_2 = []
    text_3 = []
    while b <= len(text_4)-1:
        flag = 0
        while c <= len(my_word)-1:
            d = text_4[b]
            if(text_4[b] == my_word[c][0]):      # Check word, if there is the word in
 my dictionary, or not
                flag = 1
                text_2.append(my_word[c][1].upper())          # Define the word to be
 'V'
                text_3.append(text_2)
                text_2 = []
            c+=1
        c=0
        b=b+1
        if flag == 0:
            print('*'*30)
            print('\nThe word "%s" has no in dictionary, please define it\n'%d)
            d = raw_input("Please enter its word type: ")
            text_2.append(d.upper())                       #change the input to be c
apital letter
            text_3.append(text_2)
            text_2 = []

    #.....................Change V1, V2, V4, V5, V6 and V7 to be V
.....................................
    b = 0
    while b <= len(text_3)-1:
        a = text_3[b]
        if (a == ['V1'])or(a == ['V2'])or(a == ['V4'])or(a == ['V5'])or(a == ['V6'])o
r(a == ['V7'])or(a == ['VB'])or(a == ['VX']):
            text_3[b] = ['V']
        b+=1

    b = 0
    text_3_new = []
    while b <= len(text_3)-1:
        a = text_3[b][0]
        text_3_new.append(a)
        b+=1
    sen = ' '.join(text_4)

    #------------ Matching word and its word type------------
    b = 0
    sentence = []
    while b <= len(text_4)-1:
        a = (text_4[b],text_3_new[b])
        sentence.append(a)
        b+=1

    #Split the input sentence into sub-sentences (that is sentence1 and sentence2)
    v = 0
    cj = 0
    comma = 0
    fullstop = 0
    question_mark = 0
    cj_w = []
    for w in sentence:
        if 'V' in w[1]:
            v += 1
        elif 'CJ' in w[1]:
            cj += 1
            cj_w.append(w)
        elif ',' in w[0]:
            comma += 1
        elif ('.' in w[0]):
            fullstop+=1
        elif ('?' in w[0]):
            question_mark +=1

    i =0; j = 0; k = 0
    sentence_new = []
    if (fullstop >= 1):
        for w in sentence:
            j+=1
            if ('.' in w):
                sentence_new.append(sentence[i:j-1])
                i = j
    elif (question_mark >= 1):
        for w in sentence:
            j+=1
            if ('?' in w):
                sentence_new.append(sentence[i:j-1])
                i = j
    elif (comma >= 1):
        for w in sentence:
            j+=1
            if ',' in w:
                sentence_new.append(sentence[i:j-1])
                print(sentence[i:j-1])
                i = j
            elif '.' in w:
                sentence_new.append(sentence[i:len(sentence)-1])
                print(sentence[i:len(sentence)-1])
            elif '?' in w:
                sentence_new.append(sentence[i:len(sentence)-1])
                print(sentence[i:len(sentence)-1])
    elif ((v > 1) and (cj == 0)):
        for w in sentence:
            j+=1
            if 'V' in w:
                k+=1
                if k > 1:
                    sentence_new.append(sentence[i:j-1])
                    i = j
            if '.' in w:
                sentence_new.append(sentence[i-1:len(sentence)-1])
    elif (cj > 0) and (v > 1) and (fullstop == 1):
        for w in sentence:
            j+=1
            if 'CJ' in w:
                sentence_new.append(sentence[i:j-1])
                i = j
            if '.' in w:
                sentence_new.append(sentence[i:len(sentence)-1])
    else:
        sentence_new.append(sentence)
    #=====================================================================
==================================


    #--- For example:- 'I am going to school with mary' ---> 'I am with mary going to
```

```
 school . ---
    i = 100; j = 100; k = 100; a = 0; b = 0; temp = []; l = 100; m =100; n =100
    for w in sentence_new:
        for ww in w:
            if ('V' in ww) and ('be' in ww):
                i = w.index(ww)
            elif ('V' in ww) and ('be' not in ww) and (w.index(ww) == k+1):
                i = w.index(ww)
                b+=1
            elif ('with' in ww) and (cj_w == []):
                j = w.index(ww)
            elif ('with' in ww) and (cj_w != []):
                l = w.index(ww)
            elif ('PN' in ww) and (k == 100):
                k = w.index(ww)
            elif ('PN' in ww) and (k != 100) and (cj_w != []):
                k = w.index(ww)
            elif ('DE' in ww) and (w.index(ww) == j+1):
                a += 1
            elif ('DE' in ww) and (l != 100):
                a +=1

        if (i<j) and (j!=100) and (k!=100) and (i == k+1):
            temp.append(w[j])
            temp.append(w[j+1])
            if (a != 0):
                temp.append(w[j+2])
                w.pop(j+2)
            w.pop(j+1)
            w.pop(j)
        elif (i<l) and (cj_w != []) and (k!=100) and (i == k+1):
            temp.append(w[l])
            temp.append(w[l+1])
            if (a != 0):
                temp.append(w[l+2])
                w.pop(l+2)
            w.pop(l+1)
            w.pop(l)

        if temp != []:
            if (b == 0):
                for x in temp:
                    w.insert(i+1,x)
                    i+=1
            else:
                for x in temp:
                    w.insert(k+1,x)
                    k+=1
        temp = []; i = 100; j = 100; k = 100; b = 0; a = 0
    #--- End of changing the position of words. ---

    # Split words and words' type.
    temp = ""; temp_1 = ""; A_1 = []; A_2 = []
    for w in sentence_new:
        for x in w:
            temp = temp+x[0]+' '        # A_1 keeps only word in sentences, for examp
le: Tom is with the book in the bus
            temp_1 = temp_1+x[1]+" "    # A_2 keeps only words' type, for example: PN
 V P DE N P DE N
        A_1.append(temp)
        A_2.append(temp_1)
        temp = ""; temp_1 = ""
    #================================================================================
===================

    # Delete space after each phrase (for A_1).
    i = 0; temp = ''; temp_1 = []
    for w in A_1:
        i = len(w)-1
        if (w[i] == ' '):
            temp = w[:-1]
        else:
            temp = w
        temp_1.append(temp)
    A_1 = temp_1

    # Delete space after each phrase (for A_2).
    i = 0; temp = ''; temp_1 = []
    for w in A_2:
        i = len(w)-1
        if (w[i] == ' '):
            temp = w[:-1]
        else:
            temp = w
        temp_1.append(temp)
    A_2 = temp_1
    #================================================================================
===================

    #Write sentences into phrases and clauses in tree form (Phrase structure tree)
    grammar_1 = r"""
        NP: {<PN>|<DE>?<AD>*<AJ>*<N>}
        PP: {<P><NP>|<P><AD>|<P><PN>}
        VP: {<VX>?<V>*}
        ADP: {<AD>*}
        AJP: {<AD>*<AJ>*}
        CJW: {<CJ>?}
        CLAUSE: {<NP>*<VP><ADP>*<PP>*<NP>*}"""
    cp = nltk.RegexpParser(grammar_1)
    result = []
    for w in sentence_new:
        result.append(cp.parse(w))
    #=============================================================================
=======================

    # If sentence has 'N' as subject, change it to be 'PN'.
    # Change 'PN' to be 'N' if it is not subject of the sentence.
    i = 100; j = 100
    for w in sentence_new:
        for ww in w:
            if (ww[1] == 'N') and (w.index(ww) == 0):
                i = w.index(ww)
                w[i] = (ww[0],'PN')
            elif (ww[1] == 'DE') and (w.index(ww) == 0):
                i = w.index(ww)
            elif (ww[1] == 'N') and (w.index(ww) == i+1):
                i = w.index(ww)
                w[i] = (ww[0],'PN')
            elif (ww[1] == 'CJ'):
                j = w.index(ww)
            elif (ww[1] == 'PN') and (w.index(ww) != 0) and (w.index(ww) != j+1):
                i = w.index(ww)
                w[i] = (ww[0],'N_P')     #Change from 'PN' to 'N'
                j = 100
        i = 100; j = 100

    # Extract to get just only Key words: PN, V, P, and Adverb.
    sent = ""
    temp = []; temp_1 = []
    remain_word = []
    remain_all_DT = []
    i = 0
    for w in sentence_new:
        for x in w:
            if x[1] == 'PN'or x[1] == 'V' or x[1] == 'P' or x[1] == 'AD' or x[1] == '
AJ' or x[1] == 'CJ':
                sent = sent + x[1] + ' '
                temp.append(x[0])
                temp_1.append(x[1])
        remain_word.append(temp)
        remain_all_DT.append(temp_1)
        temp = []; temp_1 = []; sent = ""; i+=1
    #===========================================================================
=========================

    # Extract indexes of keywords in sentence.
    temp = []; i = []; j = 0; k = 0; l = 0
    for w in A_1:
        AA_1 = w.split()
        for ww in AA_1:
            for x in remain_word[k]:
                if ww == x:
                    if j not in temp:
                        temp.append(j)
            j+=1
        i.append(temp)
        k+=1
        j=0
        temp = []
    temp = i

    # Because each clause is String, so I have to change it to be List form.
    temp_1 = []
    for w in A_1:
        temp_1.append(w.split())

    #--- Extract phrases from clauses----
    i = []; j = 0; k = 0; l = 0; temp_2 = []; remain_phrase = []
    for w in temp:
        temp_2 = temp_1[j]
        for ww in w:
            k = w.index(ww)
            if w.index(ww) < (len(w)-1):
                l = w[k+1]
                i.append(temp_2[ww:l])
            else:
                i.append(temp_2[ww:])
        remain_phrase.append(i)
        i = []; temp_2 = []; j+=1

    # Remove 'a', 'an', 'the' from phrases.
    for w in remain_phrase:
        for y in w:
            if len(y) > 1:
                for z in y:
                    if z == 'a' or z == 'an' or z == 'the':
                        y.remove(z)         # Delete determinant from phrases
    #--- End of extracting phrases ---
    #=================================================================================
=====================

    #--- Extract Data Type from clauses----
    temp_1 = []
    for w in A_2:
        AA_2 = w.split()
        temp_1.append(AA_2)

    i = []; j = 0; k = 0; l = 0; temp_2 = []; remain_DT = []
    for w in temp:
        temp_2 = temp_1[j]
        for ww in w:
            k = w.index(ww)
            if w.index(ww) < (len(w)-1):
                l = w[k+1]
                i.append(temp_2[ww:l])
            else:
                i.append(temp_2[ww:])
        remain_DT.append(i)
        i = []; temp_2 = []; j+=1

    # Remove 'DE' from phrases.
    for w in remain_DT:
        for y in w:
            if len(y) > 1:
                for z in y:
                    if z == 'DE':
                        y.remove(z)         # Delete determinant from phrases
    #--- Extract phrases from clauses----

    ###===== Change back from 'N_P' to 'PN' =====
    for w in sentence_new:
        for ww in w:
            if (ww[1] == 'N_P'):
                i = w.index(ww)
                w[i] = (ww[0],'PN') #Change back from 'N_P' to 'PN'
    #===== End of changing back from 'N_P' to 'PN' =====

    return (sentence_new,cj_w,A_1,A_2,remain_word,remain_all_DT,temp,remain_phrase,
remain_DT)
```

D.6 extractphrase.py

```python
from nltk.tree import *
def ExtractPhrases( myTree, phrase):
    myPhrases = []
    if (myTree.label() == phrase):
        myPhrases.append(myTree.copy(True))
    for child in myTree:
        if (type(child) is Tree):
            list_of_phrases = ExtractPhrases(child, phrase)
            if (len(list_of_phrases) > 0):
                myPhrases.extend(list_of_phrases)
    return myPhrases
```

D.7 inference.py

```
#===== This part is for apply the inference rules to Lmd expression =====

def Inference(final_locus):

    temp = ''; temp1 = []
    for w in final_locus:
        if len(w) > 1:
            for ww in w:
                if (ww != ['sand']) and (ww != ['cand']):
                    temp = ww
                    for x in w:
                        if (x != ['sand']) and (x != ['cand']) and (x != temp):
                            if (len(temp) == 8) and (len(x) == 8) and (temp[2] == x[1
]) and (temp[3] == x[1]) and (temp[4] == 'a') and (x[4] == 'a'):
                                temp1.append(temp[0])
                                temp1.append(temp[1])
                                temp1.append(x[2])
                                temp1.append(x[3])
                                temp1.append(temp[4])
                                temp1.append(temp[5])
                                temp1.append(temp[6])
                                temp1.append(temp[7])
                                if temp1 not in w:
                                    w.append(['sand'])
                                    w.append(temp1)
                                temp1 = []

                            elif (len(temp) == 8) and (len(x) == 8) and (temp[1] == x
[2]) and (temp[1] == x[3]) and (temp[4] == 'a') and (x[4] == 'a'):
                                temp1.append(x[0])
                                temp1.append(x[1])
                                temp1.append(temp[2])
                                temp1.append(temp[3])
                                temp1.append(x[4])
                                temp1.append(x[5])
                                temp1.append(x[6])
                                temp1.append(x[7])
                                if temp1 not in w:
                                    w.append(['sand'])
                                    w.append(temp1)
                                temp1 = []

                            elif (len(temp) != 8) and (len(x) == 8):
                                for y in temp:
                                    if (y[2] == x[1]) and (y[3] == x[1]) and (y[4] ==
 'a') and (x[4] == 'a'):
                                        temp1.append(y[0])
                                        temp1.append(y[1])
                                        temp1.append(x[2])
                                        temp1.append(x[3])
                                        temp1.append(y[4])
                                        temp1.append(y[5])
                                        temp1.append(y[6])
                                        temp1.append(y[7])
                                        if temp1 not in w:
                                            w.append(['sand'])
                                            w.append(temp1)
                                        temp1 = []

                                    elif (y[1] == x[2]) and (y[1] == x[3]) and (y[4
] == 'a') and (x[4] == 'a'):
```

```
                                        temp1.append(x[0])
                                        temp1.append(x[1])
                                        temp1.append(y[2])
                                        temp1.append(y[3])
                                        temp1.append(x[4])
                                        temp1.append(x[5])
                                        temp1.append(x[6])
                                        temp1.append(x[7])
                                        if temp1 not in w:
                                            w.append(['sand'])
                                            w.append(temp1)
                                        temp1 = []

                            elif (len(temp) == 8) and (len(x) != 8):
                                for y in x:
                                    if (temp[2] == y[1]) and (temp[3] == y[1]) and (
temp[4] == 'a') and (y[4] == 'a'):
                                        temp1.append(temp[0])
                                        temp1.append(temp[1])
                                        temp1.append(y[2])
                                        temp1.append(y[3])
                                        temp1.append(temp[4])
                                        temp1.append(temp[5])
                                        temp1.append(temp[6])
                                        temp1.append(temp[7])
                                        if temp1 not in w:
                                            w.append(['sand'])
                                            w.append(temp1)
                                        temp1 = []

                                    elif (temp[1] == y[2]) and (temp[1] == y[3]) and
(temp[4] == 'a') and (y[4] == 'a'):
                                        temp1.append(y[0])
                                        temp1.append(y[1])
                                        temp1.append(temp[2])
                                        temp1.append(temp[3])
                                        temp1.append(y[4])
                                        temp1.append(y[5])
                                        temp1.append(y[6])
                                        temp1.append(y[7])
                                        if temp1 not in w:
                                            w.append(['sand'])
                                            w.append(temp1)
                                        temp1 = []

    temp = []; temp1 = []
    for w in final_locus:
        for ww in w:
            if ww == ['sand'] or ww == ['cand']:
                temp.append(ww)

        for ww in w:
            if ww != ['sand'] and ww != ['cand']:
                temp.append(ww)
        temp1.append(temp)
        temp = []

    return temp1
    #----- End of inference rule. -----
```

D.8 move_anna_taro.py

```
# This is used for moving 'Anna' and 'Taro' turtles from one place to another place.
import turtle
from turtle import *
from connectionposition import ConnectionPosition
from setpositionofplaces import SetPositionofPlaces
from mindistance import MinDistance

def MoveAnnaTaro(temp1,place_position,anna_start_position,taro_start_position,
taro_anna_start_position):

    #--- Extract the moved turtle(s) and place(s) from Lmd ---
    mover = []; place = []; move_to = []; temp = []
    for w in temp1:
        for ww in w:
            if len(ww) == 7:
                if ww[3] in place_position:     # Check if the target place is
outside home.
                    if ww[0] not in mover:
                        mover.append(ww[0])
                    if ww[1] not in mover:
                        mover.append(ww[1])
                    if ww[3] not in place:
                        place.append(ww[3])
        if (mover != []) and (place != []):
            move_to.append(mover)
            move_to.append(place)
            temp.append(move_to)
            mover = []; place = []; move_to = []
    #--- End to extract moved turtle(s) and place(s) ---
    return temp
```

D.9 nlexpression_to_lmd.py

```
def NLExpressionToLMD(remain_phrase):
    import copy

    #===== Locus Formula =====
    taro_turtle = ['taro_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','
none']]      # The first member is Entity name, and last 2 members are governor and
dependant (unification)
    anna_turtle = ['anna_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','
none']]     # 'a' refers to physical location
    shop_keeper = ['shop_keeper_',[['sand'],['x','x','p','q','a','pn','+']],['x','
none','none']]
    bank_clerk = ['bank_clerk_',['sand'],[['x','x','p','q','a','pn','+']],['x','none'
,'none']]
    tom = ['tom_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    mary = ['mary_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    he = ['he_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    she = ['she_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    i = ['i_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    me = ['me_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    you = ['you_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    it = ['it_',['sand'],[['x','x','p','q','a','pn','+']],['x','none','none']]
    everything = ['everything_',['sand'],[['x','x','p','q','a','pn','+']],['x','none'
,'none']]

    phone = ['phone_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    book = ['book_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    box = ['box_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    bus = ['bus_',['sand'],[['x','x','p','q','a','n','+']],['x','none','none']]
    car = ['car_',['sand'],[['x','x','p','q','a','n','+']],['x','none','none']]
    flower_shop = ['flower_shop_',['sand'],[['x','x','p','q','a','n','-']],['x','none
','none']]
    drug_store = ['drug_store_',['sand'],[['x','x','p','q','a','n','-']],['x','none',
'none']]
    hospital = ['hospital_',['sand'],[['x','x','p','q','a','n','-']],['x','none','
none']]
    post_office = ['post_office_',['sand'],[['x','x','p','q','a','n','-']],['x','none
','none']]
    swimming_pool = ['swimming_pool_',['sand'],[['x','x','p','q','a','n','-']],['x','
none','none']]
    playground = ['playground_',['sand'],[['x','x','p','q','a','n','-']],['x','none',
'none']]
    super_market = ['super_market_',['sand'],[['x','x','p','q','a','n','-']],['x','
none','none']]
    pet_shop = ['pet_shop_',['sand'],[['x','x','p','q','a','n','-']],['x','none','
none']]
    bank = ['bank_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    home = ['home_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    house = ['house_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    apartment = ['apartment_',['sand'],[['x','x','p','q','a','n','-']],['x','none','
none']]
    restaurant = ['restaurant_',['sand'],[['x','x','p','q','a','n','-']],['x','none',
'none']]
    zoo = ['zoo_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    yard = ['yard_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    bridge = ['bridge_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    stair = ['stair_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    flower_bed = ['flower_bed_', ['sand'], [['x', 'x', 'p', 'q', 'a', 'n', '-']], ['x
', 'none', 'none']]
    fountain = ['fountain_',['sand'],[['x','x','p','q','a','n','-']],['x','none','
none']]
```

```
    road = ['road_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    street = ['street_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    river = ['river_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    town = ['town_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    university = ['university_',['sand'],[['x','x','p','q','a','n','-']],['x','none',
'none']]
    school = ['school_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    tv = ['tv_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    bed = ['bed_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    table = ['table_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    chair = ['chair_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    refrigerator = ['refrigerator_',['sand'],[['x','x','p','q','a','n','-']],['x','
none','none']]
    sofa = ['sofa_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    paravent = ['paravent_',['sand'],[['x','x','p','q','a','n','-']],['x','none','
none']]
    computer = ['computer_',['sand'],[['x','x','p','q','a','n','-']],['x','none','
none']]
    computer_chair = ['computer_chair_',['sand'],[['x','x','p','q','a','n','-']],['x'
,'none','none']]
    closet = ['closet_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    vest = ['vest_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    stove = ['stove_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    book_shelf = ['book_shelf_',['sand'],[['x','x','p','q','a','n','-']],['x','none',
'none']]
    tulip = ['tulip_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    carnation = ['carnation_',['sand'],[['x','x','p','q','a','n','-']],['x','none','
none']]
    paracetamol = ['paracetamol_',['sand'],[['x','x','p','q','a','n','-']],['x','none
','none']]
    letter = ['letter_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    voice = ['voice_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    money = ['money_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    medicine = ['medicine_',['sand'],[['x','x','p','q','a','n','-']],['x','none','non
e']]
    drug = ['drug_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]
    goldfish = ['goldfish_',['sand'],[['x','x','p','q','a','n','+']],['x','none','non
e']]
    fish = ['fis_h',['sand'],[['x','x','p','q','a','n','+']],['x','none','none']]
    dog = ['dog_',['sand'],[['x','x','p','q','a','n','+']],['x','none','none']]
    elephant = ['elephant_',['sand'],[['x','x','p','q','a','n','+']],['x','none','non
e']]
    water = ['water_',['sand'],[['x','x','p','q','a','n','-']],['x','none','none']]

    who = ['who_',['sand'],[['x','x','who','who','people','pn?','+']],['x','none','no
ne']]
    what = ['what_',['sand'],[['x','x','what','what','thing','pn?','+']],['x','none
','none']]
    where = ['where_',['sand'],[['x','x','where','where','a','pn?','+']],['x','none
','none']]
    when = ['when_',['sand'],[['x','x','when','when','time','pn?','+']],['x','none','
none']]
    how = ['how_',['sand'],[['how','x','p','q','a','pn?','+'],['x','x','how','how','a
','pn?','+']],['x','none','none']]

    there = ['there_',['sand'],[['x','y','there','there','a','ad','-']],['x','y','no
ne']]
    here = ['here_',['sand'],[['x','x','here','here','a','ad','-']],['x','none','here
']]
    please = ['please_',['sand'],[['x','x','pleased','pleased','happiness','ad','+
']],['x','none','none']]
```

```
    fine = ['fine_',['sand'],[['x','x','fine','fine','happiness','aj','+']],['x','no
ne','none']]
    good = ['good_',['sand'],[['x','x','good','good','happiness','aj','+']],['x','no
ne','none']]
    hungry = ['hungry_',['sand'],[['x','x','hungry','hungry','happiness','aj','-']],[
'x','none','none']]
    red = ['red_',['sand'],[['x','x','red','red','color','aj','+']],['x','none','none
']]
    gold = ['gold_',['sand'],[['x','x','gold','gold','color','aj','+']],['x','none','
none']]

    be = ['be_',['sand'],[['x','x','p','q','a','v','+']],['x','x','none']]
    do = ['do_',['sand'],[['x','x','p','q','a','v','+']],['x','x','none']]
    can = ['can_',['sand'],[['x','x','p','q','a','v','+']],['x','x','none']]
    run = ['run_',['sand'],[['x','x','p','q','a','v','+']],['x','none','none']]
    travel = ['travel_',['sand'],[['x','x','p','q','a','v','+']],['x','none','none']]
    go = ['go_',['sand'],[['x','x','p','q','a','v','+']],['x','none','none']]
    love = ['love_',['sand'],[['x','y','p','q','happiness','v','+'],['y','x','p','q',
'happiness','v','+']],['x','y','none']]
    like = ['like_',['sand'],[['x','x','p','q','happiness','v','+']],['x','none','n
one']]
    paint = ['paint_',['sand'],[['x','y','z','z','color','v','+']],['x','y','z']]
    buy = ['buy_',['sand'],[[['y','x'],'z','y','x','a','v','+',] ,[['y','x'],'money',
'x','y','a','v','+']],['x','y','z']] # [x,y] refers to the cooperation between two
people.
    give = ['give_',['sand'],[['x','z','x','y','a','v','+']],['x','y','z']]    # Tom(
=x) give Mary(=y) a book(=z).
    withdraw = ['withdraw_',['sand'],[[['x','y'],'money','x','y','a','v','+']],['x','
y','none']]  # Transportation of real money
    call = ['call_',['sand'],[['x','phone','x','x','a','v','+'],['x','voice','x','y',
'a','v','+']],['x','y','none']]
    carry_1 = ['carry_',['sand'],[['x','y','x','x','a','v','+'],['x','x','p','q','a',
'v','+']],['x','y','none']]
    carry_2 = ['carry_',['sand'],[['x','x','p','q','a','v','+'],['x','y','p','q','a',
'v','+']],['x','y','none']]
    drive = ['drive_',['sand'],[['x','x','y','y','a','v','+'],['x','y','p','q','a','v
','+']],['x','y','none']] # x = Controller, y = someone/something is controlled by x
    move = ['move_',['sand'],[['x','y','p','q','a','v','+']],['x','y','none']]
    keep = ['keep_',['sand'],[['x','y','p','q','a','v','+']],['x','y','none']]
    want = ['want_',['sand'],[['x','y','x','x','a','v','+'],['y','x','p','q','happine
ss','v','+']],['x','y','none']]
    take = ['take_',['sand'],[['x','y','p','q','a','v','+']],['x','y','none']]
    return_z = ['return_',['cand'],[['x','x','p','q','a','v','+'],['x','x','q','p','a
','v','+']],['x','p','none']]
    fetch_z = ['fetch_',['cand','sand'],[['x','x','p','q','a','v','+'],['x','x','q','
p','a','v','+'],['x','y','x','x','a','v','+']],['x','p','none']]

    with_z = ['with_',['sand'],[['x','y','x','x','a','pp','+']],['x','y','none']]
    at = ['at_',['sand'],[['x','x','p','p','a','pp','+']],['x','p','none']]
    in_z = ['in_',['sand'],[['x','x','p','p','a','pp','+']],['x','p','none']]    # Fo
r example: Tom in bus = [Tom,Tom,bus,bus,a]
    on = ['on_',['sand'],[['x','x','p','p','a','pp','+']],['x','p','none']]
    by = ['by_',['sand'],[['x','y','p','q','a','pp','+']],['y','x','none']]
    to = ['to_',['sand'],[['x','x','p','q','a','pp','+']],['x','q','none']]     # Fo
r example: to school = [x,x,p,school,a]
    from_z = ['from_',['sand'],[['x','x','p','q','a','pp','+']],['x','p','none']]  #
For example: from town = [x,x,town,q,a]
    along = ['along_',['sand'],[['x','x','p','p','direction','pp','+']],['x','p','non
e']]
    north = ['north_',['sand'],[['x','x','north','north','direction','pp','-']],['x
','none','none']]
```

```
    south = ['south_',['sand'],[['x','x','south','south','direction','pp','-']],['x
','none','none']]

    All_Locus = [taro_turtle,anna_turtle,shop_keeper,bank_clerk,tom,mary,he,she,i,me,
it,who,what,where,how,drug_store,
                 phone,book,box,bus,car,bed,book_shelf ,super_market,pet_shop,bank,
home,town,university,school,voice,money,
                 medicine,drug,goldfish,fish,dog,water,there,here,fine,good,red,gold,
hungry,buy,everything,
                 withdraw,give,call,carry_1,carry_2,drive,run,move,travel,go,keep,lov
e,like,want,take,with_z,at,in_z,on,
                 by,to,from_z,table,chair,refrigerator,flower_shop,tulip,carnation,
paracetamol,hospital,post_office,letter,
                 swimming_pool,playground,zoo,elephant,tv,sofa,paravent,computer,
computer_chair,closet,vest,stove,be,please,road,river,
                 stair,yard,fountain,apartment,street,along,north,south,restaurant,
bridge,do,you,can,house,flower_bed]
    #--- End of locus formula ---


    #===== Change natural expression to be logical terms =====
    temp = []; temp1 = []; temp2 = []
    for w in remain_phrase:
        for ww in w:
            if len(ww) == 1:
                for www in ww:
                    if not www[-1].isdigit():
                        i = remain_phrase.index(w)
                        if i not in temp:
                            temp.append(i)
                        for x in All_Locus:
                            if www == x[0][:-1]:
                                temp.append(x)
                        temp1.append(temp)
                        temp = []
                    else:
                        i = remain_phrase.index(w)
                        if i not in temp:
                            temp.append(i)
                        for x in All_Locus:
                            if www[:-1] == x[0][:-1]:
                                temp.append(x)
                        temp1.append(temp)
                        temp = []
            else:
                for www in ww:
                    if not www[-1].isdigit():
                        i = remain_phrase.index(w)
                        if i not in temp:
                            temp.append(i)
                        for x in All_Locus:
                            if www == x[0][:-1]:
                                temp.append(x)
                    else:
                        i = remain_phrase.index(w)
                        if i not in temp:
                            temp.append(i)
                        for x in All_Locus:
                            if www[:-1] == x[0][:-1]:
                                temp.append(x)
                temp1.append(temp)
                temp = []
        temp2.append(temp1)
        temp1 = []
    #===== End of changing NL expression to be logical term =====

    #===== Start to make the difference, in case of there are morn than 1 input
sentence =====
    temp = []; temp1 = []
    for w in temp2:
        i = temp2.index(w)
        for new_ww in w:
            ww = copy.deepcopy(new_ww)      # .deepcopy is used to copy every object
in the variable.
            if (len(ww) == 2) and (ww[0] == i):
                if len(ww[1][2]) == 1:
                    if len(ww[1][2][0]) == 7:
                        ww[1][2][0].insert(7,i)
                    else:
                        if ww[1][2][0][7] != i:
                            ww[1][2][0].pop(7)
                            ww[1][2][0].insert(7,i)
                    temp.append(ww)

                else:
                    for x in ww[1][2]:
                        if len(x) == 7:
                            x.insert(7,i)
                        else:
                            if x[7] != i:
                                x.pop(7)
                                x.insert(7,i)
                    temp.append(ww)

            elif (len(ww) > 2) and (ww[0] == i):
                j = 1
                while j <= len(ww)-1:
                    for y in ww[j][2]:
                        if len(y) == 7:
                            y.insert(7,i)
                        else:
                            if y[7] != i:
                                y.pop(7)
                                y.insert(7,i)
                    j+=1
                temp.append(ww)
        temp1.append(temp)
        temp = []
    #===== End of making the difference =====

    for w in temp1:
        for ww in w:
            ww.pop(0)
    return temp1
    #===== Finish to change natural expression to be logical terms =====
```

D.10 call_phone.py

```
# This function is for detecting 'call phone'.
import turtle
from turtle import *
from home_connectionposition import HomeConnectionPosition
from home_placeposition import HomePlacePosition
from my_dictionary import my_dictionary
from nlexpression_to_lmd import NLExpressionToLMD

def CallPhone(temp1):

    #--- Locus_Extra_Attribute ---
    taro = [['taro','voice','taro','q','a','pn','+'],['taro','phone','p','p','a','pn'
,'+'],
            ['x','taro','p','q','happiness','pn','+'],['taro','taro','male','male','
sex','pn','+']]
    anna = [['x','anna','p','q','a','pn','+'],['anna','anna','p','q','happiness','pn'
,'+'],
            ['anna','anna','female','female','sex','pn','+']]
    tom = ['tom','tom','male','male','sex','pn','+']
    mary = ['mary','mary','female','female','sex','pn','+']

    Locus_Extra_Attribute = [taro,anna,tom,mary]
    #--- End of Extra locus ---

    #--- Detect for the important word, then keep 'it' and "its" lmd(s) in 'temp' and
 "temp_3". ---
    temp = []; temp_3 = []
    for w in temp1:
        for ww in w:
            if (ww != ['sand']) and (ww != ['cand']):
                if ('phone' in ww) or ('voice' in ww):
                    if (ww[2] == ww[3]) or (ww[0] == ww[2]):
                        if ww[2] == 'i':
                            temp = 'taro'
                        if ww not in temp_3:
                            temp_3.append(ww)      # Keep important lmd in temp3

    if temp == 'taro':
        i = 0
        for w in temp_3:
            for ww in w:
                if ww == 'i':
                    i = w.index(ww)
                    w.pop(i)
                    w.insert(i,temp)
    #--- Finish to detect important keyword and its lmd(s) ---

    #--- Extract necessary locus(es) from Locus_Extra_Attribute ---
    temp_2 = []
    for w in Locus_Extra_Attribute:
        for ww in w:
            if temp in ww:
                temp_2.append(ww)       # select locus formula from
Locus_Extra_Attribute
    #--- End of extraction of necessary loci ---

    #--- Classify the advantage lmd(s) in temp_2 ---
    temp = []
    if (temp_2 != []) and (temp_3 != []):
        for w in temp_3:    # temp_3 keeps the key locus
            for x in temp_2:    #temp_2 keeps attributes of a person in
```

```
Locus_extra_attribute
                if w[1] in x:   # if keywords(i.e. 'phone' and 'voice') in temp_2
                    temp.append(x)

    send_back_1 = temp
    send_back_2 = temp_3
    #--- Finish to classify the advantage lmd ---

    #--- Change NL expression to be Lmd ---
    temp_2 = []; tempp = ''; temp_4 = []
    for w in temp:
        for x in temp_3:
            if w[1] == x[1]:
                if (w[2] == w[3]) and (x[2] == x[3]) and (w[2] != (x[2])):  # In case
 of 'phone'
                    print ("Anna: Where is the %s?"%w[1])
                    tempp = w[1]
                    text = raw_input("\nYour command: ")
                    sentence = my_dictionary(text)
                    sentence_new = sentence[0]
                    cj_w = sentence[1]
                    A_1 = sentence[2]
                    A_2 = sentence[3]
                    remain_word = sentence[4]
                    remain_all_DT = sentence[5]
                    index_of_keyword = sentence[6]
                    remain_phrase = sentence[7]
                    remain_DT = sentence[8]
                    temp_2 = NLExpressionToLMD(remain_phrase)
                    for y in temp_2:
                        for yy in y:
                            for yyy in yy:
                                temp_4.append(yyy)
    #--- End of changing NL expression to be logical form ---

    #===== Combine 'PP' and 'N' term =====
    # ===== Combine 'PP' and 'AJ' terms =====
    i = 100; temp = ''; temp1 = ''; temp2 = ''; j = 100
    for w in temp_4:
        for ww in w:
            if (type(ww) == list) and (len(ww[0]) == 8):
                if (ww[0][5] == 'pp'):
                    i = temp_4.index(w)
                    temp = w[3][0]
                elif (ww[0][5] == 'n') and (temp_4.index(w) == i + 1):
                    temp1 = w[0][:-1]
                    for x in temp_4[i][2]:
                        for xx in x:
                            if (x.index(xx) == 2) and (xx == 'p'):
                                x.pop(2)
                                x.insert(2, temp1)
                            elif (x.index(xx) == 3) and ((xx == 'q') or (xx == 'p')):
                                x.pop(3)
                                x.insert(3, temp1)
                            elif (xx == temp):
                                j = x.index(xx)
                                x.pop(j)
                                x.insert(j,tempp)
                    temp_4[i][3].pop(1)
                    temp_4[i][3].insert(1, temp1)
                    temp_4[i][3].pop(0)
                temp_4[i][3].insert(0,tempp)
                temp1 = temp_4[i][0] + "_" + w[0][:-1]
                temp_4[i].pop(0)
                temp_4[i].insert(0, temp1)
                i = 100
                temp_4.remove(w)
# ===== End of combination 'PP' and 'N' terms =====
#=====================================

return (temp_4,send_back_1,send_back_2)
```

D.11 home_placeposition.py

```
#===== Create turtles as things' position in the house =====
import turtle
from turtle import *


def HomePlacePosition():

    place_position = {}
    #connection_position = {}

    #===== Set position of the shops and places =====
    table = turtle.Turtle()
    table.shape('circle')
    table.turtlesize(0.01)
    table.color('green')
    table.penup()
    table.setposition(-175.00, -150.00)
    place_position['table'] = table.position()

    gold_fish_jar = turtle.Turtle()
    gold_fish_jar.shape('circle')
    gold_fish_jar.turtlesize(0.01)
    gold_fish_jar.color('green')
    gold_fish_jar.penup()
    gold_fish_jar.setposition(-175.00, -110.00)
    place_position['gold_fish_jar'] = gold_fish_jar.position()

    chair = turtle.Turtle()
    chair.shape('circle')
    chair.turtlesize(0.01)
    chair.color('green')
    chair.penup()
    chair.setposition(-275.00, -150.00)
    place_position['chair'] = chair.position()

    paravent = turtle.Turtle()
    paravent.shape('circle')
    paravent.turtlesize(0.01)
    paravent.color('green')
    paravent.penup()
    paravent.setposition(-175.00, 0.00)
    place_position['paravent'] = paravent.position()

    sofa = turtle.Turtle()
    sofa.shape('circle')
    sofa.turtlesize(0.01)
    sofa.color('green')
    sofa.penup()
    sofa.setposition(-175.00, 110.00)
    place_position['sofa'] = sofa.position()

    tv = turtle.Turtle()
    tv.shape('circle')
    tv.turtlesize(0.01)
    tv.color('green')
    tv.penup()
    tv.setposition(-175.00, 190.00)
    place_position['tv'] = tv.position()

    bed_room = turtle.Turtle()
    bed_room.shape('circle')
bed_room.turtlesize(0.01)
bed_room.color('green')
bed_room.penup()
bed_room.setposition(80.00, 60.00)
place_position['bed_room'] = bed_room.position()


bed = turtle.Turtle()
bed.shape('circle')
bed.turtlesize(0.01)
bed.color('green')
bed.penup()
bed.setposition(130.00, 160.00)
place_position['bed'] = bed.position()

pillow = turtle.Turtle()
pillow.shape('circle')
pillow.turtlesize(0.01)
pillow.color('green')
pillow.penup()
pillow.setposition(130.00, 210.00)
place_position['pillow'] = pillow.position()

closet = turtle.Turtle()
closet.shape('circle')
closet.turtlesize(0.01)
closet.color('green')
closet.penup()
closet.setposition(310.00, 180.00)
place_position['closet'] = closet.position()

book_shelf = turtle.Turtle()
book_shelf.shape('circle')
book_shelf.turtlesize(0.01)
book_shelf.color('green')
book_shelf.penup()
book_shelf.setposition(145.00, 0.00)
place_position['book_shelf'] = book_shelf.position()

computer = turtle.Turtle()
computer.shape('circle')
computer.turtlesize(0.01)
computer.color('green')
computer.penup()
computer.setposition(260.00, 0.00)
place_position['computer'] = computer.position()

computer_chair = turtle.Turtle()
computer_chair.shape('circle')
computer_chair.turtlesize(0.01)
computer_chair.color('green')
computer_chair.penup()
computer_chair.setposition(260.00, 50.00)
place_position['computer_chair'] = computer_chair.position()

room_window = turtle.Turtle()
room_window.shape('circle')
room_window.turtlesize(0.01)
room_window.color('green')
room_window.penup()
room_window.setposition(340.00, 70.00)
```

```
    place_position['room_window'] = room_window.position()

    washer = turtle.Turtle()
    washer.shape('circle')
    washer.turtlesize(0.01)
    washer.color('green')
    washer.penup()
    washer.setposition(180.00, -230.00)
    place_position['washer'] = washer.position()

    refrigerator = turtle.Turtle()
    refrigerator.shape('circle')
    refrigerator.turtlesize(0.01)
    refrigerator.color('green')
    refrigerator.penup()
    refrigerator.setposition(240.00, -230.00)
    place_position['refrigerator'] = refrigerator.position()

    sink = turtle.Turtle()
    sink.shape('circle')
    sink.turtlesize(0.01)
    sink.color('green')
    sink.penup()
    sink.setposition(320.00, -230.00)
    place_position['sink'] = sink.position()

    oven = turtle.Turtle()
    oven.shape('circle')
    oven.turtlesize(0.01)
    oven.color('green')
    oven.penup()
    oven.setposition(320.00, -165.00)
    place_position['oven'] = oven.position()

    stove = turtle.Turtle()
    stove.shape('circle')
    stove.turtlesize(0.01)
    stove.color('green')
    stove.penup()
    stove.setposition(320.00, -100.00)
    place_position['stove'] = stove.position()

    house_window = turtle.Turtle()
    house_window.shape('circle')
    house_window.turtlesize(0.01)
    house_window.color('green')
    house_window.penup()
    house_window.setposition(-175.00, -250.00)
    place_position['house_window'] = house_window.position()

    return place_position
```

D.12 home_connectionposition.py

```
#===== Create turtles as shop position =====
import turtle
from turtle import *

def HomeConnectionPosition():

    connection_position = {}

    #===== Set connection position =====
    connection01 = turtle.Turtle()      #Connection01 is default point where Anna and
 Taro always stay.
    connection01.shape('square')
    connection01.turtlesize(0.01)
    connection01.color('pink')
    connection01.penup()
    connection01.setpos(90.00, -180.00)
    connection_position['connection01'] = connection01.pos()

    connection02 = turtle.Turtle()      # For table
    connection02.shape('square')
    connection02.turtlesize(0.01)
    connection02.color('pink')
    connection02.penup()
    connection02.setpos(-100.00, -150.00)
    connection_position['connection02'] = connection02.pos()

    connection03 = turtle.Turtle()      # For gold_fish_jar
    connection03.shape('square')
    connection03.turtlesize(0.01)
    connection03.color('pink')
    connection03.penup()
    connection03.setpos(-175.00, -75.00)
    connection_position['connection03'] = connection03.pos()

    connection04 = turtle.Turtle()      # For chair
    connection04.shape('square')
    connection04.turtlesize(0.01)
    connection04.color('pink')
    connection04.penup()
    connection04.setpos(-300.00, -150.00)
    connection_position['connection04'] = connection04.pos()

    connection05 = turtle.Turtle()      # For drinking_counter
    connection05.shape('square')
    connection05.turtlesize(0.01)
    connection05.color('pink')
    connection05.penup()
    connection05.setpos(-175.00, -35.00)
    connection_position['connection05'] = connection05.pos()

    connection06 = turtle.Turtle()      # For sofa
    connection06.shape('square')
    connection06.turtlesize(0.01)
    connection06.color('pink')
    connection06.penup()
    connection06.setpos(-175.00, 70.00)
    connection_position['connection06'] = connection06.pos()

    connection07 = turtle.Turtle()      # For TV
    connection07.shape('square')
    connection07.turtlesize(0.01)
```

```
connection07.color('pink')
connection07.penup()
connection07.setpos(-175.00, 160.00)
connection_position['connection07'] = connection07.pos()

connection08 = turtle.Turtle()      # For walk_way
connection08.shape('square')
connection08.turtlesize(0.01)
connection08.color('pink')
connection08.penup()
connection08.setpos(10.00, -50.00)
connection_position['connection08'] = connection08.pos()

connection09 = turtle.Turtle()      # For walk_way
connection09.shape('square')
connection09.turtlesize(0.01)
connection09.color('pink')
connection09.penup()
connection09.setpos(10.00, 60.00)
connection_position['connection09'] = connection09.pos()

connection10 = turtle.Turtle()      # For walk_way
connection10.shape('square')
connection10.turtlesize(0.01)
connection10.color('pink')
connection10.penup()
connection10.setpos(10.00, 150.00)
connection_position['connection10'] = connection10.pos()

connection11 = turtle.Turtle()      # For walk_way_in_bed_room
connection11.shape('square')
connection11.turtlesize(0.01)
connection11.color('pink')
connection11.penup()
connection11.setpos(200.00, 60.00)
connection_position['connection11'] = connection11.pos()

connection12 = turtle.Turtle()      # For walk_way_in_bed_room
connection12.shape('square')
connection12.turtlesize(0.01)
connection12.color('pink')
connection12.penup()
connection12.setpos(230.00, 130.00)
connection_position['connection12'] = connection12.pos()

connection13 = turtle.Turtle()      # For washer
connection13.shape('square')
connection13.turtlesize(0.01)
connection13.color('pink')
connection13.penup()
connection13.setpos(180.00, -210.00)
connection_position['connection13'] = connection13.pos()

connection14 = turtle.Turtle()      # For freezer
connection14.shape('square')
connection14.turtlesize(0.01)
connection14.color('pink')
connection14.penup()
connection14.setpos(240.00, -210.00)
connection_position['connection14'] = connection14.pos()
```

```
    connection14 = turtle.Turtle()      # For sink
    connection14.shape('square')
    connection14.turtlesize(0.01)
    connection14.color('pink')
    connection14.penup()
    connection14.setpos(305.00, -230.00)
    connection_position['connection14'] = connection14.pos()

    connection14 = turtle.Turtle()      # For oven
    connection14.shape('square')
    connection14.turtlesize(0.01)
    connection14.color('pink')
    connection14.penup()
    connection14.setpos(305.00, -165.00)
    connection_position['connection14'] = connection14.pos()

    connection14 = turtle.Turtle()      # For stove
    connection14.shape('square')
    connection14.turtlesize(0.01)
    connection14.color('pink')
    connection14.penup()
    connection14.setpos(305.00, -100.00)
    connection_position['connection14'] = connection14.pos()

    connection14 = turtle.Turtle()      # For house_window
    connection14.shape('square')
    connection14.turtlesize(0.01)
    connection14.color('pink')
    connection14.penup()
    connection14.setpos(-165.00, -235.00)
    connection_position['connection14'] = connection14.pos()
    return (connection_position)
```

D.13 closest_place.py

```
import turtle
from turtle import *
import math
from math import hypot
from setpositionofplaces import SetPositionofPlaces

def ClosePlace(current_point):
    place_position = SetPositionofPlaces()
    dis = {}     # dis is dictionary
    for w in place_position:
        if place_position[w] != current_point:
            dis[w] = math.hypot(place_position[w][0] - current_point[0],
place_position[w][1] - current_point[1])

    return (sorted(dis, key = dis.__getitem__)) # sort by values.
```

D.14 closepoint.py

```
#===== Create turtles as shop position =====
import turtle
from turtle import *
#import sys

def ClosePoint():

    close_point = {}

    #===== Set position of places =====
    post_office = turtle.Turtle()
    post_office.penup()
    post_office.color('yellow')
    post_office.turtlesize(0.01)
    post_office.setposition(120.00,200.00)      #Connection1
    close_point['post_office'] = post_office.pos()

    yard = turtle.Turtle()
    yard.penup()
    yard.color('yellow')
    yard.turtlesize(0.01)
    yard.setposition(50.00,150.00)      #Connection2
    close_point['yard'] = yard.pos()

    home = turtle.Turtle()
    home.penup()
    home.color('yellow')
    home.turtlesize(0.01)
    home.setposition(-20.00,90.00)      #Connection4
    close_point['home'] = home.pos()

    house = turtle.Turtle()
    house.penup()
    house.color('yellow')
    house.turtlesize(0.01)
    house.setposition(-20.00,90.00)      #Connection4
    close_point['house'] = house.pos()

    apartment = turtle.Turtle()
    apartment.penup()
    apartment.color('yellow')
    apartment.turtlesize(0.01)
    apartment.setposition(-120.00,0.00)      #Connection5
    close_point['apartment'] = apartment.pos()

    restaurant = turtle.Turtle()
    restaurant.penup()
    restaurant.color('yellow')
    restaurant.turtlesize(0.01)
    restaurant.setposition(-270.00,-140.00)      #Connection6
    close_point['restaurant'] = restaurant.pos()

    bridge = turtle.Turtle()
    bridge.penup()
    bridge.color('yellow')
    bridge.turtlesize(0.01)
    bridge.setposition(85.00,130.00)
    close_point['bridge'] = bridge.pos()

    bridge_1 = turtle.Turtle()
    bridge_1.penup()
    bridge_1.color('yellow')
    bridge_1.turtlesize(0.01)
    bridge_1.setposition(370.00,90.00)
    close_point['bridge_1'] = bridge_1.pos()

    stair = turtle.Turtle()
    stair.penup()
    stair.color('yellow')
    stair.turtlesize(0.01)
    stair.setposition(320.00,-90.00)
    close_point['stair'] = stair.pos()

    fountain = turtle.Turtle()
    fountain.penup()
    fountain.color('yellow')
    fountain.turtlesize(0.01)
    fountain.setposition(420.00,90.00)      #Connection8
    close_point['fountain'] = fountain.pos()

    table = turtle.Turtle()
    table.penup()
    table.color('yellow')
    table.turtlesize(0.01)
    table.setposition(420.00,20.00)      #Connection9
    close_point['table'] = table.pos()

    river_north = turtle.Turtle()
    river_north.penup()
    river_north.color('yellow')
    river_north.turtlesize(0.01)
    river_north.setposition(300.00,200.00)      #Connection10
    close_point['river_north'] = river_north.pos()

    river_south = turtle.Turtle()
    river_south.penup()
    river_south.color('yellow')
    river_south.turtlesize(0.01)
    river_south.setposition(100.00,-250.00)      #Connection11
    close_point['river_south'] = river_south.pos()

    road = turtle.Turtle()
    road.penup()
    road.color('yellow')
    road.turtlesize(0.01)
    road.setposition(220.00,0.00)      #Connection12
    close_point['road'] = road.pos()

    road_north = turtle.Turtle()
    road_north.penup()
    road_north.color('yellow')
    road_north.turtlesize(0.01)
    road_north.setposition(105.00,180.00)      #Connection13
    close_point['road_north'] = road_north.pos()

    road_south = turtle.Turtle()
    road_south.penup()
    road_south.color('yellow')
    road_south.turtlesize(0.01)
    road_south.setposition(-250.00,-200.00)      #Connection14
    close_point['road_south'] = road_south.pos()
    flower_bed = turtle.Turtle()
    flower_bed.penup()
    flower_bed.color('yellow')
    flower_bed.turtlesize(0.01)
    flower_bed.setposition(450.00, -160.00)
    close_point['flower_bed'] = flower_bed.pos()
    return close_point
```

D.15 sex_detection.py

```
def SexDetection(gender,final_locus,word):

    #===== Locus formula =====
    taro = ['taro_', ['sand'], [['x', 'x', 'male', 'male', 'sex', 'pn', '+']], ['x', 
'none', 'none']]
    tom = ['tom_', ['sand'], [['x', 'x', 'male', 'male', 'sex', 'pn', '+']], ['x', '
none', 'none']]
    mary = ['mary_', ['sand'], [['x', 'x', 'female', 'female', 'sex', 'pn', '+']], ['
x', 'none', 'none']]
    anna = ['anna_', ['sand'], [['x', 'x', 'female', 'female', 'sex', 'pn', '+']], ['
x', 'none', 'none']]
    All_locus = [taro,tom,mary,anna]
    #===== End of locus =====

    temp = ''
    for w in All_locus:
        if gender in w[2][0]:
            for x in final_locus:
                for xx in x:
                    if w[0][:-1] in xx:
                        temp = w[0][:-1]

    if (temp == ""):
        temp = word

    for w in final_locus:
        for ww in w:
            if word in ww:
                for www in ww:
                    if www == word:
                        i = ww.index(www)
                        ww.pop(i)
                        ww.insert(i,temp)
    return final_locus
```

D.16 question_answer.py

```python
import random
from random import randint
from setpositionofplaces import SetPositionofPlaces
from connectionposition import ConnectionPosition

def QuestionAnswering(final_locus, text, last_locus):

    if last_locus != "":
        place_position = SetPositionofPlaces()
        print place_position
        for w in final_locus:
            for ww in w:
                if (ww != 'sand') and (ww != 'cand'):
                    for www in ww:
                        if www in place_position:
                            print www, place_position[www]
                        else:
                            print 'pass'
```

D.17 pattern_matching.py

```
def PatternMatching(final_locus,text):

    temp = []; temp1 = ''
    if (len(final_locus) > 1) and ('?' in text):
        for w in final_locus[1:]:
            for ww in w:
                if (ww != ['sand']) and (ww != ['cand']):
                    for x in final_locus[0]:
                        if (x != ['sand']) and (x != ['cand']):
                            if ww[0] == x[0]:
                                if x[2] != 'p':
                                    if x[2] not in temp:
                                        temp.append(x[2])
                                        temp1 = x[0]
                                elif x[3] != 'q':
                                    if x[3] not in temp:
                                        temp.append(x[3])
                                        temp1 = x[0]

                            elif ww[1] == x[1]:
                                if x[2] != 'p':
                                    if x[2] not in temp:
                                        temp.append(x[2])
                                        temp1 = x[1]
                                elif x[3] != 'q':
                                    if x[3] not in temp:
                                        temp.append(x[3])
                                        temp1 = x[1]
    if temp1 != "":
        temp.append(temp1+"_")
    return temp
```

REFERENCES

[1.1] G. D. Logan and D. D. Sadler, *A Computational Analysis of the Apprehension of Spatial Relations*. Cambridge, MA: MIT Press, 1996.

[1.2] T. Winograd, “Shifting Viewpoints: Artificial Intelligence and Human-Computer Interaction”, *Journal Artificial Intelligence*, pp. 1256 - 1258, 2006.

[1.3] J. Weizenbaum, “ELIZA – A Computer Program for the Study of Natural Language Communication between Man and Machine”, *Communication of the ACM*, 1966.

[1.4] A.M. Turing, “Computing Machinery and Intelligence”, *Mind 49*, pp. 433 - 460, 1950.

[1.5] H. J. Levesque, “The Winograd Schema Challenge”, *Proceeding of the CommonSense-11 Symposium*, 2011.

[1.6] H. J. Levesque, E. Davis, and L. Morgenstern, “The Winograd Schema Challenge”, *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 552 - 561, 2012.

[1.7] L. Morgenstern and C. L. Ortiz, Jr., “The Winograd Schema Challenge: Evaluating Progress in Commonsense Reasoning”, *Proceedings of the Twenty-Seventh Conference on Innovative Applications of Artificial Intelligence*, pp. 4024 - 4025, 2015.

[1.8] D. Ferrucci et al., *Building Watson: An Overview of the DeepQA Project*, AAAI AI Magazine, Fall 2010.

[1.9] C. Thompson, *What is I.B.M.'s Watson?*, The New York Times Magazine, June 16, 2010.

[1.10] B. M. Lake et al., “Building Machines That Learn and Think Like People”, *Center for Brains, Minds & Machines (CBMM) Memo 46*, 2016.

[1.11] M. Yokota, “An Approach to Natural Language Understanding Based on Mental Image Model”, *NLUCS 2005 Proceedings of the 2nd International Workshop on Natural Language Understanding and Cognitive Science*, pp. 22 - 31, 2005.

[1.12] M. Yokota, “Aware Computing in Spatial Language Understanding Guided by Cognitively Inspired Knowledge Representation”, *Journal Applied Computational Intelligence and Soft Computing*, 10 pages, 2012.

[2.1] M. Yokota and R. Khummongkol, “Representation and Computation of Human Intuitive Spatiotemporal Concepts as Mental Imagery”, *IEEE 6th International Conference on Awareness Science and Technology (iCAST)*, 6 pages, 2014.

[2.2] J. F. Allen, “Towards a general theory of action and time”, *Journal Artificial Intelligence*, pp. 123 - 154, 1984.

[2.3] M. Yokota, “Systematic Formulation and Computation of Subjective Spatiotemporal Knowledge Based on Mental Image Directed Semantic Theory: Toward a Formal System for Natural Intelligence”, *The 6th International Workshop on Natural Language Processing and Cognitive Science*, pp. 133-143, 2009.

[3.1] M. Egenhofer, “Point-Set Topological Spatial Relations”, *Geographical Information Systems*, pp.161 - 174, 1991.

[4.1] F. Heylighen, “Formulating the Problem of Problem-Formulation”, *Cybernetics and Systems '88*, pp. 949 - 957, 1988.

[4.2] S.J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (3rd ed.)*, Upper Saddle River: Prentice Hall, 2010.

[5.1] R. Khummongkol and M. Yokota, “Computer Simulation of Human-Robot Interaction in Natural Language”, *the 21st International Symposium on Artificial Life and Robotics (ISAROB’16)*, 5 pages, 2016.

[5.2] R. Khummongkol and M. Yokota, “Computer Simulation of Mental Image Processing in Natural Language Understanding by Human”, *IEEE 7th International Conference on Awareness Science and Technology*, 6 pages, 2015.

[5.3] R. Khummongkol and M. Yokota, “Spatiotemporal Language Understanding Based on Mental Image Directed Semantic Theory”, *the 22nd International Symposium on Artificial Life and Robotics (ISAROB’17)*, 7 pages, 2017.

[6.1] H. Levesque and G. Lakemeyer, *Cognitive Robotics*, Handbook of Knowledge Representation, Elsevier, 2008.

[6.2] R. Khummongkol and M. Yokota, “Simulation of Human Awareness Control in Spatiotemporal Language Understanding as Mental Image Processing”, *2014 IEEE International Symposium on Independent Computing (ISIC)*, 6 pages, 2014.

[6.3] R. Khummongkol and M. Yokota, “Computer Simulation of Human-Robot Interaction through Natural Language”, *Artificial Life and Robotics*, pp.1-10, 2016.

[6.4] R. Khummongkol and M. Yokota, “Systematic Representation and Computation of Human Intuitive Spatiotemporal Knowledge as Mental Imagery”, *the International Conference on Machine Learning and Cybernetics (ICMLC)*, 6 pages, 2016.

[6.5] R. Khummongkol and M. Yokota, “Logical Formalization of Human Intuitive Mental Imagery for Spatiotemporal Language Understanding”, *International Journal of Control Systems and Robotics*, pp. 150-157, pp.150-157, 2016.

[6.6] R. Navigli, “Word Sense Disambiguation: A Survey”, *ACM Computing Surveys*, pp.1–69, 2009.

[6.7] C.D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.

[6.8] A. Nguyen, J. Yosinski, and J. Clune, “Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images”, *Computer Vision and Pattern Recognition (CVPR’15)*, 2015.

[6.9] B. M. Lake et al., “Building Machines That Learn and Think Like People”, *Center for Brains, Minds & Machines (CBMM Memo 46)*, 2016.

[A.1] M. Yokota, *Subjective Knowledge Representation for Intuitive Human – Robot Interaction Based on Mental Image Directed Semantic Theory*, Horizons in Computer Science Research, Vol. 7 (Edited by T. S. Clary), Nova science Publishing Co., ISBN: 978-1-61942-777-8, 2012.