ISSN 0918-2802

Technical Report

Processing of 3-D Spatial Relations for Virtual Agents Acting on Natural Language Instructions

Yusuke Shinyama

TR00-0010 July

Department of Computer Science
Tokyo Institute of Technology
Ôokayama 2-12-1 Meguro Tokyo 152-8552, Japan
http://www.cs.titech.ac.jp/

©The author(s) of this report reserves all the rights.

To appear in Virtual Agent 99

Processing of 3-D Spatial Relations for Virtual Agents Acting on Natural Language Instructions

Yusuke Shinyama, Takenobu Tokunaga, and Hozumi Tanaka

Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, 2-12-1, Ôokayama, Meguro-ku, Tokyo, 152-8552, Japan
{euske, take, tanaka}@cs.titech.ac.jp

Abstract. We are developing a system in which a user can control virtual agents through natural language dialogue. We particularly focus on the semantic analysis of spatial expressions in natural language instructions. Spatial expressions are relative to the speaker's viewpoint and the indicated positions can change dynamically. In this paper we propose a semantic representation using lambda structures. Some features of lambda structure enable us to keep the speaker's viewpoints in such semantic representations undetermined, to keep track of changing positions, and to encapsulate representations like English prepositions. With this representation, we can handle this kind of spatial expressions efficiently. We also state the method of translating natural language instructions to this representation.

1 Introduction

Graphical user interface is recently a popular way of controlling computers. Although direct manipulation such as mouse control and motion capture corresponds to intuitive human analogy, such interfaces are not always appropriate. For example, consider the situation where a user works with a CG animation tool which has a graphical user interface only. If the user want a CG character to run along a road, the user will have to trace a path along which the character is to move using a mouse. Moreover, if the user wants the character to run along the road with his hand waving, the user will need some way of modifing its bodily movement. Presumably the user will have to use many buttons or control bars to adjust movement in this way, because many GUI systems inevitably facilitate such control through switches and buttons, to deal with the vast amount of information associated with the task. It can be hard to learn to use such interfaces, and the GUI is probably misleading. Many such disadvantages of GUIs are enumerated by Gentner et al. [4]. On the other hand, if the tool can understand natural language instructions, the user can accomplish his goal simply by saying "Run along the road". For the latter more complicated goal, the user could simply say "Run along the road with your hand waving".

Interacting with the computer through natural language has several advantages. First, it is quite natural and easy for a user to give instructions. Second,

language instructions can express complicated commands such as repetition and simultaneous action. Third, we can adequately and succinctly convey requests using contextual information. For example, we can just say "Further right" after commanding the character to "Move right".

However, there are many research issues to realize such a system. In order to satisfy the user's requests, the system is required to decode the users' intentions from natural language. Since Winograd's SHRDLU [9], very few researchers have studied aspects of language connected directly to actions in a physical, dynamic world. And natural language understanding systems can still fail to capture users' intentions at various levels.

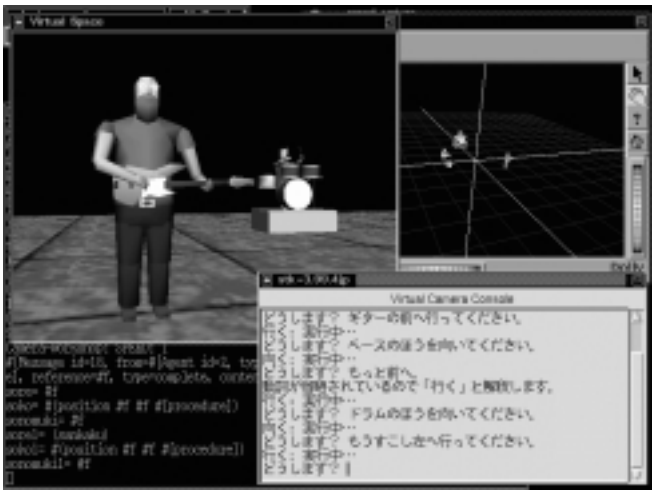

Fig. 1. The screenshot of the system

We have developed a prototype version of the system in which virtual agents reside in the virtual space (see Fig. 1). A user can give the agents instructions in Japanese and see gradual changes in the virtual space through a graphic window. This system provides a testbed for investigating the relation between natural language as an interface to computers, and actual actions performed by the computer. For the present, the user can control only a camera agent which project the virtual space into graphic windows. In this sort of system, instructions contain many spatial expressions with which a user commands the camera agent to move or turn. In this paper we define a spatial expression to be a natural language phrase which specifies a position, area or direction in the virtual space. In English, many prepositional phrases modifying places or regions are regarded as spatial expressions. We require that a variety of spatial expressions are handled. In this paper we propose a semantic representation for spatial expression. The representation enables us to handle spatial expressions efficiently.

2 Brief Introduction to Japanese Grammar

Before describing the system, let us give a brief outline of Japanese grammar. Ordinary Japanese sentences are a sequence of so-called *bunsetsu* phrases, that is a chunk of content words (nouns, pronouns, adjectives, etc.) followed by a postposition which marks the case of the preceding elements. Usually the verb resides in the last bunsetu phrase of the sentence. For example, the sentence "He went forward." can be written in Japanese as follows:

Example 1. "Kare (pronoun-he) wa (postposition-agent) / mae (noun-front) ni (postposition-to) / itta (verb-went)."

Here each "/" denotes a delimiter between bunsetsu phrases. A bunsetu phrase containing "no" modifies the proceeding phrase. For example, "Kare (pronoun-he) no (postposition-possession) / tsukue (noun-desk)" means "his desk". However, the meaning of the postposition "no" varies depending on what it modifies. In particular many spatial expressions use the postposition "no". For example, let us consider the phrase "tsukue (noun-desk) no (postposition-of) / mae (noun-front)". In this phrase, the word "mae" is modified by the previous noun "tsukue no". In this case, "no" indicates the origin of the directional expression "mae". So this phrase means "the front of the desk". In a later section, we will see a fragment of Japanese grammar which can express some spatial relations with the postposition "no".

3 System Overview

In the current prototype system, a user can give natural language instructions to the agents in the virtual space through voice input. When the system receives an input, it is analyzed and the action is performed. While only the camera agent can be controlled by the user at present, there are other agents in the virtual space. The other agents stay at a pre-determined location. The user can command the camera agent to move or turn in a particular way. The camera agent moves or pans according to the commands and the user can see gradual view change as in the graphic view. The following are examples the current system can deal with:

- "Isu (noun-chair) no (postposition-of) /
  mae (noun-front) ni (postposition-to) /
  ike (verb-go)." (Move to the front of the chair)
- "Tsukue (noun-desk) no (postposition-of) /
  hou (noun-direction) wo (postposition-to) /
  muke (verb-turn)." (Turn toward the desk)
- "Mousukoshi (adverb-somewhat) /
  sore (pronoun-it) ni (postposition-to) /
  chikazuke (verb-approach)." (Approach it a little)

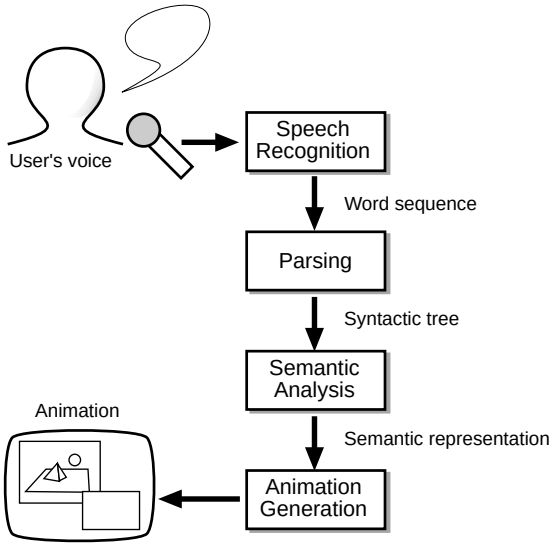

Fig. 2. The components of the current system

The system consists of following modules (see Fig. 2):

1. Speech recognition module
2. Syntactic analysis module
3. Semantic analysis module
4. Animation generation module

The user's natural language instructions are processed through these modules. At first the speech recognition module converts the voice input into a word sequence. The syntactic analysis module parses it to generate a syntactic tree. For the sake of modularity, in this phase we use a general-purpose parser [1] which analyzes sentences using a given grammar. The tree is traversed by the semantic analysis module to generate a semantic representation, which is language independent structure containing more detailed information. For animation generation, the system has to specify precisely several parameters such as the coordinates of objects. Information not explicitly specified in the instruction, such as motion speed, distance of movement, are filled in by default values during this process. Then, the animation generation module is passed the semantic representation and animates the virtual space accordingly.

In this paper we particularly focus on the semantic analysis module. In order to control the virtual agents, the system is required to interpret spatial expressions such as "rightward" and "near" and determine the specific coordinate or directional vector which is implied by the instructions. But there are several problems in this process.

4 Problems with Spatial Expressions

Relativeness to the speaker: User-designated spatial expressions are usually relative to the user's viewpoint. For example, consider the case that two persons standing at different positions are saying "Look to the right of the chair" (see Fig. 3). The actual meaning of the sentence varies depending on each speaker's position. In this case, the desired result would be different for persons standing at position A and B. In our system, the user gives instructions while viewing the virtual space through the camera agent, so his position will change as he commands the camera agent to change the view. Therefore we should take account of the position from which the the instruction was issued in computing the goal position.

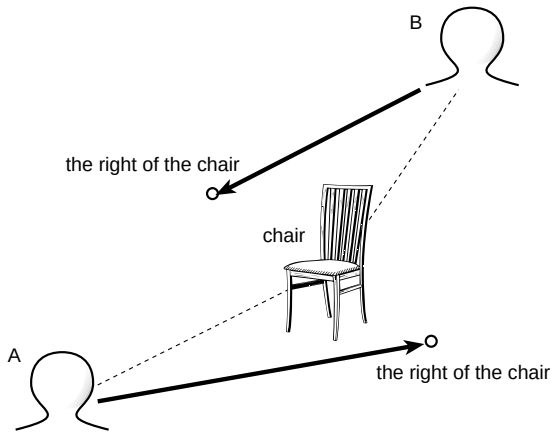

Fig. 3. The relativeness of spatial expressions

Nested expression: One solution to the problem of relativeness is to introduce a viewpoint argument into a function which compute a goal position. For example, consider the function $f_{RightOf}(p_s, p_t, d_1)$ which takes the speaker's coordinates p_s, the target coordinates p_t and the distance to the right of the target coordinates d_1. This function returns the coordinates of a certain point to the right of position p_t at distance d_1, as viewed from position p_s. With this approach, we obtain the coordinates expressed by "the right of A" by $f_{RightOf}(p_s, P_A, d_1)$ where P_A is the coordinates of the target A. Adopting this approach, we need to use our function to obtain the coordinates while traversing the syntactic tree. However, we sometime encapsulate spatial phrases using prepositions, to make new constructs such as "near the right of A". Therefore we may have to synthesize such functions into something like $f_{Near}(p_s, f_{RightOf}(p_s, p_t, d_1), d_2)$. This approach has the disadvantage that once the value of the function is computed,

the nesting information is completely lost. If we want to reuse part of this information at a later stage to adjust the distance d_1 or d_2, there is no way to do it. Relations between functions should thus be preserved for later reference.

Kalita et al. [6] have proposed routines to solve spatial relations. Although their approach can be applied to many prepositions, they haven't proposed how to translate nested expressions into the proposed form. In addition, since their method is comparatively expensive computationally, it is not useful for our system which requires real-time response.

Moving targets: When two or more objects in the virtual space are moving simultaneously, the spatial relations between them also change. In this case, it is not a good idea to keep track of objects' coordinates at a particular time.

Vagueness: When we say "go to the front of A", the distance between A and the goal position is vague [5]. Without this information the system cannot determine the actual goal coordinates. A default value is initially fed into the system to account for such vague quantities. However, there may be cases where the amount is affected by the relation between the speaker and A.

Spatial expressions are frequently used in controlling agents with natural language. In the following section, we propose a semantic representation which solves the above problems and realizes versatile handling of spatial expressions.

5 Semantic Representation Using Lambda Structure

We use abstract lambda structures for the efficient handling of spatial expressions. Our basic idea in handling spatial expressions is that all semantic representations of spatial expressions should be represented with lambda structures. A lambda structure can be regarded as an entity representing a mathematical function which acts as a black box to receive input values and output a value based thereupon. Unlike ordinary functions, a lambda structure can be treated as an object. We therefore can substitute, store and retrieve lambda structures in the same manner as other objects such as numbers. In addition, a lambda structure can take another lambda structure as input. This enables us to synthesize lambda structures to compose a new structure which also works as a function.

"Application" is an important operation on lambda structures. Only when all arguments are applied to a lambda structure, does it act as a function, and it acts as an object otherwise. An application can alternatively be considered as sending a message to an object, producing a value. For example,

$$\lambda x.add(x, add(x, 2)) \tag{1}$$

usually acts as an object. Here add is an arithmetic function to calculate the sum of two arguments. $\lambda x.$ denotes lambda abstraction, which can be regarded as the interface for the input. When a value is fed into this, all appearances of the

variable x are replaced with the input value. That is called "beta-translation". So, if the number 3 is fed into the structure, it returns the value 8. If we use lambda structures instead of ordinary functions in handling spatial expressions, we can keep the speaker's viewpoint undetermined.

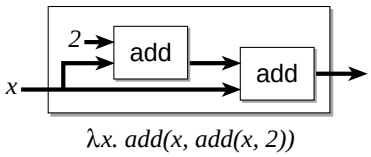

Fig. 4. A lambda structure

In addition, a lambda structure is internally encoded with a description of how to use its input value. In this case, input variable x is used twice in the structure. So, a lambda structure can be regarded figuratively as a pipeline which connects between the input values with other functions (see Fig. 4). That means that a lambda structure preserves relations between functions used in the structure. This feature seems to be useful in handling nested expressions. But to exploit this advantage, we need to introduce another important concept of lambda structures: "higher order function". Since a lambda structure is taken merely as an object, a function can return a function as its value. Such a function is called a higher order function [2]. Consider the following example:

$$\lambda x.\lambda y.add(x, y) \tag{2}$$

When one value is applied to the structure, say 5, we produce

$$\lambda y.add(5, y) \tag{3}$$

which is also a lambda structure. And remember that, because a lambda structure is an object, it is allowed to take other lambda structures as arguments. With these two features we can make functions that "take a function as an argument and return a function". For example,

$$convertplusone \equiv \lambda x.\lambda y.add(x(y), 1) \tag{4}$$

works as a "function converter", which converts a function into another function that adds 1 to its output. So we have:

$$\begin{aligned} convertplusone(convertplusone(\lambda z.z))(2) \Rightarrow \\ \lambda y.add(add((\lambda z.z)(y), 1), 1)(2) \Rightarrow \\ add(add(2, 1), 1) \Rightarrow 4 \end{aligned} \tag{5}$$

Based on this characteristic, we can make lambda structures which can be applied infinitely and always take one argument. That makes the lambda structure based representation of nested expressions quite tractable. Furthermore,

the computation of lambda structure is not executed until the input value is actually supplied. This is called "lazy evaluation" which enables us to handle dynamically moving positions. Therefore, using lambda structures has several attractive benefits in the implementation of an efficient semantic representation. On the basis of the above characteristics, we can design our system to translate spatial expressions into lambda structures with the following principles.

At first we prescribe that every spatial expressions should be translated into a lambda structure. Every phrase containing a spatial expression should be handled as a function, not as a value, because this rule enables the system to keep the speaker's viewpoint undetermined and to keep track of dynamically changing positions. When a spatial expression is translated, the system stores it and uses it for animation generation, in which the speaker's viewpoint changes every frame. Let us call this type of lambda structure a "spatial lambda structure" (denoted below by S_i, etc.). All spatial lambda structures must have one input comprising the speaker's viewpoint coordinates and return one output comprising the coordinates computed by the structure. This regulates the interface between lambda structures, such that all spatial lambda structures can be handled in the same manner. In this way, we can represent spatial expressions.

We use lambda structures for another purpose, that is to convert spatial lambda structures into other spatial lambda structures. Let us call this type of lambda structure a "converter lambda structure" (denoted below by C_i etc.). This kind of structure converts a spatial lambda structure denoting a certain type of position in the virtual space, into another type of position. So, we can restrict or modify spatial lambda structures using converter lambda structures. Converter lambda structures are higher order functions, regarded as function templates. A converter lambda structure takes one or more arguments. The first argument must be the spatial lambda structure to be converted. The other arguments passed to the converter let the structure know how to restrict or modify the spatial lambda structure. Therefore, a converter lambda structure acts like an English preposition representing a relative relation between positions. This enables us to "encapsulate" a lambda structure within another lambda structure, as for nested expressions.

When constructing structures, we at first use spatial lambda structures which initially exist in the system. Such spatial lambda structures points to a certain constant position or a certain moving position in the virtual space and are considered as primitives for composing semantic representations. Next we apply converter lambda structures as many as we want to restrict or modify the initial position. Suppose that we have a primitive spatial lambda structure S_0 and some converter lambda structures C_1, C_2, C_3, ...,C_n, then the following rule is satisfied:

$$C_1 C_2 C_3 ... C_n S_0 \Rightarrow S_{Final} \tag{6}$$

In this way, we can derive a nested semantic representation for a nested spatial expression.

In the remainder of this section, we illustrate how the problems mentioned in the previous section are solved through the use of these principles.

First, we can represent a constant position A as a spatial lambda structure S_A, at which a fixed object is located in the virtual space, with:

$$S_A \equiv \lambda p_s.P_A \tag{7}$$

Here P_A denotes the actual coordinates of the object A, which is fixed in the virtual space. Applying the viewpoint of the speaker to p_{s1}, the structure can be reduced by beta-translation to:

$$(\lambda p_s.P_A)p_{s1} \Rightarrow P_A \tag{8}$$

This means that when p_{s1} is fed into the lambda structure it discards this value and returns the value P_A. Although this simply acts as a constant, we encapsulate even constants with lambda abstraction in order to maintain the above regulation of the interface. We can use the converter lambda structure $C_{RightOf}$ to represent the meaning of "right of s_1":

$$C_{RightOf} \equiv \lambda s_1.\lambda p_s.f_{RightOf}(p_s, (s_1)p_s) \tag{9}$$

Note that s_1 is also a spatial lambda structure which satisfies the above principles. Here $f_{RightOf}(p_s, p_t)$ denotes an ordinary function, which takes the coordinates of the speaker as p_s and the coordinates of the target as p_t, and returns a location located to the right of position p_t, as seen from the position p_s. From this structure, we obtain another spatial lambda structure $S_{RightOfA}$ which represents "the right of A" as (see Fig. 5):

$$S_{RightOfA} \equiv (C_{RightOf})(S_A) \Rightarrow \lambda p_s.f_{RightOf}(p_s, (\lambda p_{s'}.P_A)p_s) \tag{10}$$

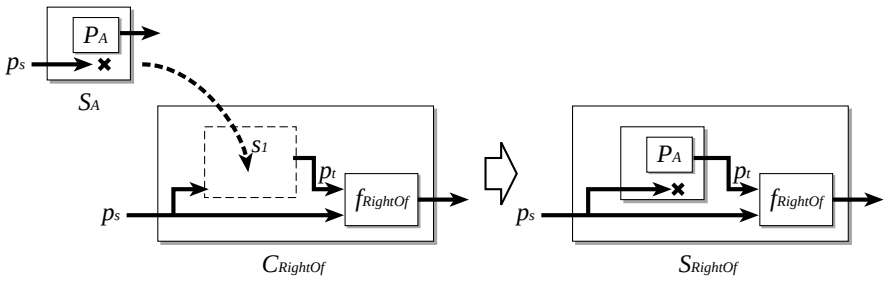

Fig. 5. Composing a lambda structure

Note that alpha-translation is used here to avoid conflict of variable names. The structure can be reduced when the speaker's viewpoint is applied, as in:

$$(S_{RightOfA})p_{s1} \Rightarrow f_{RightOf}(p_{s1}, P_A) \tag{11}$$

Now we can obtain the exact coordinates by applying the coordinates of the camera agent (= the speaker's viewpoint) to the structure. Thus, the converter

lambda structure forms a function template of a function which determines where the right of $s1$ is.

Moreover we can synthesize structures recursively. For example, we can similarly define another converter lambda structure C_{Near} representing the meaning of the word "near" as:

$$C_{Near} \equiv \lambda s_1.\lambda p_s.f_{Near}(p_s, (s_1)p_s) \tag{12}$$

And synthesizing C_{Near} with $C_{RightOf}$, we can construct another converter structure $C_{NearTheRightOf}$ which represents the meaning of "near the right of" as :

$$C_{NearTheRightOf} \equiv (C_{Near})(C_{RightOf}) \tag{13}$$

Following the above principles, many phrases containing spatial expressions can be represented as spatial lambda structures. The system can use the structure in computing the goal coordinates for each animation frame. Once a spatial lambda structure is constructed, it is stored in the system and supplied with the speaker's viewpoint (the position of the camera in this case) every frame. In addition, applying the viewpoint is not computationally expensive. Actually the user can alter not only the position but also the direction of the camera. Though we omitted details of direction to simplify the explanation, the same formalism can be used to compute direction. The system carries out moving and panning as primitive operations, and we can command the system to execute them simultaneously. Thus, using lambda structures, we can make a semantic representation which solves the problems mentioned in the previous section efficiently.

6 Translation of Natural Language Instructions

In our approach, there is an isomorphism between syntax and formal semantics. We designed the semantic representation to be constructed recursively in correspondence to the syntactic tree. In linguistics, such isomorphisms are commonly employed in Montague grammar [8, 3], and are used in a similar fashion to our proposed formulation. Applying this idea to our approach, we can translate spatial expressions into our semantic representation. Although we mention here only the case of Japanese, the proposed method can be also applied readily to English. For example, consider the sentence in Example 2, meaning "near the right of the desk" :

Example 2. "Tsukue (noun-desk) no (postposition-of) / migi (noun-right) no (postposition-of) / chikaku (noun-neighbourhood)"

This expression would occur as part of an overall instruction. The expression is syntactically analyzed as shown in Fig. 6. The semantic analysis module traverses the tree to construct the semantic representation. For each node, the module recursively composes a representation of subtrees of that node, including spatial expressions. The final structure constitutes a lambda structure which

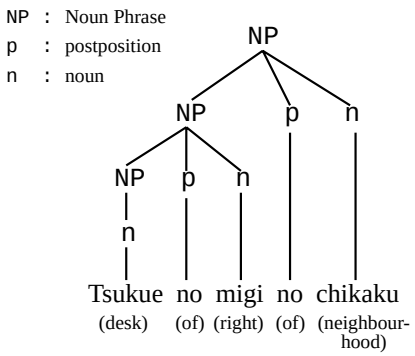

Fig. 6. The syntactic tree for "Tsukue no / migi no / chikaku"

takes the speaker's viewpoint as an argument and returns the coordinates indicated by that expression. Here we can define the function which translates a subtree containing spatial expression into a semantic representation. Let `NP:(NP p n)` be a tree node `NP` consisting of three subtrees: `NP`, `p` and `n` in this order. Function f_{sem} is defined as:

$$f_{sem}(\texttt{NP:(NP p n)}) \equiv (f_{sem}(\texttt{n}))(f_{sem}(\texttt{NP})) \tag{14}$$

$$f_{sem}(\texttt{NP:n}) \equiv f_{sem}(\texttt{n}) \tag{15}$$

$$f_{sem}(\texttt{n:tsukue}) \equiv \lambda p_s.P_{Desk} \tag{16}$$

$$f_{sem}(\texttt{n:migi}) \equiv \lambda s_1.\lambda p_s.f_{RightOf}(p_s,(s_1)p_s) \tag{17}$$

$$f_{sem}(\texttt{n:chikaku}) \equiv \lambda s_1.\lambda p_s.f_{Near}(p_s,(s_1)p_s) \tag{18}$$

Applying function f_{sem} to the root node of the tree in Fig. 6, we obtain a spatial lambda structure representing "near the right of desk" as:

$$\begin{aligned} f_{sem}(\texttt{NP:(NP:(NP:(n:tsukue) pp:no n:migi) pp:no n:chikaku)}) &= \\ (f_{sem}(\texttt{n:chikaku}))(f_{sem}(\texttt{NP:(NP:(n:tsukue) pp:no n:migi)})) &= \\ (f_{sem}(\texttt{n:chikaku}))((f_{sem}(\texttt{n:migi}))(f_{sem}(\texttt{NP:(n:tsukue)}))) &= \\ (f_{sem}(\texttt{n:chikaku}))((f_{sem}(\texttt{n:migi}))(f_{sem}(\texttt{n:tsukue}))) &= \\ (\lambda s_1.\lambda p_s.f_{Near}(p_s,(s_1)p_s))((\lambda s_1.\lambda p_s.f_{RightOf}(p_s,(s_1)p_s))(\lambda p_s.P_{Desk})) &\Rightarrow \\ \lambda p_s.f_{Near}(p_s,(\lambda p_{s'}.f_{RightOf}(p_{s'},(\lambda p_{s''}.P_{Desk})p_{s'}))p_s) & \end{aligned} \tag{19}$$

To implement the proposed method, a programming language is required which has facility to handle lambda structures. Some functional languages such as Scheme and ML support the storage and application of lambda structures as a primitive operation. We chose Scheme to implement the system. The semantic analysis module written in Scheme uses such functions as f_{sem} to build the semantic representation. In the animation generation module, the lambda structures are supplied with the coordinates of the camera agent for each picture frame.

7 Conclusion

In this paper, we proposed the representation of spatial expressions in terms of lambda structures. Thanks to particular characteristics of lambda structures, semantic representations of spatial expressions can be easily constructed and efficiently computed. With this formalism, we can effectively handle spatial relations in natural language instructions.

In future work, we hope to achieve the following goals:

- Apply this formulation to control other objects.
- Use contextual information to allow users to provide more succinct commands.
- Disambiguation of modification. Currently our method is based on the assumption that a spatial expression modifies only one target. However there may be cases where the system need to determine which target is modified by spatial expressions from syntactic or semantic standpoints.
- Detailed tuning. Make the camera agent aware of the size of the subjects, Realize simultaneous movement of multiple agents by more complicated commands, and so on.

References

1. Aizawa, M., Tokunaga, T., Tanaka, H., "Integration of Morphological and Syntactic Analysis on LR Parsing Algorithm", Technical Report of the Institute of Electronics, Information and Communication Engineers, NLC93-2, pp. 9-16, May, 1993.
2. Curry, H. B., Feys, R., "Combinatory Logic", North-Holland, 1958.
3. Dowty, D. R., Wall, R. E, Peters, S., "Introduction to Montague Semantics", D. Reidel Pub. Company, 1981.
4. Gentner, D., Nielson, J., "The Anti-Mac Interface", Communication of the ACM, Vol. 39, No. 8, pp. 71-82, August 1996.
5. Herskovits, A., "Language and Spatial Cognition: An Interdisciplinary Study of the Prepositions in English", Cambridge University Press, 1986.
6. Kalita, J. K., Badler, N. I., "Interpreting Prepositions Physically", AAAI-91 Proceedings Ninth National Conference on Artificial Intelligence, Volume One, pp. 105-110, July 14-19, 1991.
7. Katagiri, Y., "The World of Discourse", Natural Language Understanding, Tanaka H., Tsujii J. (ed.), pp. 159-190, ISBN4-274-07398-X, OHM Press, 1988.
8. Montague, R., "Proper Treatment of Quantification in Ordinary English", Formal Philosophy, pp. 247-270, Yale University, 1974.
9. Winograd, T., "Understanding Natural Language", Academic Press, 1972.
10. Yamada, A., Amitani, K., Hoshino, T., Nishida, T., Doshita, S., "The Analysis of the Spatial Descriptions in Natural Language and the Reconstruction of the Scene", Monthly Journal of the Information Processing Society of Japan, Vol. 31, No. 5, pp. 660-672, May, 1990.