REPRESENTING DEFINITIONS AND ITS ASSOCIATED KNOWLEDGE IN A LEARNING PROGRAM[1]

Kam-Hoi Cheng

Computer Science Department
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

UH-CS-07-01
January 8, 2007

Keywords: Knowledge Representation, Definition, Condition, Method, Category

Abstract

Category allows the abstraction of similar ideas into a single term for easy communication with another person. The similarities may be succinctly captured by a definition. In this paper, we are interested in the implementation of a program to acquire definitions, and use them to answer questions concerning these definitions. Several major components of a definition have been identified and implemented: given, condition, and method. The different subclasses and other associated class hierarchies needed are presented. The methods learned may be used to answer two different kinds of questions, namely the "is" and the "what" questions. The "is" question is to solve the classification problem, whereas the "what" question is to find an answer that satisfies some definition. When answering a question posted by a human, the answer is obtained by executing the appropriate method, with its required arguments collected from the given question. A simple scheme to store and name the methods has been designed to locate effectively the correct method. Our design is done using Object-Oriented paradigms, and the system implemented in C++.

[1] This research is based in part upon work supported by the University of Houston Small Grant program under Grant I092371.

Representing Definitions and its Associated Knowledge in a Learning Program

Kam-Hoi Cheng
Computer Science Department
University of Houston
Houston, Texas 77204-3010

Abstract

Category allows the abstraction of similar ideas into a single term for easy communication with another person. The similarities may be succinctly captured by a definition. In this paper, we are interested in the implementation of a program to acquire definitions, and use them to answer questions concerning these definitions. Several major components of a definition have been identified and implemented: given, condition, and method. The different subclasses and other associated class hierarchies needed are presented. The methods learned may be used to answer two different kinds of questions, namely the "is" and the "what" questions. The "is" question is to solve the classification problem, whereas the "what" question is to find an answer that satisfies some definition. When answering a question posted by a human, the answer is obtained by executing the appropriate method, with its required arguments collected from the given question. A simple scheme to store and name the methods has been designed to locate effectively the correct method. Our design is done using Object-Oriented paradigms, and the system implemented in C++.

Keywords: Knowledge Representation, Definition, Condition, Method, Category.

1. Introduction

Category is very important in the development of human knowledge. It allows the abstraction of a number of ideas that share some common qualities and/or quantities into one single category. One may then use this new generalized category to communicate with another person. Category consists of many different knowledge components including attributes, properties, and definition. The definition of a category specifies precisely the similarities of all objects in the category, and may be used to perform the classification task, i.e., to determine if a given object belongs to a category. Once an object has been classified to belong to a category, it will possess all the attributes and properties of that category and its ancestor categories. Recently, [CHEN06] implemented a simple hierarchy class to maintain the generalization/specialization relationships among the different categories. By using the generalization knowledge, an object may be inferred efficiently as another more general category, and inherits all its characteristics. The solution for inheritable knowledge greatly reduces the amount of teaching. Instead of teaching the same fact to many children categories, one only needs to teach such inheritable knowledge once to belong to the most appropriate ancestor category.

Definitions have been widely used in the study of many different subjects. To narrow the scope of the current development, we focus on the definitions involved in the study of number theory. This allows us to focus on developing the key components necessary for definitions without the complexity of a complicated subject. Several major components of a definition have been identified: the given, the condition, and the method. The given of a definition includes all the necessary entities related in the way specified by the definition. It includes not only the

target, but may also include the reference information. The target is what the definition is trying to classify, whereas reference are related information in order for the definition to make sense. In number theory, both the target and the reference may each be a single number or a list of two or more numbers. For example, in the definition of a number x being a factor of another number y, the number x is the target while the number y is the reference, each may be a natural number. Note that the given is also used in defining the parameters of a method discussed later in the paper. The given can be implemented in a relatively simple way and so will not be discussed further in this paper.

The condition of a definition specifies the criteria that need to be satisfied in order to classify an object to be a member of a category. Condition, by itself, is widely used in knowledge and everyday life. Besides its uses in a definition, condition is an important component of first order logic, which may be used to represent domain knowledge [RUSS03]. A large scale effort in providing a knowledge base for common sense knowledge using first order logic can be found in [LENA90] and [LENA95]. Conditions may also be used in properties of categories such as polygons. Still another application for condition is to allow a user of any program to locate entities that satisfied the condition specified by the user. Such application program may include the query function of a database, the search function in a web browser, and several interfaces of our learning program.

A method specifies exactly how to accomplish a task. There are usually several different methods associated with a definition; each may try to find an answer related to the definition. For example, one method may be to decide whether a given number is a factor of another number; another method may be to find the next factor of a number larger than some given number. A method is comprised of a number of steps to be executed sequentially. Two major uses of any learned method have been implemented. First, a method may be used as a step in another method. Second, a method is used to answer a question posted by a human. We are interested in answering two different kinds of questions, namely the “is” and the “what” questions. The “is” question is to solve the classification problem, whereas the “what” question is to find an answer that satisfies some definition. The answer is obtained by executing the appropriate method, with its required arguments collected from the given question. A simple scheme to store and name the different methods has been designed so that the correct method can be located easily when needed.

In this paper, we focus on the development of acquiring definition of categories, which includes the representation of condition, method and the different uses of methods. It is part of an on-going project, A Learning Program System (ALPS) [CHEN00]. Its goal is to develop a system that mimics human learning, and make the program a knowledgeable agent. Once the knowledge has been learned, the program may be able to answer questions concerning the knowledge just like any learned human. The program is based on a simple building block premise that complicated knowledge is built on simpler knowledge. To learn complex knowledge, we need to learn simpler knowledge first, then using those to compose the complicated knowledge. What we mean by learning a specific kind of knowledge involves the identification of its important component knowledge, and the development of the functionalities needed to maintain and to use knowledge of that kind. The system in ALPS is designed using Object-Oriented paradigm, and a knowledge object is composed of a number of simpler knowledge objects. To compose a knowledge object, one simply includes a specific instance of each of its components. The development of a complex knowledge kind is therefore decomposed into simpler sub-problems of providing the solution to each of its knowledge component. Our solution has all the advantages of an Object-Oriented solution. One such advantage is the reusability of the implementation. For example, anywhere a condition is needed; simply

including an object of the newly developed class may provide its structure and functionalities. It also allows mix-and-match of the different components and provides simpler future extensions of any component. Finally, using objects to represent knowledge allows all knowledge about a specific piece of knowledge to be stored in one single place. It makes the problem of locating the different sub-knowledge of that knowledge more efficiently accomplished.

The rest of the paper is organized as follows. Section 2 describes the condition class, its major function, and its different sub-classes. Section 3 describes the method class for those methods comprised of a sequence of steps and the different children classes of the step hierarchy. It also discusses the storage organizations of definitions and methods, and when and how to use the various methods. Problems of the current implementation and how they were handled are presented in Section 4. Finally, Section 5 concludes the paper.

2. The Condition Class and Its Different Kinds

The condition class is the base class that represents all kinds of conditions. The major function that each condition needs to be provided is to test whether the condition is satisfied by a given. Now given this condition class and its functionalities, whenever a condition is needed in any knowledge object, an object of this condition class can then be included. The actual object used is actually an object of its derived class. In this way, polymorphism of behavior can be exhibited, and future extensions of new types of conditions can easily be accommodated.

Currently, we identify five different kinds of conditions. They are called basic, composite, is-a, quantified, and enumeration conditions, respectively. The condition of a definition usually involves a combination of a number of these conditions. Basic condition is the simplest kind of conditions. In its simplest form, it involves a simple comparison operator. It may be used to compare with a constant value, for example, an angle equals to 90 degree. It can also be used to compare two different values such as the two given angles are equal. Since some knowledge objects are complicated and may have multiple aspects, the basic condition allows a specific aspect to be chosen for the comparison. For example, the height of an object may be specified for comparison. To make basic condition to be more versatile, we chose to improve the capability of a basic condition, and make it an inner product condition. It allows an associative operation to be performed to combine two values before using its result for comparison with another value. For example, the sum of two angles equals to 180 degree may be represented by a basic condition. To represent a basic condition, one needs to record its aspect, the combining operator, the comparison operator, and the comparison value. Both the combining operator and the comparison value may not be needed in some instances.

One often finds that conditions are more complicated than a single condition, for example, an acute angle is an angle greater than 0 degree and less than 90 degree. Composite condition is a condition that combines two conditions using logical operators: “and”, “or”, or “not”. Conditions involving more sub-conditions may be formed easily by using composite conditions repeatedly. An instance of a composite condition is composed of two conditions and the logical operator. Note that if operator “not” is used, there is only one condition instead of two, and the second condition is not needed.

The third kind of conditions is the “is-a” condition. It may be used in a definition to limit the domain to be a smaller set of possibilities. For example, instead of working on all numbers, we may limit the domain to be natural numbers only. This can be specified by a condition stating that it is a natural number. Similarly, to define parallelogram, one part of its condition may be

that it is a quadrilateral. The class is-a condition allows a refinement of an idea by stating that one part of the condition must satisfy the condition of some general idea. An instance of this kind of conditions only needs to record the knowledge used in the condition.

The quantified condition involves the use of quantifiers: for all or there exists. This is the most powerful/convenient kind of conditions. This kind of conditions is not only used in simple situations such as the property for regular polygons that all angles and all sides are equal. This is also useful in first order predicate calculus that may be used to represent a rich set of knowledge. An instance of the quantified condition consists of the quantifier and the specific condition.

The enumeration condition allows the specification to be a finite list of alternatives. This kind of conditions appears in situations where there is no simple relationship that can be identified. For example, the list of prepositions in the English grammar, can only be specified by an enumeration of the preposition words such as “of”, “for”, “at”. An instance of this condition kind simply consists of a list of alternative constants.

Our program uses a method, called “classify” to solve the classification problem. The “classify” method of many definitions may simply be a method that consists of a condition. Executing the method involves checking whether the given input satisfies the condition of the definition. To verify input that satisfies most of these conditions are relatively simple. For example, basic and composite conditions simply use the involved operator. For quantified and enumeration conditions, the verification basically involves the use of a simple loop, and then applies the appropriate comparison operator. To verify the “is-a” condition, it is solving exactly the same classification problem except that it is working on a simpler definition. Even though we have provided the functions to decide whether a given input will satisfy a specific condition or not, one needs to use this function in a prudent manner. This is because although the time complexity of verifying these conditions normally will take a polynomial amount of time, it may take an exponential amount of time in some cases such as in verifying quantified conditions. In many such situations, human have developed alternative methods to identity answers that satisfy those definitions, and these methods usually are comprised of a sequence of steps.

3. The Method and the Step Hierarchy

A method specifies exactly how to accomplish a task. Here, we are mainly concerned with methods that are comprised of a number of steps to be executed sequentially. To specify such methods, the following kinds of steps have been identified: method, decision, repetition, expression, operand, exception, and handle-error, each are a sub-class of the step base class. The method step allows any existing method to be used as a step of any new method. This is one of the major uses of a method because it allows more complicated method to be specified using simpler methods. The decision step allows alternative sequences of steps to be executed based on whether a given condition is satisfied or not. The repetition step allows a sequence of steps to be repeated a number of times as long as a specific condition is still satisfied. To prevent a repetition step to execute indefinitely, certain steps within the repetition must modify data that determine the outcome of the repeat-condition. The expression step allows an expression involving multiple well-known operators such as addition to be specified in a single statement. This is provided solely for convenience purposes. The operand step allows simple operations on a given operand to be specified as a step of the method, such as getting the next item in a list of items. The exception steps allows the possibility of a failure execution when executing any given step or method, such as there is no more items in the list of items when trying to obtain the next

item, or finding the next larger factor of a number when the given is already the largest possible factor. The handle-error step allows the specification on how to handle a specific exception.

Several important questions arise after knowledge has been learned. One such question is what is an appropriate storage organization such that the appropriate learned knowledge can be efficiently retrieved when they are needed? In our system, each definition is stored within a category object that bears its name. Although this storage of definition is distributed with its category, the current storage organization of categories is centralized. All categories are stored in one generic category object within the "brain" object of the learning program. This has the advantage of easy accessibility, namely a definition object can be located easily by providing its name to either the generic category object or the "brain" object. This will not pose any problem when there are only a small number of categories, but it will not be efficient when definitions of many subject areas have been learned. A better storage organization should be distributed such that definitions of one subject should be stored under its subject name. For example, storage of all definitions on number theory may in the future be stored in the knowledge object called "number". Based on this discussion, and recognizing that methods for many different definitions have similar names, the storage organization of methods is distributed among the different definition objects. In this way, although the total number of methods for all the definitions is large, the number of methods for each definition is a very small constant, and hence can be efficiently located. To locate a method, we need to locate the correct definition knowledge first, followed by searching for the appropriate method within the definition object. Searching for methods using this distributed scheme is more efficient than a centralized scheme of storing all methods in a single place.

The next questions are when and how to use these definitions and their associated methods? Two major uses of any learned method have been identified and implemented. One usage of a method is to answer questions posted by a human. A problem object is created using the appropriate method of the affected definition to represent a question. If the problem can be created successfully, our program will call the "solve" function of the problem object, which executes the method and passes the returned result back to the human user. In this paper, we are interested in answering two kinds of questions, "is" and "what". The "is" question is to solve the classification problem, whereas the "what" question is to find an answer that satisfies some definition. To facilitate an effective way to locate the correct method used to answer a question, we adopt the following convention in naming the methods. The naming scheme is based on the purpose of the method to answer a specific kind of questions. Each definition must have a method named classify, which will be used to answer the "is" question, e.g., is a given target a prime number or a multiple of a given reference number? The classify method may be formed by using the condition of the definition or that it itself is a method specified as a sequence of steps. The methods associated with answering the "what" question all starts with the word "find", followed by an appropriate adjective. The reason is that for definitions using the article "a", there are more than one "find" methods, such as find first, find next, find previous, etc. As a result, the adjective is used to distinguish the different intentions. For example, the method "find first" will be used to answer questions such as "What is the first common multiple of a given list of numbers?" etc. Using this method's naming scheme, searching for the appropriate method to solve a problem reduces to the following hierarchical scheme. The "is" problem object will locate the "classify" method of the affected definition. The "what" problem object will concatenate the appropriate adjective found in the question to the word "find" to form the method name before searching for the method in the involved definition.

Another usage of a method is that it may be needed in another method. This is accomplished by specifying the method as a step of a method using the method step object. To specify a

method as a step in another method, besides the name of the actual used-method, information about the exact argument used for each required parameter is also needed. Each argument should be known in the current method, and matches exactly the kind of information of the expected parameter. The responsibility of the method step is to ensure that each required parameter has an argument, and will check whether the correct kinds of arguments were being passed, such as a list of natural numbers, or a single natural number, etc. However, it is the responsibility of a user to provide the correct arguments to have a meaningful execution of a method. To facilitate this, each method, when inquired, will provide its user a simple English statement "explaining" what the method is computing. For example, the "classify" method of the "multiple" definition will produce a usage sentence such as "Given target and reference, classify is to determine whether target is a multiple of reference" together with the information that both target and reference are natural numbers. It is the responsibility of the teacher to provide such a sentence for each method.

4. Discussions

One notices that the problem of locating the correct method is not limited to the internal "problem" object; the external human teacher, when specifying a method as a step of another method, also faces the same problem. Suppose the teacher uses a word that is a synonym of the method name, for example, the word "compute" may be used instead of the word "find". Similarly, the teacher may initially provide such a synonym name when the method was first taught. Currently, our system does not handle synonyms, in other words, methods cannot be located using other method names that have a similar meaning. This implies that a method taught with a synonym name will not be used by the appropriate "problem" object since there is no conversion to the appropriate method name. A simple solution is to develop a function that will convert synonyms into the standardized method names during the method learning phase. On the other hand, when locating methods, the function will automatically search for synonyms from a dictionary, and try to see whether it matches one of the stored names. This remains to be a part of the future work for the ALPS project.

Notice that the solution for a "is" question may also be arrived by other means. For example, this may be obtained by using the hierarchy object developed in [CHEN05] to infer that it is of a more general category. One should be able to infer a cat to be an animal. Similarly, the answer to a "what" question may be obtained by other means such as calculating the value of an attribute of an object based on its relationships with other objects. Currently, they are treated as two different kinds of "is" and "what" questions, respectively. These solutions will be integrated into a single solution in the future. When the solve function of a problem object is called, alternative ways should be tried if one method fails.

There are several other problems associated with learning the methods of a definition. One such problem is that the computation time of many methods can be exponential in terms of the size of the input. In other words, the program will have to wait a very long time for an answer when these methods were executed. To recognize such a problem involves the learning of computational complexity. However, the learning of this capability is relatively late in human because of its difficulty. Computational complexity is taught in College, and NP theory is taught in Senior or Graduate level courses of Computer Science. In other words, most humans and even Computer Science students may never learn about these subjects. How can the computer accumulate enough knowledge and capability to learn them properly? A more immediate problem is how to recognize such a method so that it will not dominate the usage of the CPU? A

mechanism is needed to stop its computation, or control its execution. The idea of time-sharing in Operating System seems to be a reasonable solution to this problem. Currently, a new process is forked when a method is executed, as a temporary solution to avoid the possibility of an excessive wait.

The original intention of our solution is to provide an easy to use interface to teach the learning program methods interactively. Although each kind of steps is simple and consists of a small number of parts, it quickly becomes clear that it is very easy for the human user to lose track of where they are in the method, especially when the method involves multiple use of decision and repetition steps in a nested manner. The problem seems to stem from the requirement that all detailed steps have to be provided in a continuous sequence interactively. New development is required to provide interface that allows the presentation of a method in a higher level of logic, and/or in fragments under specific conditions. New capability has to be developed to combine them into a detailed low-level method automatically.

Another problem faced by the program is the correctness problem. How does the program know that the method is the correct method to solve a given problem? Why does each step needed, and is the given order of execution correct? Given the problem to be solved, and the method for it, how does the system decide that each step in the method make sense or not? What is the knowledge necessary to support the reasoning, and how to reason using the correct knowledge? All these problems will need to be addressed in the future.

5. Conclusion

Category is one of the most complicated kinds of knowledge that the learning program has to maintain. It has a number of different responsibilities, and one of them is to classify whether or not a given entity is a member of this category. A definition can be provided to accomplish this functionality. A definition consists of the given, the exact condition of the category, and the various methods to compute results related to the definition. In this paper, we have described the implementation of condition, method, step, and question using Object-Oriented paradigm.

Reference

[CHEN00] K-H Cheng, *An Object-Oriented Approach to Machine Learning*, Proc. WSES International Conference on Artificial Intelligence, June 2000, pp. 487-492.

[CHEN06] K-H Cheng, *The Representation and Inferences of Hierarchies*, Proc. IASTED International Conference on Advances in Computer Science and Technology, January 2006, pp. 269-273.

[LENA90] D.B. Lenat and R.V. Guha, *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*, Addison-Wesley, Reading, Massachusetts, 1990.

[LENA95] D.B. Lenat, *Cyc: A large-scale investment in knowledge infrastructure*, Communications of the ACM, 38(11), 1995.

[RUSS03] S. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*, 2nd Edition, Prentice Hall, 2003.