TR-030

Paradigms of Knowledge Based Software System and Its Service Image

by
Hajime Enomoto,
Naoki Yonezaki
and Motoshi Saeki
(Tokyo Institute of Technology, Japan)
Susumu Kunifuji
(Institute for New Generation Computer Technology)

November, 1983

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191 - 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Paradigms of Knowledge Based Software System and Its Service Image

Hajime Enomoto
Naoki Yonezaki
Motoshi Saeki
(Tokyo Institute of Technology)

Susumu Kunifuji
(Institute for New Generation Computer Technology)

Abstract

This article consists of four topics, which are about the very high level language and new intelligent software system. Very high level language is one of software description languages and it has facilities to describe both specifications and programs. It is useful for developing high quality softwares.

First paper describes the structure of information which is fundamental in modeling a software system. Second discusses the framework for knowledge base system used for software development. Third presents specification language which is a part of our very high level language and specification technique using it. The language is based on natural language. Final paper concerns the project of Fifth Generation Computer System in Japan. Meta inference and its applications are investigated.

1. Introduction

Recently, as many large-scale softwares are developed, many kinds of persons concern some period of software life cycle - requirement, design, fabrication, test, and operation/maintenance. In such an environment as this, it is important for the persons being concerned with softwares to communicate exactly with each other about software systems. This means that each person concerning some period should understand the ideas of persons concerning another period and a framework of software which is a common concept throughout the life cycle. It is preferable that a software development has a software description language which can be used throughout a software life cycle. We call this language a very high level language and it has facilities to describe both specifications and programs in the same form. This language is used not only in requirement, design, and fabrication periods but also in test and operation/maintenance periods as documents. The unification of the concepts of various languages used in each period of a life cycle enables us to understand many aspects of described software.

Huge amount of researches about languages or methodologies have been studied, however each of them concerns a single period of a life cycle. For example, ISDOS (14) and SREM (15) are for requirement stage, stepwise refinement (8), structured programming (7), Parnas's modularization (1), Myer's composit design (5), abstraction (3), and Jackson method (4) are for design stage. To construct high quality softwares, the persons touching software development should know many languages or methodorogies, where concepts used are so different that it is a heavy load to master them.

This article consists of four topics, which are about the very high level language and new intelligent software system. First paper describes the structure of information which is fundamental in modeling a software system. Second discusses the framework for knowledge base system used for software development. Third presents specification language which is a part of our very high level language and specification technique using it. The language is based on natural language and its semantics is given by formal logic. Final paper concerns the project of Fifth Generation Computer System in Japan. Meta inference and its applications are investigated.

2. Information and its representation in natural language

2.1 Introduction

The basis of new concept of software systems is considered as exploring a information representation method which is comprehensive for persons and is efficient for machine processing. In this section, the structure of information are explored and categorized from the communication view among people, since the essential property of information is moving around human society. Then we consider comprehensive representation scheme in natural language. Especially, it is interesting to discuss whether representations of information depend on kinds of natural languages, e.g. English or Japanese.

2.2 Category of information

In order to communicate with each other, people use various semiotic phenomena being interpreted both syntactically and semantically into meaningful information. Informations got by our eyes and ears are the most fundamental and they appear as acoustic sound, pattern and picture. Information has coordinate pair of a media and a message. Communication is phenomena that a media conveys a message and there exists at least two individuals, sender and receiver in the media. The sender behaves as an initiator of representing action and the receiver interprets a products of its action.

Information can be categorized by the relations between a media and a message into one of signal, symbol, metasymbol and index.

(1) signal

Signal is defined as the message information such as media A triggers message B. Then message B is accompanied with features of media A as information source. A typical example of signal is traffic light A which corresponds to medias. Green, yellow, and red lamp signals trigger motion of go, wait, and stop respectively as messages. In this case, interpreter functions are activated by human vision sense and these signals belong to phenomena signs.

In the case of a telecommunication system, a media consists of channels, and messages is transmitted as signal in the channels. An interpreter corresponds to encoder and decoder which translate information sources so that it matches to the channel. For information processing, a media is put/get commands and channels or sensor. A message is telesignal. Interpreter functions are given by encoder and decoder.

Above examples show that a media is closely related to messages. Therefore, telecommunication channels should be match to the statistical properties of transmitting messages, and encoder and decoder functions too, like Huffman coding.

Signal has a syntax and it depends only on its generating mechanism. Therefore signal has no definite semantics because signal has no direct semantical information itself. However, signal is a concept of abstracting the mechanical generations of semantics.

(2) symbol

Symbol has both syntax and semantics, and it is either interpreted or pointed. Then we can understand or recognize the object represented by a symbol. For symbol, concepts of type, token and standard frame described later constitute syntax ,and interpretation or pointing mechanism corresponds to semantics. In the case of information processing, these features should be expressed by mathematical basis such as logical formlas.

In general, each symbol is considered to be a symbolized sign in order to express an abstracted concept of corresponding object. This enables us to consider that media corresponding to a language system specifies messages as sentences. Each word of the language is associated with an object having corresponding meaning. Meaning of symbol should be expressed as the corresponding properties of each word. The relation between them constitutes a rule system, which is considered as semiology. Geometrical features of pictures belong to this category.

(3) metasymbol

Typical example of metasymbol is picture. Picture can tell many things, however we can not speak about contents of a picture completely and we may paint many kinds of pictures for one scene. Furthermore, we find many new matters for a scene or picture whenever we observe it carefully. This means that metasymbol has only partial syntax and semantics. In order to process or recognize metasymbol, complex usage of metasymbol and symbol is very useful and efficient such as usage of geometrical features to recognize pictures. For metasymbol, a media is an original naturally generated object and a message may be obtained as signal by structured encoding scheme such as scanning or by explanation sentence of symbols.

(4) index

When syntax and semantics have a structure, it is necessary to get structure of a specified part by index or by indicating attribute value in the structure as index. This means that index is used as tool for part indication or association as a whole. Therefore an index indicates such a relation that a media corresponds to structure indication system and a message has a role to indicate partial structure.

For symbol, concept of attribute can be used as index. Attribute structure behaves as a media and actual attribute name and its value plays a role of a message. Common noun has Has or Part-of relation and these relation can be expressed in hierarchical attribute structure. Adjective modification can be used for indicating subset of a set shown by common noun. Proper noun is used for label naming as an element of the specified set, so it is considered as index.

Similarly a verb has an attribute which specifies an acting manner as an adverb.

Another kind of index is produced by some association effect. Typical examples are that fire is associated with smoke by denotation and pigeons are associated to peace by connotation.

Fig.2.1 shows the properties of signals, symbols, metasymbols and index. Fig.2.2 indicates fundamental functions for service. Usual information service uses symbolic processing by encoding to and decoding from signal.

usage entities	Interpretation	examples	role of index
signal (syntax)	encoder, decoder	Voice, TV waveform	part indication
	sense organ	work, sign	association for object
symbol (syntax, semantics)	logical definition	graph, artificial language	part, connection and structure indication
	inference by environment	natural language	denotational and connotational association
metasymbol [=icons] (partial syntax, semantics)	partial association connotation	picture, scene	part indication by feature expression mapping to symbol by feature

Fig.2.1 Properties of signals, symbols, metasymbols and indices

output / input	signal	symbol	metasymbol
signal	Transmission storage	decoding	structure decoding
symbol	encoding	processing	global synthesis using features
meta-symbol	structure encoding	partial understanding of features	reproduction through symbolic interpolation

signal service : media
symbol service : data processing
metasymbol service : interpretation
index service : database retrieving
communication service : compound and interactive service

Fig.2.2 Fundamental function for information service

2.3 Language culture and information representing scheme

Since information representing scheme partly depends on language culture, software development method concerns the character of a nation which reflects the language culture.

Fig.2.3 shows the comparison of grammatical structure of English with that of Japanese. Roughly speaking, the number of syllables of spoken Japanese is rather small and characters of written Japanese have large varieties such as Kanji and Kanas which has two kinds, Hiragana and Katakana and corresponds to phonetic symbols. Furthermore paradigm of Japanese is flexible but not complicated. Especially particles such as 'ga', 'wa', 'o', 'ni' play important role in the paradigm.

	English	Japanese
Number of syllables	about 2000	about 120
No. of syllables /sentence	20% ~ 30% long	
Length of char. /sentence	nearly equal	
Paradigm for	strictly determined	flexible
1. case 2. person 3. number 4. tense	vary vary vary many categories	by particle fixed fixed a few categories
No. of abstract nouns	>	
Expressions	dynamic	static

Fig.2.3 Comparison of grammatical structure of English with that of Japanese

Principal features of Japanese language are listed as follows:

1. Subject particles 'ga' and 'wa' enable us to discriminate subjective view point (well known) from objective one (unknown), and there are many roles in usage of particles.

2. High capability of making coined words is equiped by using conversion particles to noun from noun, verbs or adjectives in many foreign languages such as English, French and so on. This increases the transparent structure of the language and association capabilities.

3. Conjugations of verbs and adjectives provide fact description power for following cases.

   (a) Forms
   Future form, participal adverb form, stopping form, participal adjective form, past form and command form.

   (b) Concatenation facilities of auxiliary verbs.
   By the conjugation rules facts such as assertion, estimation and remembrance for event or state can be described in a short form.

4. Association mechanism from sequence of sentence exists and abridged sentences are frequently used.
5. Scene description capability by simple sentence exists. Typical examples are seventeen syllabled verse (Haiku poem) and Japanese verse of 31 syllables (Tanka poem) including rhyming.
6. Semantics of Japanese language can be interpreted by analyzing and integrating local conjugation properties, bottom up analysis and association facilities by use of partial pattern matching and referencing main syntax paradigms.

These features are closely related to information analysis and information representing form. In general Japanese have excellent skill for bottom up fine synthesis of object along some model, and also have an association abilities by implicit descriptions. These abilities connect with good pattern matching caused by usage of chinese characters having complicated patterns. High capability of making coined word makes adaptive introduction of foreign civilization into Japanese society possible. Typewriters are not used widely because selection mechanism of Japanese characters is more complex than European alphabets, but Japanese word processors are now intensively developed and some office environments in Japan are now changing by using them. Therefore, Japanese computer languages will be developed in near future and they should have some abilities for effective usage of Japanese language features. But mutual mechanical translation between Japanese and foreign languages is now underway in order to further increase international coordination. If we limit for computer language, mechanical translation is rather easy.

We think Japanese language features will be suitable for static and symbolic representation of specification.

3. Knowledge base system for software development from the view point of service

3.1 Introduction

Service is an offering process of a work specified by some client. The content of the work is given by intention and interactive communication between clients and a service system, then service system do the work defined by the specification. Working procedure is obtained by protocol specifications.

Service depends on the category of information and has very complex structure. So, integration of services is necessary today. This section discusses a knowledge based system as a service system.

Then a frame of a language for stepwise specification of problems throughout life-cycle of service software is discussed. In the stepwise specification process, a definition of a work is divided into individual subworks and selfdoing subworks. They are given by respective specifications of dynamic flow and managing process. Specifications derived by such stepwise derivation give finally executable program. Many databases are used as for the purpose of stepwise specification throughout these subcontracting works.

3.2 Framework of knowledge base system from service view point

All knowledge based systems should be organized in such a way as knowledges of many experts can be easily implemented in the system and many kinds of knowledges can be integrated as unified knowledge database by adding suitable mutual relations among individual knowledges. Therefore it is desired that some suitable very high level language is utilized widely by many users specialized in some specific field and system manager can integrate knowledges of experts by using the very high level language.

From above description, concept of service becomes important since many knowledges of experts implemented in the system should be utilized through service network. Service is an offering process of some works responding to requests of customers, and it is requested that cost of the work is economical, and that service procedure of the work is simple, pleasant and convenient. Therefore, knowledge based system is considered as one of the functions of nodes in service network and this system should be conveniently offered for other persons. Also useful knowledge services are easily organized and an evolutional service system should have adaptabilities to their own service scheme.

Software life-cycle includes consultation, requirement analysis, design, implementation, maintenance, revisement, and training. All of software tools in the life-cycle should be written in single very high level languages and some integrating mechanism is requested in order to implement knowledge databases used in software development. Communication mechanism is also one of the fundamental functions that tells us how to organize node elements in network or terminal facilities.

Service procedure is as follows:

1. A client has an intention for service and selects a suitable service system by understanding the service system specification.
2. Client tells his intention to the service system and complete intention is

obtained through interactive communication.

3. The specification of the work offered by the service system is determined by retrieving the knowledge database and input informations, and then service system confirms the specification for the client.

4. The service system divides the work specified by the specification into a set of subworks and specifies their specifications and timings of subwork executions.

5. The service system determines mission for its own subworks and other subworks given as suborder to several subcontractors as the subservices. Simultaneously set integrity conditions which must be satisfied by the subworks are obtained.

6. A mission specifies the procedure excluding the manner of cooperational concurrent operations and resource managements.

7. The cost is calculated and the contract of the work is concluded.

8. Executions of the divided subworks are done and the integrity conditions are checked.

9. Result of work is offered to the client and he checks the result.

10. If the result is not satisfactory to the client, he raises complaint to the service system and the system will make some revisement for the result.

11. The client pays for end of service or result of service in the specified period.

In the above description, there are many service module descriptions. Each service module has a service specification which specifies the service content able to offer, that is, describing the service requesting syntax and semantics. A client requests a work as an intention to some service systems only using the words indicated in the service specification. The service system determines the corresponding specification of the works and manages the execution of them. In some case, the service system proposes specifications to an expectable client. If some mutual coordination is necessary to arrange subworks, conference of correlated subcontractors will be opened to determine a disjoint interface of subworks.

By the service scheme described above, verifications of each work offered by individual modules will assure of correctness of the total service. A basic support system may consist of a network, communication and service protocols of it. Protocol specification has a role to specify procedures for executing the works requested by a client using the content of the service specification. Communication protocols specify the manner of interactive communication procedure in order to determine intention of client.

If clients use nest structure for specification frame, the work corresponding to nest structure is described as adjective or adverb modification depending on noun or command form for the term in an intention. For conjunctive statements in an intention, the service system can divide the work into subworks corresponding to individual conjunctive elementary terms, and the integrity conditions must be satisfied by the result of the subworks. These procedures are executed concurrently by subcontractors because mutually non-correlated subworks can be executed concurrently. Even for subworks which have a small number of shared variables, a suitable algorithm converging to a fixed point can be available checking the integrity conditions.

Service entities are represented as common nouns in service specification using natural language and it is necessary to introduce some common noun variables in protocol specification of corresponding service. These common nouns have different properties from each other, that is, common nouns used in the specification are limited to a definite element by several Is-a relations, comparing that common nouns appearing in protocol specification specify properties of variables belonging to the set of the common nouns in order to generate procedures to execute the work.

Common nouns mentioned above have both syntax and semantics, but common nouns appearing in data type or abstract data type are concerned with their attributes. This means that type is used to define structure or syntax. Structures correspond to Has-a relations and they are defined hierarchically by declarations.

Specification body defines semantics as a set of conjunctive relations of common nouns and each relation is expressed as Is-a relation, in general. Functions are defined as a set of Is-a relations between input elements and output elements which are expressed as attribute values. Attributes of a common noun have two kinds of Has-a relations. One is having 'property of', the other is having 'part of'. 'Property of' is represented by an adjective and 'part of' is represented by common noun object.

There are three kinds of definitions of verb. First kind of verbs is stationary verbs. A intensional verb is defined as a relation between states before and after a specified time interval as a step movement. Second kind of verbs has a fixed point of the goal which can be defined as recursive representation having the least fixed point. Last one gives some effects such that one gives it to another or show intention of subject itself. These verbs shows change of configuration of system. If we consider duality between verb and noun, we can define syntax and semantics of verb as similar to those of noun. Verb has two kinds of attribute such as adverb and complementary or objective noun word.

Some words of adjective, adverb or some definitive purpose words can delimit a word with their own property and type of delimited word may have generic property. Therefore generic type should be introduced for parameterized type.

3.3 Paradigm of the language and system feature

Usage of framework mentioned above causes a paradigm of software system based on the service concept. By using this paradigm we can adopt natural language like language having following features.

1. Since the language is a limited paradigm of sentences and natural language like one, users can easily learn it and it has no ambiguity.

2. In the software system employing service concept, we can use implicit expressions.

3. Specification body is expressed as a conjunction of statements which are mainly described by Is-a relation. Then concurrent processing systems are easily implemented.

4. Data type is expressed by symbolic Has-a relations.

5. Concept of category of information can be introduced into definitions of data type. Data have type name and information category name, e.g. signal, symbol and so on. Algorithm for detecting category of information is applied to all data and algorithm for data processing is applied to data having the same category.

6. Modifications by adjectives, adverbs and relative pronoun clauses are used for hierarchical and independent description.

7. Service and protocol specifications are separately described. (4) Protocol specifications can be described in such a way as implementation methods of service specification is independent on the content of service specifications.

8. Indefinite and definite articles are used, since referencing functions for a common noun, and proper nouns are used to identify a individual object by naming it. Pronouns are used for pointing it. They are used for static representation.

9. Auxiliary verbs are introduced in order to introduce modal operators and passive form is considered as a state representation.

These features may serve such purposes as modularization, information hiding, and localized software generation are easily achieved and written specifications are easily understood. These facts are deduced from the fact that hierarchical independency and orthogonal description are possible and every description in this language can be translated into logical expression and vice versa.

If we implement the powerful interpretation mechanism of this language, downward and upward implication are possible. Downward implication is a function as infer some neglected attribute of object word, and corresponds to denotational level. Conversely, upward implication is such a function that an example is referred to its class object name corresponding to connotational level.

There are three kinds of integrations of the system from the service view point. First one is resource integration. This can be achieved by transparency of media and interface. Generalized functions for resource integration are obtained by assembling orthogonal or independent information hiding service functions feature and total management of resource. Secondly, integration is cooperative one. Network architecture of cooperative parallel processing, non-deterministic module representation in mission representation of standard synchronization function and independency of subworks with resource, are important for cooperative integration.

Last integration is evolutional one. This is achieved by renewing functions considering their effectiveness, affinity of resource and cooperative integrations with new function, and improving functions of total management.

3.4 Software development and knowledge base

In order to develop large scale software and integrate various systems, life-cycle concept is important. Analysis of information itself is requested at first and it is necessary to develop some conceptual schema for contents represented by natural language and picture, because they are most fundamental means representing human thinkings.

Montague grammar (1) may be one of the excellent tool for understanding and expressing a meaning of natural language. It satisfies Frege's functionality principle and discuss the meaning of denotation and intension. Universe of all individuals is denoted by D and possible state of affairs can be thought of as an index $i \in I$.

Montague's intensional logic (2), have types and its syntax is a set of construction rules for terms from primitive symbols and some modal operators. Semantic is given by standard frame and standard model and value assignments. Terms are characterized recursively by the standard frame based on D and I.

The standard model is a system discussed by standard frame and accompanies meaning function. This function is a mapping which assigns to each constant a function from I into an element whose index is the same type as the constant of standard frame. Value assignment functions have a role of definition of the assignments over on the set of variables which is independent on indices.

Many researches in programming methodology emphasize an importance of structure representation, modularity, locality, information hiding, representation based on genuine structure of data and etc. Furthermore importance of independency and orthogonality should be given more weight to software development. If an intensive independency and orthogonality are included in a software specification by describing works in static way, relations independent on specification will be introduced in description about executions. Independency between definitions of simultaneous linear equations and its solving method is a good example. Introduction of various independency and orthogonality concepts will have many good effects on maintenance.

Montague's intensional logic can be used for independent and orthogonal description of software and is suitable for knowledge based development software. The structure of syntax and semantics have hierarchies. Meaning functions may be implemented as knowledge database. In addition, many useful words are organized as keys of knowledge base, inference rules should be organized as retrieval mechanism of database. These schema may be suitable for evolutional knowledge base and natural language like description of softwares because specifications can be revised as their analysis advance and mission may be obtained by adding suitable combining commands.

Furthermore concept of service network is useful for software development especially for picture service systems. Pictures are structure encoded as signal and geometrical features, e.g. cusp points and structure lines (3), which may be useful for understanding of picture. Various transformation services of picture are obtained in signal and symbol level. Typical examples of signal service are fast fourier or fast hadarmard transformations. Then we can consider that the various service belongs to the same level.

Concepts of service concern implicit expressions of target work. Implicit descriptions may be classified into two kinds in this case. First one is abbreviation and second one is meta statements. For abbreviation, its supplement is necessary by using some tools and skill in the service system. Meta statements consists of metacommands and metaobject. Both of them uses high level service names which are not defined explicitly. Service system will choose suitable plan and resource from user's meta statements and define the concrete procedures and execute the demanded user's works. In this procedure the system must utilize effectively its skills stored as knowledge.

[REFERENCES]

(1) Montague,R. : The Proper Treatment of Qualification in Ordinary English, Approaches to Natural Language, Reidel Dordrecht (1973).

(2) Gallin,D. : Intensional and Higher Order Modal Logic; With Applications to Montague Semantics, North-Holland Publishing, Amsterdam (1975).

(3) Enomoto,H, Yonezaki,N, Watanabe,Y., and Saeki,M. : Towards Evolutional Structure for Database of Image and Object Body, Proc. of 1st Australian Conf. of Computer Graphics, 1983

(4) Bochmann,G.V. and Sunshine,C.A. : Formal Methods in Communication Protocol Design, IEEE Trans. Commun., Vol.COM-28, No.4, pp.624-631 (1980)

4. Specification technique using natural language (16)

4.1 Introduction

In this section, new specification technique using unambiguous natural language (a fragment of English) will be introduced. Specifications of software systems have been written in natural language until now. Natural language is too ambiguous to express specifications strictly and exactly, so that readers of them might misunderstand what they present and that semantical processings with computers, e.g. automatic verification and document reconstruction, might be impossible. Then specification description method based on logic or algebra whose specification body is a set of axioms are studied. (8),(9),(13) Although this methods provides rigorous semantics, it is difficult to read for untrained persons in mathematics. Natural language excels not only in readability and understandability but also in constructibility of specifications.

Hierarchical decomposition of a software system is important for us to represent software systems easy to understand and specification language should support this facility. It has been studied by many researchers. (1)~(5) Our method gives a new decomposition principle based on lexical decomposition manner in language theory field, which is different from that proposed before. Decomposition principle is closely related to constructibility of specifications and we think that our lexical decomposition method have enough constructibility in addition to understandability.

The ideal specification language is simple natural language which has unambiguous formal semantics. One of main objectives of this research is to develop a subset of natural language satisfying such properties.

We make following assumption as to the natural language.

(i) Object software system can be described in the natural language.

(ii) For each software module, there is a word whose concept corresponds to it.

(iii) We can select a word which can reminds almost all persons who read a specification of the same concept.

In general a software system is a modeling of real world. A word in natural language is used to represent a unit of concepts. If we select a word which represents a concept of an object or a matter in real world modeled by a software system, we can image the system at least vaguely by the word.

Of course this idea is usually used, when we write programs. It is just the same as an appropriate label is assigned to a program as it name.

Our main idea is usage of restricted natural language and that meaning of a word is specified by a set of sentences which explain the system identified by the word in the natural language. Meanings of the words used in the sentences except for reserved words are specified by a set of sentences hierarchically. This specification method correspouds to the notion of lexical decomposition. The syntax of the restricted natural language is so restricted that selection of words which corresponds to module decomposition is guided.

Other advantages of restriction of syutax are that the semantics of sentences becomes unambiguous and that it provides readability of sentences. When we read a specification in one level, although precise meaning of words used in the sentences of the specification are unknown, if words are selected so suitably that almost all people can image a fact or a matter in real world modeled by the module, we can image a model specified in the level or we can make assumptions which will help to understand next lower level specifications.

4.2 Specification technique

The specification technique is outlined as follows. The specifications are written hierarchically in natural language according as the meanings of the words used in them are defined in lexical decomposition manner. In brief, the meanings of words are defined using a collection of sentences in which other words occur. We consider that each word corresponds to a software module, so in this specification technique, software systems are decomposed hierarchically into smaller software modules according to occurrences of words in sentences of specifications. Furthermore, this decomposition method is natural to images which people have for software systems. Thus specifications can be written in simple and comprehensive English sentences. Because readers can image the system vaguely by the word.

Specifications written in English are translated into logical formulas by the method based on Montague's. (9) In consequence, strict meaning is assigned to the specification ,so that semantical processings with computers are possible. Translation rules are automatically generated from definitions of words used in the specification or associated with syntax rules of English. The meanings of general words, for example, 'is', 'are', 'a', 'no', and so on, have already been defined. The way to define the meanings of words is
1) a collection of sentences which is semantically equivalent to them or
2) a collection of sentences which represent the relations between the other words such as the operation definitions in axiomatic abstract data type specification. (10),(11),(17)

Fig.4.1 shows the overview of our method.

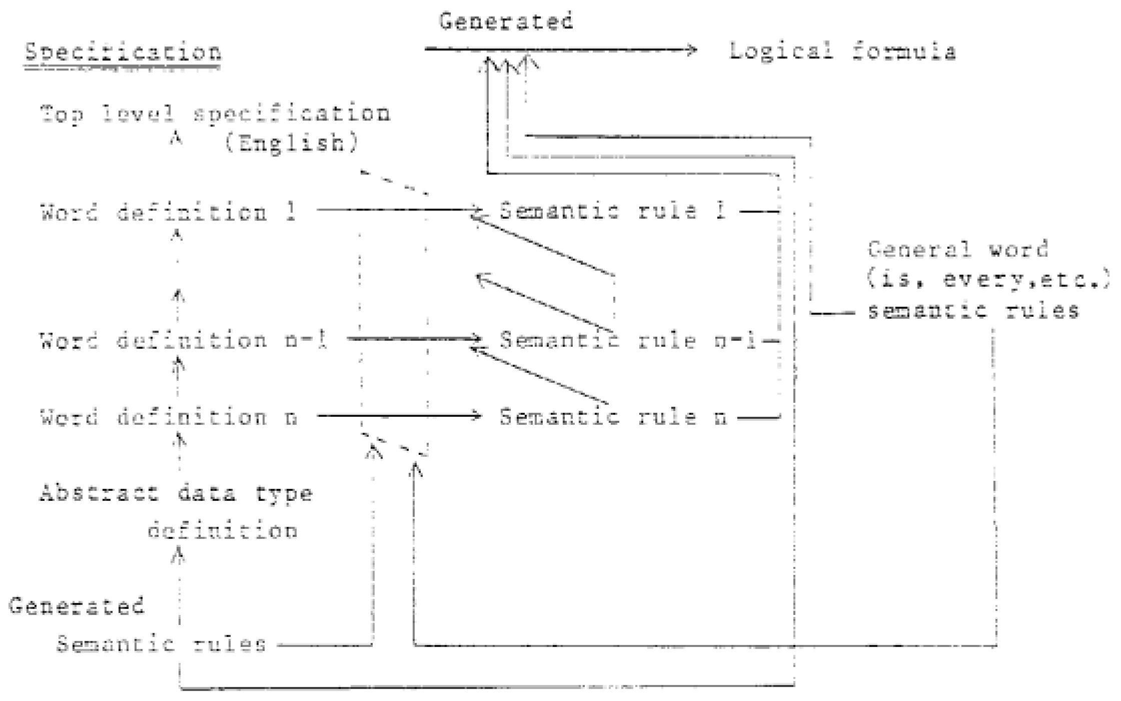

Fig.4.1 Specification technique

It is difficult for untrained persons to construct the specification of an object system using abstract concept, (1),(6) e.g. abstract data type. (10),(11),(17) Specification methods should provide specifiers easily understandable guidelines for the construction of specifications. Our specification method is based on lexical decomposition of words, but on the other hand specifiers have to make the words correspond to modules which represent mathematical concepts, for example, function, predicate, abstract data type, and so on. In the following, we present as follows natural and clear way how correspondence of words to mathematical concepts and hierarchical structure of decomposed modules are explored in natural language description of object systems. In brief we write specifications hierarchically using guidelines by natural language features, e.g. syntactic category of words, relations of modifications and of references among words, cases of words, and so on. The process of the construction of specifications are shown as follows using example of eight queens' problem.

1) Describe the content of the object system, .e.g. its purpose, its constraints, and so on, in natural language. Then draw lines under particular words and phrases which occur in these sentences.

[ex.]

This program computes the <u>arrangement on a chessboard</u> in which eight <u>queens</u> are <u>placed</u> and no <u>queen checks</u> against any other <u>queen</u>.

2) If all the underlined words have been already defined, go to step 4. Otherwise define the meanings of words by the sentences in natural language in the same way as step 1.

[ex.]

a) Queens are entities and there are eight queens.

b) Arrangement on a chessboard is a set of chessboard's square.

c) That queen q1 checks against queen q2 in the arrangement means that they are on the same row or on the same column or on the same diagonal in it.

d) That queen q1 is placed in the arrangement means that q1 is included in the arrangement.

3) Repeat step 3 until there are no undefined words or undefined words are too basic to be defined.

[ex.]

a) That a pair of queens are on the same row in the arrangement means that the x-coordinates of their positions in the arrangement are equivalent.

b,c) The definitions of 'on the same column' and 'on the same diagonal' are similar to above.

d) Chessboard's square is 8 × 8 entities and has x,y coordinate.

4) Make directed graph whose nodes are words underlined words and whose arcs are defined by reference relation and modification relation between the words. This graph determines the hierarchical structure of the definitions of the words and it obeys scope rules of definitions of as in the case of programing languages. The hierarachical structure of words reflects that of the modules into which the object system is decomposed.

[ex.]

Fig.4.2 shows the directed graph generated from the above example. Solid arcs and dotted arcs show reference relations and modification relations respectively.

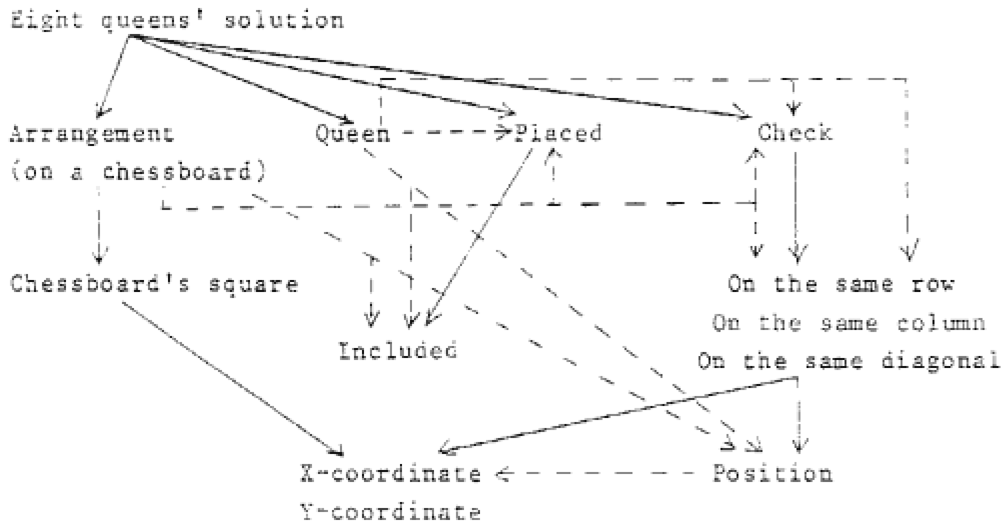

Fig.4.2 Reference and modification relation

5) Determine modules corresponding to the words using informations of them such as syntactic category, case, modification relation, and so on.

5-1) In a sentence, the verb corresponds to a module name and the objectives, subject, and the modifiers to the verb are inputs or outputs of the module. When the main verb is be-verb, the complement is a module name, and the subject and the modifiers to it are inputs or outputs. But the objectives and the modifiers are often omitted from the sentence. In such a case, we must consider what is omitted and make up for these ellipses. Considering the contents of ellipses corresponds to inputs and outputs of the module.
[ex.]

placed, check, on the same row
'Placed'module has queen and arrangement as input-output arguments.
...

5-2) Common noun which is the subject, objective or modifier to the verb or to the complement corresponds to the abstract data type or operation of the abstract data type. The latter is the only case that the common noun is modified by other common noun and modifiers to it correspond to inputs or outputs of the operation.
[ex.]

type : arrangement, queen
operation : x-coordinate, y-coordinate

5-3) Material nouns correspond to abstract data type modules.
[ex.]

queen, chessboard's square

6) Describe modules according to syntax of specification language mentioned later.

Fig.4.3 shows the finished specification of eight queens' problem and our specification language will be introduced in the next section using this example. It should be noted that function 'length[X]' and 'element[q,X]' have been defined to sequence type and they compute the length of sequence X and q-th element in X respectively.

```
Arrangement X is a eight queens' solution
 means that
    1) Eight queens are placed in X .
    2) No queen is checking against any other queen in X .

    Queen q1 is checking against queen q2 in arrangement X
      means that
         1) Q1 and q2 are on the same row in X or
            q1 and q2 are on the same column in X or
            q1 and q2 are on the same diagonal in X .

      Queen q1 and queen q2 is on the same row in arrangement X
         means that
             1) The X-coordinate of the position of q1 in X
                  is the X-coordinate of the position of q2 in X .
      end  on the same row ;

      Queen q1 and queen q2 is on the same column in arrangement X
         means that
             1) The Y-coordinate of the position of q1 in X
                  is the Y-coordinate of the position of q2 in X .
      end  on the same column ;

      Queen q1 and queen q2 is on the same diagonal in arrangement X
         means that
             1) /X-coordinate[element[q1,X]] + Y-coordinate[element[q1,X]]
                  = X-coordinate[element[q2,X]] + Y-coordinate[element[q2,X]] v
                 X-coordinate[element[q1,X]] - Y-coordinate[element[q1,X]]
                  = X-coordinate[element[q2,X]] - Y-coordinate[element[q2,X]]/ .
      end  on the same diagonal ;

   end  checking ;

   Queen q is placed in arrangement X
      means that
          1) q is included in X.
   end  placed ;

   Arrangement is type
        sequence of  chessboard's square
   end arrangement ;

   Queen is type
         [1..8]
   end  queen ;
 Chessboard's square is type
       (X-coordinate: [1..8] , Y-coordinate: [1..8])
 end  chessboard's square

lexicon
     q is included in X
          := length[X] in sequence ;
     p is a position of q in X
          := element[q,X] in sequence ;
     c is a X-coordinate of y
          := X-coordinate[y] ;
     c is a Y-coordinate of y
          := Y-coordinate[y] ;

end eight queens' solution
```

Fig.4.3 Specification of eight queens' problem

4.3 Specifications of functional systems

Happily, the previous example, eight queens' problem, is simple because input and output relations of modules are essential in the specification. Such systems as this are called functional systems and in this section we present specifications of functional systems using our specification language.

The syntax of a functional module is as follows.

```
<defining sentence>
  means that
       <specification body>
       <sub functional modules>
       <type specification modules>
       <lexicon declaration>
  end
```

Defining sentence declares the module name and its input and output parameters in natural language form. In functional module, module name is a common noun or an adjective (phrase), therefore only be-verb is a main verb in sentences of specifications. Defining sentence is a typical sentence in which the module name is used. Specification body is a set of itemized sentences which present the conditions that input and output parameters of the module should satisfy. Those sentences can be written in arithmetic expressions or logical formulas in the case that they seem more comprehensive than sentences of natural language, for example, in the case of 'on the same diagonal' module. Sub functional modules part are a set of specifications of sub modules into which the module is decomposed. The syntax of the part is the same as that of the module. In Fig.4.3, 'eight queens' solution' module has 'checking' and 'placed' module as a sub functional module.

Type specification modules part is the set of specifications of abstract data types and its specification technique is based on axiomatic approach. (17) Those modules are written according to the following syntax similar to functional module.

```
<type name> is type
   <type specification body>
   <sub functional modules>
   <sub type specification modules>
   <lexicon declaration>
end
```

Type name is a common noun (phrase) and in the case of parameterized type it is modified by preposition phrases whose objectives correspond to type parameters. Sub type specification modules are a set of type specification modules. In the section of type specification body, the operations of the data type are defined and it is written in the form,

```
functions
    <defining sentences>
predicates
    <defining sentences>
satisfy
    <specification body>
```

or

```
<type expression> .
```

Operations of abstract data type are divided into functions and predicates. A defining sentences declares an operation, that is to say, it includes informations of an operation name, domain, and range. Operation names are also restricted to common noun or adjective. Specification body part is the essential part of the specification and presents the relations which hold among the operations as axioms. It is described in the same manner as that of functional module. Type expression is used to define data type using general purpose basic type or structured type such as integer, string, enumeration type, record type, function type, set type, sequence type and so on. Its syntax is just like one in Pascal. Operations to data type represented by type expression are automatically defined like (12). In Fig.4.3, chessboard's square is defined using record type declaration in type expression and constructor ( ) , selectors x-coordinate and y-coordinate, and their axiom, e.g. x-coordinate[(x,y)]=x are automatically defined.

Lexicon declaration part is used to define new words which are available in the module. In Fig.4.3, new word 'included' is defined using function 'length' to sequence type.

Finally it should be noted that our hierarchical specification method imports usual scope rules, so that the same words can be used for different meanings and synonymous words can be defined.

4.4 Specifications of concurrent systems

This section is devoted to our specification method for concurrent systems. The description form is basically the same as functional module. One of the main different aspect is that they need specifications about timing of actions of the object system such as synchronization, mutual exclusion, and so on. In our method, concurrent systems consists of cooperating processes and shared resources which are accessed by them. Usually resources can be specified as the abstract data type whose operation specifications are added to their synchronization specifications. We call it abstract monitor type.

We explain specification description method of concurrent modules using example Fig.4.4. The example is n-producer m-consumer problem, which consists of n instances of the producer process, m instances of the consumer process and an instance of the buffer abstract monitor type.

```
N-producer and m-consumer problem is the system of
  Configuration
       1) There are n producers.
       2) There are m consumers.
  Timing
       1) Initially every consumer is starting to consume.
       2) Initially every producer is starting to produce.
       3) Initially buffer is empty.
       4) Every producer which finishes to produce will begin to produce.
       5) Every consumer which finishes to consume will begin to consume.

He produces means that
  Timing
    1) Initially he begins to generate .
    2) He finishes to generate data x
         → he will begin to write data x to buffer.
    3) He finishes to write to buffer
         → he will finish to produce.

    He generates data x means that
      Timing
        1) He begins to generate
             → he will finish to generate.
    end  generate ;

 end  produce ;

He consumes means that
  Timing
    1) Initially he begins to read from buffer.
    2) He finishes to read data x from buffer
         → he will begin to use data x.
    3) He finishes to use
         → he will finish to consume.

    He uses data x means that
      Timing
        1) He begins to use data x
             → he will finish to use.
    end  use ;

 end  consume ;
```

```
  Buffer is abstract monitor type
     functions
          I write data x to buffer B
          :=  _iwrite[B,x] : (buffer,data) → buffer ;
          I read data arg2[read[B]] from buffer B
          :=  _iread[B] : buffer → (buffer,data) ;
          Buffer B is empty
          :=  ∅ ;
    satisfy
       1) /_jread(_iwrite(∅,α)) = (∅,α)/.

    Timing
       State predicate
          at :: begin [v], waiting  [adj] ;
          in(1) :: 1°ing  [adj] ;
          after :: finish  [v] ;
      1) Producer i begins to write data x to  B
             and B is not empty
            → producer i is waiting to write data x to B
                   until B is empty .
      2) A producer begins to write to B
           and B is empty → one of producers
             which begin to write to B is writing to B.
      3) consumer i begins to read from B
            and B is empty
           → consumer i is waiting to read from B
                   until B is not empty.
      4) A consumer begins to read from B
           and B is not empty → one of consumers
             which begin to read from B is reading from B.
      5) A producer is writing to B
           → there is no consumer which is reading from B .
      6) A consumer is reading from B
           → there is no producer which is writing to B .
  end  buffer ;
end  n-producer and m-consumer problem
```

Fig.4.4 Specification of n producers and m consumers problem

Concurrent module specification is described in the form,

```
<defining sentence>
  (means that)
  <configuration specification>
  <state declaration>
  <timing specification body>
  <sub concurrent modules>
  <sub functional modules>
  <monitor specification modules>
  <type specification modules>
  <lexicon declaration>
end
```

Defining sentence declares a module name, and input and output, but it is different from that of functional module in the point that a module name must be verb except be-verb. Sub concurrent modules part specifies sub-processes and monitor specification module part specifies shared resources. Sub concurrent module is described in the same manner. Functional module and type specification module are allowed as sub-modules of the system.

Configuration specification part presents the number of sub-processes (modules) and the number of shared resources using the sentences in natural language. Those numbers are fixed during execution time of the system, i.e. no instance of sub-processes and shared resources are generated dynamically.

In state declaration part, words (or phrases) whose syntactic category is verb or adjective are declared to use for identifying an execution state of the processes specified by the module. If the part is omitted, words which is modified by infinitive or gerund of verb corresponding to the concurrent module distinguish its execution states. In Fig.4.4, 'begin', 'finish' are declared implicitly to distinguish the execution states in 'produce', 'consume', 'generate', and 'use' modules.

Timing specification body consists of the initial conditions, timing and control specifications and input-output relation between its sub-modules. They are written using itemized sentences in which words defined by the sub-module specifications are used.

Monitor specification modules part consists of a set of abstract monitor types and has the following syntax.

```
<monitor name> is abstract monitor type
  <monitor specification body>
  <sub concurrent modules>
  <sub functional modules>
  <monitor specification modules>
  <type specification modules>
  <lexicon declaration>
end
```

Monitor name is a common noun which represents the type of shared resource, like type name. Monitor specification body is the same as type specification body except that the former has state declaration and timing specification body mentioned above. In Fig.4.4, abstract monitor type buffer is defined.

4.5 Translation into temporal logic

In this section we explain the mechanism translating specification sentences into temporal logical formulas. Specification are translated using translations of words occurring in them. This translation method is based on that of Montague grammar.(9) However, our method has some differences from Montague's one as follows.

1) The types of the logical expressions into which words are translated are not uniquely decided from syntactic categories of the words. Generally, the type of logical expression is decided by data type or functionality of operation associated with a word. However, syntactic categories have fixed form of type schemata, so that the logical type depending on type or functionality of the translation of words which are elements of a syntactic category is specified as generic type α in the semantic rules.

2) Prepositions are introduced syncategorematically, in brief they have no translation, but control for matching with formal parameters and actual parameters of the module.

3) Every sentence in natural language is always translated into only one logical formula i.e. there is no ambiguity.

The logic we use is a first order many sorted temporal logic with temporal operators $\Diamond$, $\Box$, $\circ$ and <u>until</u>. In our model of the temporal logic, we use a linearly ordered set of time. Intuitionally speaking, $\Diamond A$ is true, if there is a time when A is true in the future. $\Box A$ is true, if A is always true in the future. $\circ A$ is true, if A is true in the next time. A <u>until</u> B is true, if A is true until B becomes true, so that if $\Box \sim B$ then $\Box A$.

A set of English sentences available in our specifications is very restricted. A part of their syntactic rules and semantic rules (translation rules into logical expressions) are shown in Fig.4.5. Symbol α in those rules represents type of the logical expression corresponding to a data type in specifications. If there are more than two associated data types with a word, that is to say, the corresponding module has more than two arguments, they are bracketted with '[' and ']'. For instance, $\langle adjective\rangle_{[\alpha_1,\ldots,\alpha_n]}$ represents that the number of the arguments of the defined word is n and that a data type of each argument corresponds to a type α_i ($1 \le i \le n$) in logic. As in the case of Montague grammar, each syntactic category of the natural language is associated with unique type (in our case, type schema) of formal logic. Correspondence of syntactic categories to types and their instances in natural language which are actually used in our specifications are summarized in the following table in Fig.4.6.

Fig.4.7 shows the example translation of the first sentence in the top level specification of eight queens' problem shown in Fig.4.3 into a logical formula. In the translation of the English sentence, simple logical formulas are obtained by applying logical axioms and theorems, e.g. λ-reduction and global reductions. The numbers associated with nodes of the tree are rule numbers - listed in Fig.4.5 - which are used in the derivation of the sentence.

<sentence>::=<subject>$_\alpha$(P) <verb phrase>$_\alpha$(F)| :1
P(F)

<subject>$_\alpha$::=<term>$_\alpha$(P) :6
P

<term>$_\alpha$::=name$_\alpha$ | :7
$\lambda f[f(\text{name}'_\alpha)]$
<determinar>$_\alpha$(D) <noun clause>$_\alpha$(F)| :8
D(F)

<term2>$_\alpha$::=<determinar2>$_\alpha$(D) <noun clause>$_\alpha$(F) :44
D(F)

<verb phrase>$_\alpha$::=<be verb>$_\alpha$(W) <complement>$_\alpha$(P) :11
W(P)

<be verb>$_\alpha$::=**is** (**are**) | :12
$\lambda p \lambda x[p(\lambda y[x=y])]$

<complement>$_\alpha$::=<adjective phrase>$_{[\alpha]}$(F)| :14
$\lambda g \exists x[F(x) \wedge g(x)]$
<term>$_\alpha$(P)| :15
P

<adjective phrase>$_{[\alpha_1,\ldots,\alpha_n]}$::=adjective$_{[\alpha_1,\ldots,\alpha_n]}$| :22
$\lambda x_1 \ldots \lambda x_n[\text{adjective}'(x_1)\ldots(x_n)]$
<adjective phrase>$_{[\alpha_1,\ldots,\alpha_{n+1},\ldots,\alpha_n]}$(U) <preposition> <term>$_{\alpha_{n+1}}$(Q)| :23
$\lambda x_1 \ldots \lambda x_n \exists x_{n+1}[U(x_1)..(x_{n+1})..(x_n) \wedge Q(\lambda z[z=x_{n+1}])]$
<adjective phrase>$_{[\alpha_1,\ldots,\alpha_{n+1},\ldots,\alpha_n]}$(U) <preposition> <term2>$_{\alpha_{n+1}}$(Q)| :45
$\lambda x_1 \ldots \lambda x_n \exists x_{n+1}[U(x_1)..(x_{n+1})..(x_n) \wedge Q(\lambda y_1 \lambda y_{n+1}[y_1=x_1 \wedge y_{n+1}=x_{n+1}])]$

<noun phrase>$_{[\alpha_1,\ldots,\alpha_n]}$::=common noun$_{[\alpha_1,\ldots,\alpha_n]}$| :24
$\lambda x_1 \ldots \lambda x_n[\text{common noun}'(x_1)\ldots(x_n)]$

Fig.4.5 A part of syntax rules and semantics

$\langle$noun clause$\rangle_{\alpha}$ ::= $\langle$noun phrase$\rangle_{[\alpha]}(F)$ | :2
y

$\langle$determinar$\rangle_{\alpha}$ ::= a (the) | :3
$\lambda f \lambda g \exists x[f(x) \wedge g(x)]$
every :3
$\lambda f \lambda g \forall x[f(x) \rightarrow g(x)]$ (positive)
$\lambda f \lambda g \exists x[f(x) \wedge g(x)]$ (negative)
no :4
$\lambda f \lambda g \neg \exists x[f(x) \wedge g(x)]$

$\langle$determinar2$\rangle_{\alpha}$ ::= any other :4'
$\lambda f \lambda b \exists x \forall y[x \neq y \wedge f(y) \rightarrow b(x)(y)]$ (positive)
$\lambda f \lambda b \exists x \exists y[x \neq y \wedge f(y) \wedge b(x)(y)]$ (negative)

Types of the variables and meta-variables in the above semantic rules are as follows.

$x, y: \alpha$, $z: \alpha_{n+1}$, $x_i, y_i: \alpha_i$, $F, f, g: \langle\alpha\ t\rangle$,
$Q: \langle\langle\alpha_{n+1}\ t\rangle\ t\rangle$, $p, P: \langle\langle\alpha\ t\rangle\ t\rangle$,
$W: \langle\langle\langle\alpha\ t\rangle\ t\rangle\ \langle\alpha\ t\rangle\rangle$, $U: \langle\alpha_1 \langle\ldots\langle\alpha_{n+1}\langle\ldots\langle\alpha_n\ t\rangle\ldots\rangle$
$b: \langle\alpha\ \langle\alpha\ t\rangle\rangle$, $D: \langle\langle\alpha\ t\rangle\ \langle\langle\alpha\ t\rangle\ t\rangle\rangle$.

Fig.4.5 A part of syntax rules and semantic rules(continued)

Syntactic category	Type	Instance
$\langle$sentence$\rangle$	t	no queen is checking against any other queen in X
$\langle$term$\rangle_{\alpha}$	$\langle\langle\alpha\ t\rangle\ t\rangle$	no queen, any other queen, X
$\langle$be-verb$\rangle_{\alpha}$	$\langle\langle\langle\alpha\ t\rangle\ t\rangle\ \langle\alpha\ t\rangle\rangle$	is , is not
$\langle$determinar$\rangle_{\alpha}$	$\langle\langle\alpha\ t\rangle\ \langle\langle\alpha\ t\rangle\ t\rangle\rangle$	a , every , no , the
$\langle$determinar2$\rangle_{\alpha}$	$\langle\langle\alpha\ t\rangle\ \langle\langle\alpha\ \langle\alpha\ t\rangle\ t\rangle\rangle\rangle$	any other
$\langle$noun clause$\rangle_{\alpha}$	$\langle\alpha\ t\rangle$	queen , X-coordinate , Y-coordinate
$\langle$adjective phrase$\rangle_{[\alpha_1,\ldots,\alpha_n]}$	$\langle\alpha_1\ \langle\ldots\langle\alpha_n\ t\rangle\ldots\rangle$	checking (n=3) , placed (n=2)
$\langle$adjective phrase$\rangle_{[\alpha]}$	$\langle\alpha\ t\rangle$	checking against any other queen in X
$\langle$connective$\rangle$	$\langle t\ \langle t\ t\rangle\rangle$	and , or

Fig.4.6 Relations of syntactic categories, types and their instances

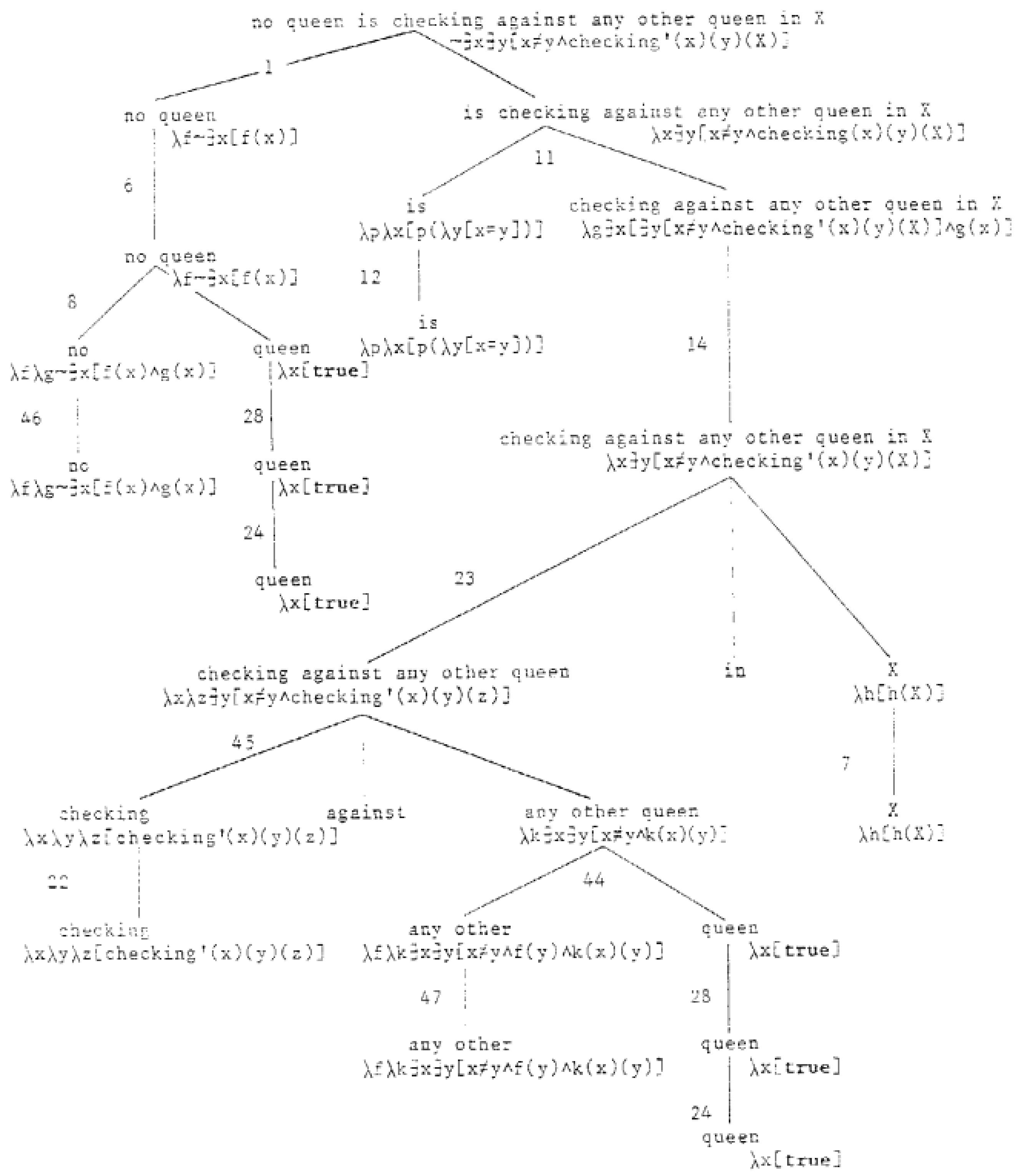
no queen is checking against any other queen in X
¬∃x∃y[x≠y∧checking'(x)(y)(X)]
1
no queen
λf¬∃x[f(x)]
6
no queen
λf¬∃x[f(x)]
8
no
λfλg¬∃x[f(x)∧g(x)]
46
no
λfλg¬∃x[f(x)∧g(x)]
queen
λx[true]
28
queen
λx[true]
24
queen
λx[true]
is checking against any other queen in X
λx∃y[x≠y∧checking(x)(y)(X)]
11
is
λpλx[p(λy[x=y])]
12
is
λpλx[p(λy[x=y])]
checking against any other queen in X
λg∃x[∃y[x≠y∧checking'(x)(y)(X)]∧g(x)]
14
checking against any other queen in X
λx∃y[x≠y∧checking'(x)(y)(X)]
23
checking against any other queen
λxλz∃y[x≠y∧checking'(x)(y)(z)]
in
X
λh[h(X)]
7
X
λh[h(X)]
45
checking
λxλyλz[checking'(x)(y)(z)]
22
checking
λxλyλz[checking'(x)(y)(z)]
against
any other queen
λk∃x∃y[x≠y∧k(x)(y)]
44
any other
λfλk∃x∃y[x≠y∧f(y)∧k(x)(y)]
47
any other
λfλk∃x∃y[x≠y∧f(y)∧k(x)(y)]
queen
λx[true]
28
queen
λx[true]
24
queen
λx[true]

Fig.4.7 Translation of a specification

Similarly, as to the specification of the producer-consumer problem in Fig.4.4, we have a following set of translations of 'produce' module specification. They contain temporal operators unlike the example of eight queens' problem.

$({}_i\text{produce})$

$\quad \text{begin}({}_i\text{produce}) \rightarrow \text{begin}({}_i\text{generate})$

$\quad \text{finish}({}_i\text{generate}) \wedge {}_i\text{generate-arg1}=x$

$\qquad\qquad \rightarrow \Diamond(\text{begin}({}_i\text{write}) \wedge {}_i\text{write-arg2}=x)$

$\quad \text{finish}({}_i\text{write}) \rightarrow \Diamond\text{finish}({}_i\text{produce})$

$\quad ({}_i\text{generate})$

$\qquad \text{begin}({}_i\text{generate}) \rightarrow \Diamond\text{finish}({}_i\text{generate})$

Fig.4.8 Translation of the 'producer' module

In Fig.4.8, it should be noted that for each $1 \leq i \leq n$, there is the set of translations i.e. those formulas with index i are the schemas of formulas. State predicate 'begin' and 'finish' represents the execution state of the processes whose name are their arguments.

Fig.4.9 shows the translations of specification of abstract monitor type 'buffer'. In the figure, state predicate 'at', 'in' and 'after' are declared in the state predicate part. The first two sentences are generated from the description of the functional specification section.

(Abstract monitor type buffer)

$$\mathrm{in}({}_{i}\mathrm{write}) \wedge \mathrm{buffer}=x \wedge {}_{i}\mathrm{write\text{-}arg2}=y \rightarrow$$
$$\Diamond\circ[\mathrm{after}({}_{i}\mathrm{write}) \wedge \mathrm{buffer}={}_{i}\mathrm{write}(x,y)\ \underline{\mathrm{until}}\ \mathrm{Dsj}\{\mathrm{in}(\mathrm{ops})\}\ ]$$

$$\mathrm{in}({}_{i}\mathrm{read}) \wedge \mathrm{buffer}=x \rightarrow$$
$$\Diamond\circ[\mathrm{after}({}_{i}\mathrm{read}) \wedge \mathrm{buffer}=\mathrm{arg1}({}_{i}\mathrm{read}(x))\ \underline{\mathrm{until}}\ \mathrm{Dsj}\{\mathrm{in}(\mathrm{ops})\}$$
$$\wedge\ {}_{i}\mathrm{read\text{-}arg2}=\mathrm{arg2}({}_{i}\mathrm{read}(x))\ \underline{\mathrm{until}}\ \mathrm{in}({}_{i}\mathrm{read})\ ]$$

$$\mathrm{at}({}_{i}\mathrm{write}) \wedge {}_{i}\mathrm{write\text{-}arg2}=x \wedge \mathrm{buffer}\neq\emptyset \rightarrow$$
$$(\mathrm{at}({}_{i}\mathrm{write}) \wedge {}_{i}\mathrm{write\text{-}arg2}=x)\ \underline{\mathrm{until}}\ \mathrm{buffer}=\emptyset$$

$$\mathrm{buffer}=\emptyset \wedge \exists i[\mathrm{at}({}_{i}\mathrm{write})] \rightarrow$$
$$\exists S[\ \exists Q[\forall i(\mathrm{element}(i,Q) \leftrightarrow \mathrm{at}({}_{i}\mathrm{write})) \wedge \mathrm{subset}(S,Q)] \wedge$$
$$\forall j(\mathrm{element}(j,S) \leftrightarrow \mathrm{in}({}_{j}\mathrm{write})) \wedge \mathrm{number}(S)=1\ ]$$

$$\mathrm{at}({}_{i}\mathrm{read}) \wedge \mathrm{buffer}=\emptyset \rightarrow \mathrm{at}({}_{i}\mathrm{read})\ \underline{\mathrm{until}}\ \mathrm{buffer}\neq\emptyset$$

$$\mathrm{buffer}\neq\emptyset \wedge \exists i[\mathrm{at}({}_{i}\mathrm{read})] \rightarrow$$
$$\exists S[\ \exists Q[\forall i(\mathrm{element}(i,Q) \leftrightarrow \mathrm{at}({}_{i}\mathrm{read})) \wedge \mathrm{subset}(S,Q)] \wedge$$
$$\forall j(\mathrm{element}(j,S) \leftrightarrow \mathrm{in}({}_{j}\mathrm{read})) \wedge \mathrm{number}(S)=1\ ]$$

Fig.4.9 Translation of 'buffer' module

Although the sentences described in the section do not take notion of states into account, we can generally generate formulas in temporal logic which describe state changes from such statements. The set of values of an abstract monitor type is considered to be a set of states. The constructors which are functions whose range type contains the abstract monitor type changes states. For example, in the case of the abstract monitor type 'buffer', ${}_{i}$read, ${}_{j}$write are such constructors.

The sentences in temporal logic are generated automatically from a input-output relation description in functional specification section, in such a way that a constructor name, its argument list, and abstract monitor type name are substituted in a schema of temporal logic. For instance, if a constructor's functionality is Abstract monitor type × In-type$_1$ × ... × In-type$_n$ → Abstract monitor type × Out-type$_1$ × ... × Out-type$_m$, then the schema is of the form :

$$\mathrm{in}({}_{i}\underline{\mathrm{op}}) \wedge \underline{\mathrm{Amt}} = x_0 \wedge {}_{i}\underline{\mathrm{op}}\mathrm{inarg2} = x_1 \wedge \ldots \wedge {}_{i}\underline{\mathrm{op}}\mathrm{inargn} = x_{n-1}$$
$$\rightarrow\Diamond\circ[(\mathrm{after}({}_{i}\underline{\mathrm{op}}) \wedge \underline{\mathrm{Amt}} = \mathrm{arg1}({}_{i}\underline{\mathrm{op}}(x_0,\ldots,x_n))$$
$$\wedge\ {}_{i}\underline{\mathrm{op}}\mathrm{outarg1} = \mathrm{arg2}({}_{i}\underline{\mathrm{op}}(x_0,\ldots,x_n))$$
$$\wedge\ldots\wedge{}_{i}\underline{\mathrm{op}}\mathrm{outargm} = \mathrm{argm{+}1}({}_{i}\underline{\mathrm{op}}(x_0,\ldots,x_n)))\ \mathrm{until}\ \mathrm{Dsj}\{\mathrm{in}(\underline{\mathrm{ops}})\}].$$

The instance of the schema for a constructor is generated by replacing underlined parts of the schema, that is op is replaced by the constructor name and Amt is replaced by the abstract monitor type name and Dsj{in(ops)} is the disjunction of in(op1),...,in(opn), where opi ($1 \leq i \leq n$) are constructor names other than the considering constructor. If the constructor has some outputs except abstract monitor type, then the sentence indicating that the value is held until the execution of next state change is added. The second sentence is such a case.

Next four formulas are generated in the same way as the case of the eight queens problem by the translation rules from the specification body of the timing section of the abstract monitor type.

From these generated formulas in temporal logic, we can prove the various dynamic properties of the system such as mutual exclusion and reachability.

(18)~(24)

It should be noted that the method translating static input-output property description into the formulas which express dynamic properties of the operator is properly new idea and it provides the minimality of specification.

4.6 Tell project

We have introduced specification technique based on lexical decomposition which provides natural way of module design, readability and capability of semantical processing by machine such as verification, document generation.

Another advantage of our approach we should mention here is that there is a possibility to translate the specification into another natural language by giving correspondence of words of both languages and syntactic translation rules, since very simple semantics are exploited.

Executable language and its semantics by temporal logic with a method of verification of total correctness are presented in (24). They are very naturally connected in the sense that the executable language is designed to be functional style language, so that the program can be constructed as a function according to the modules of the specification and correctness of a program can be proved in this module hierarchy.

From the experiences of writing various specifications such as communication protocols and text editor, we feel this specification method is very natural and comprehensive, and found out that more sophisticated environment or tools which support writing specification are desired.

We are now developing software development system 'Tell' (Tokyo Institute of Technology Elaboration Language), into which such tools as mentioned above, e.g. structured editor, interactive verifier, translator of distinguished natural languages, and so on, are integrated. Fig.4.10 shows the overvew of Tell environment. Specification database is used for specification designers to store the definitions of words which are often used. Therefore, it is possible to construct specifications hiding common knowledges which are stored in the database. The database can be considered as knowledge base of the user.

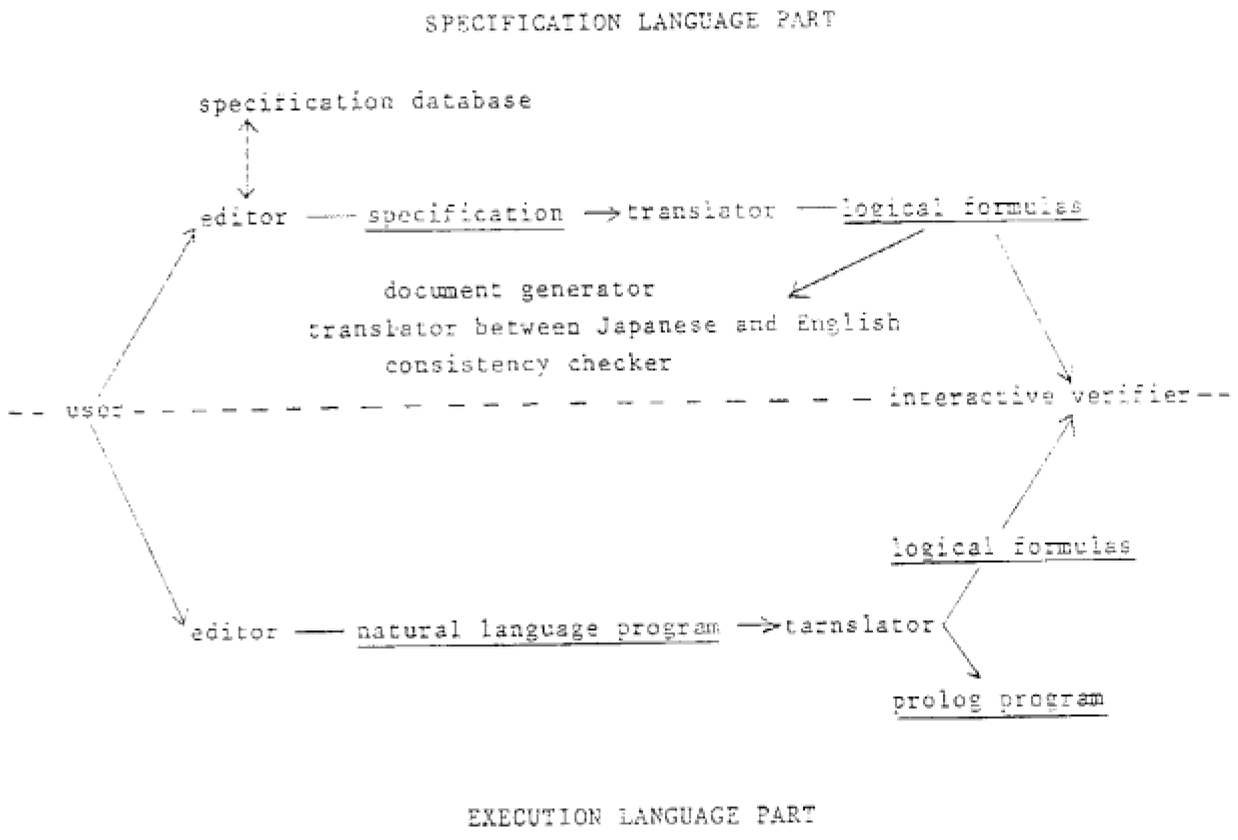

Fig.4.10 Overview of Tell Project

We have implemented a prototype of translator into logical formulas and of stuructured editer in English version. One of the future researches is that the remaider of the tools will be implemented. To integrate and minimize the specifications, it may be needed to exploit more expressive logic which can treat context information (25) or to exploit non-monotonic logic (26). Synthesis of a program from a specification in this style must be built and tested in a real environment. This is also one major direction of future research.

4.7 Execution language - concept of mission and meta inference rule

The Specifications of functional modules are described as a conjuntion of static Is-a relations mainly without any additional information, so that it is necessary to enumerate and check universe of whole structure when we need results of functional modules. In this case definition of configuration and database are requested to give efficient computations. Configuration and databases express states of computation and are dynamically updated as a computation advances.

Mission consist of three definitions, a way of computations represented by the specification, beginning condition, terminating condition of computation.

In general, configuration is related to a way of doing work and it is a structure composed of pairs of symbol name and its attribute values.

Configuration may have an order of updating its value. We consider eight queens' problem as an example. We assume that a queen has just been placed on the position I row J column, where $1 \le I \le 8$ and $1 \le J \le 8$. At this time, the configuration, which represents the arrangement of queens on a chessboard, satisfies a condition that I queens are placed at i-th row ($1 \le i \le I$) and that no queen checks against any other queen. We call it I sub-arrangement. It should be noted that the specification in Fig.4.3 can be seen as the previous condition by replacing 'Eight' with 'I'. The way to solve eight queens' problem is as follows. If I sub-arrangement is computed, I+1 sub-arrangement should be computed. The mission try to put a queen on I+1 row in such a way that the queen is moved along from column 1 to column 8, skipping all squares checked by the preceding queens. If the trial suceeds, new configuration becomes I+1 sub-arrangement. If the trial fails, the mission rearrange the queen in row I column J in such a way that the queen is moved along from column J to column 8, skipping all checked squares. On failing in rearranging queen on I, backtracking to I-1 is needed.

Above description means that configuration has a control structure such as nesting and backtracking on some index structure.

Fig.4.11 shows an example of the mission solving eight queens' problem.

```
 Mission solving the arrangement X as the eight queens' solution
  is the system of
  Configuration
     1)  There are a set of chessboard's squares having
         x-coordinate index I: {(I,J) | index I=1..i}.
     2)  Current position, (I,J).
  Order
     1)  j≤8 -> new position is i+1.
     2)  j>8 -> new position is i-1.
  Timing
     1)  Initially, Configuration is {(1,1)}.
     2)  Mission selects new position satisfying arrangement
         in the order and constructs configuration by inserting it.
     3)  Finally, Configuration is {(I,J) | index I=1..8, index J=1..8}.
end
```

Fig.4.11 The mission of eight queen's problem

Mission includes a description of dynamic behaviors of configuration which are used during computation. In order to define an efficient computation method, further descriptions for configuration are necessary. They are order and timing specifications of updating configuration, and have the same style as the mission specification. Order specification describes controls of indices. Timing section includes initial condition, fundamental commands which the mission issues, and final condition for stopping mission. Fundamental commands appears as verbs for updating values of the configuration. For example, the mission of Huffman coding tree shown in Fig.4.13 includes the invocation of select command whose execution results in the relation between lengths of paths to nodes shown in Timing 2). This fact is associated with the relation shown in 2) of the specification of in Fig.4.12.

Huffman tree is an optimum binary tree computed from a set of events whose probabilities are specified. In Fig.4.12, input Sleaf is a set of probabilities of events' occurrence. Condition 2) in the specification describes that the value of each node in Huffman tree is the sum of the value of left node and that of right one. 3) shows that the smaller the value of the node is, the longer the path from the root to it is. The way how to explore Huffman tree is as follows. Variable Wsettree is a set of binary trees and its initial value is Sleaf. First, such two binary trees S_x, S_y as values of their root nodes are the smallest in Wsettree are selected. A binary tree is constructed from them. The value of the root node of constructed tree is the sum of the value of selected two nodes. Selected trees are removed from Wsettree. This step is repeated until Wsettree contains only a binary tree, which is Huffman tree.

```
Binarytree of real t is Huffman tree of set of real Sleaf
    means that
       1) Sleaf is leaves of t .
       2) For every x such that x is node of t and x is not a leaf,
                 /value[x]=value[ltree[x]]+value[rtree[x]]/ .

       3) For every x,y such that x is a node of t
                              and y is a node of t
                              and /value[x]≤value[y]/ ,
                  /length[y,t]≤length[x,t]/ .

end  Huffman tree

Binarytree of α is type
     represented by
             (isleaf: (value: α) |
             (value: α, ltree: binarytree of α,
                        rtree: binarytree of α)) .
     functions
         t is a left subtree of X := ltree[X]
         t is a right subtree of X := rtree[X]
         v is a value of X := value[X]
         The length of α in binarytree of α X is string.
     predicates
         Binarytree of α X is a leaf.
         α p is a node of binarytree of α X.
     satisfy
         1) p is a value of X → p is a node of X .
         2) X is a leaf and p is not a value of Y
                → p is not a node of X .
         3) X is not a leaf and p is not a value of X
                → /node[p,X] = node[p,ltree[X]] ∨ node[p,rtree[X]]/ .
         4) p is a value of X → /length[p,X]=0/ .
         5) p is a node of X and p is a not a value of X →
                 5-1) p is a node of a left subtree of X
                     → /length[p,X]=L∘length[p,ltree[X]]/ .
                 5-2) p is a node of a right subtree of X
                     → /length[p,X]=R∘length[p,rtree[X]]/ .
end  binarytree
```

Fig.4.12 Specification of Huffman tree

```
Mission solving binarytree of real t as Huffman tree
    is the system of
    Configuration
        1) There is current Wsettree which is a set of subtree $S_I$
           satisfying Huffman tree
           having index I:{$S_I$ | index I=1..i}.
        2) Current number of index, i.
    Order
        1) New two subtrees is ($S_x$,$S_y$).
        2) New number of index is i-1.
    Timing
        1)  Initially, configuration is Sleaf.
        2)  Mission selects new two subtrees ($S_x$,$S_y$),
            constructs a subtree by 2), and inserts it to configuration
                -> next number of index is i-1,
           the lengths of ($S_x$,$S_y$) > length of any subtree in Wsettree.
        3)  Finally, number of index i=1.

    Note : meta inference mechanism proves that 3)
          Timing
                -> values of roots $S_x$,$S_y$ $\le$ value of root of any subtree
                     in new Wsettree.
                -> generate new  mission  seeking two subtree
                     having minimum values of their root in current Wsettree.
end
```

Fig.4.13   The mission of Huffman coding tree

Meta inference mechanisms are an useful tool for verifying the fact and use a set of rules called world as object which can be unified with variable. Knowledge based software system should have specifications of various modules and generate procedures by using interactive communication tool with user specifying various meta inference rules. These knowledge bases can act an excellent abilities of new software generation.

Meta inference rule can be categorized as follows:

1.Translation from implicit descriptions to explicit descriptions -- upward and downward abstractions and abbreviations are supplemented

2.Unification for meta variables

3.Inference of results obtained by commands in mission level

4.Inference of efficient execution sequence for non deterministic statements

5.Inference of cooperation of control timing in the mission level

Acknowledgement

We wish to thank Dr. A. Kuremastu, Mr. K. Chiba, and Mr. T. Takizuka of

KDD(International Telephone and Telegram Japan) Inc. for insightful comments and numerous discussions.

[REFERENCES]

(1) Parnas,D.L. : On the Criteria to Be Used in Decomposing Systems into Modules, Commun. ACM, 15, 12 (Dec.1972) pp.1053-1058.

(2) Mealy,G.H. : Another Look at Data, Proc. of AFIPS, FJCC Vol31, AFIPS Press, Montrale, N.J., (1967), pp525-534.

(3) Liskov,B.H. : A Design Methodology for Reliable Software Systems, FJCC, (1972), pp191-199.

(4) Jackson,M.A. : Principles of Program Design, Academic Press (1975)

(5) Myers,G.J. : Reliable Software through Composite Design, Mason/Charter Pub. (1975).

(6) Liskov,B.H. and Zilles,S.N. : Specification Techniques for Data Abstractions, IEEE SE-1, 1, (Mar. 1975) pp.7-19.

(7) Dijkstra,E.W. : Notes on Structured Programming, Structured Programming, Academic Press, (1972), pp.1-pp.82

(8) Wirth,N. : Program Development by Stepwise Refinement, Commun.ACM, Vol.14, No.4, (1971), pp.221-227

(9) Montague,R. : The Proper Treatment of Qualification in Ordinary English, Approaches to Natural Language, Reidel Dordrecht (1973).

(10) Guttag,J.V., Horowitz,E., and Musser,D.R. : Abstract Data Types and Software Validation, Commun.ACM, Vol.21, No.12 (1978) pp.1048-1064.

(11) Goguen,J.A. : An Initial Algebra Approach to the Specification of Reliable Software, Boston M.A. (1979) pp.162-169.

(12) Hoare,C.A.R. : Notes on Data Structuring, Structured Programming, Academic Press, (1972), pp.83-174

(13) Gallin,D. : Intensional and Higher Order Modal Logic; With Applications to Montague Semantics, North-Holland Publishing, Amsterdam (1975).

(14) Teichroew,D. and Herskey III,E.A. : PSL/PSA:A Computer-aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE Trans. Software Engineering, Vol. SE-3, No.1 (1977) pp.41-48.

(15) Bell,T.E., Bixer,D.C., and Dyer,M.E. : An Executable Approach to Computer-aided Software Requirements Engineering, IEEE Trans. Software Engineering, Vol. SE-3, No.1 (1977) pp.49-60

(16) Saeki,M., Yonezaki,N., and Enomoto,H. : Formal Specification Method Based on Lexical Decomposition of Natural Language, Transactions of Information Processing Society of Japan, to be appeared, (in Japanese)

(17) Nakajima,R., Honda,E., and Nakahara,H. : Hierarchical Program Specification and Verification -- a Many-sorted Logical Approach, Acta Informatica, Vol.14, Fasc.2 (1980) pp.135-155.

(18) Owicki,S. : Axiomatic Proof Techniques for Parallel Programs, Ph. D. Th., Cornell University, (Aug. 1975).

(19) Pnueli,A. : The Temporal Logic of Programs. 18th Annual Symposium on Foundations of Computer Science. (Nov. 1977) pp.46~57.

(20) Manna,Z. and Pnueli,A. : The Modal Logic of Programs, 6th International Colloquim on Automata, Language and Programming, Graz, Austria, Lecture Notes in Computer Science, Vol. 71, Springer Verlag (July 1979) pp.386~408.

(21) Hailpern,B. : Verifying Concurrent Processes Using Temporal Logic. Technical Report 195. Computer Systems Laboratory, Stanford University. (Aug. 1980).
(22) Keller,R.M. : Formal Verification of Parallel Programs, Commun. ACM, 19, 7 (1976).
(23) Lamport,.L : Proving the Correctness of Multiprocess Programs, IEEE Transaction on Software Engineering 3, 2 (1977) pp.125~143.
(24) Yonezaki,N. and Katayama,T. : Functional Specification of Synchronized Processes Based on Modal Logic, Proc. of 6th ICSE (Sept. 1982) pp.208-217.
(25) Hausser,R. and Zaefferer,D. : Question and Answering in a Context-dependent Montague Grammar, Formal Semantics and Pragmatics for Natural Languages (Guenthner,F. and Schmidt,S.J. Ed.) , Reidel Dordrecht (1978) pp.339-358.
(26) McDermott,D. and Doyle,J. : Non-Monotonic Logic 1, Artificial Intelligence 13 (1980) pp.41-72.

5. Meta Inference and Its Application in Fifth Generation Kernel Language

5.1 Introduction

Fifth Generation Kernel Language (FGKL) is the name of a family of languages which are to be used for software development in Japanese Fifth Generation Computer System (FGCS) project.(1) Its first member called KL0 (Kernel Language version 0) (2) is the machine language of the Personal Sequential Inference machine (PSI), currently being developed at ICOT Research Center. Its second member called KL1 (Kernel Language version 1) is the machine language of the Parallel Inference Machine (PIM), scheduled to be developed during the intermediate stage of the FGCS project. KL1 will be an extended version of KL0. The language features and concepts incorporated in KL1 are concurrency, set abstraction, and meta inference, and so on. This chapter presents conceptual specifications of the meta inference mechanism in KL1 and its applications.

Meta inference as discussed here means 'inference using meta knowledge'. Meta knowledge means 'knowledge concerning how to use and/or meta knowledge'. As object knowledge, we will treat only Horn logic axiom sets that can be expressed in KL1. To realize meta inference in KL1 requires embedding in KL1's predicates to express and utilize knowledge concerning how to use the KL1 interpreter/compiler. Specifically, it is necessary to introduce meta predicates, called demo and simulate, which reflect the process of solving goals in KL1.

Sophisticated meta inference functions including world's updation can be easily realized by expressing meta and object worlds in a single language and thereby amalgamating them. This was suggested by Bowen et al. (3) as the conceptual usefulness of the demo predicate.

We have developed two methods for implementing the demo predicate in DEC-10 prolog. The first method (4) was based strictly on the idea presented in (3) and the second (6)~(8) was developed by the hint of the 'Prolog Interpreter in Prolog'. (5) As a result of comparing the two methods, we currently use the second method because it permits easy implementation of diverse concepts and offers a high speed execution of the demo predicate.

5.2 Basic concept of meta inference

The simulate and demo predicates discussed in this chapter are basically divided into three types, respectively. We will first define the type III simulate predicate, which serves as the core, and then introduce two types of simulate predicates and two types of demo predicates as specialized meta predicates of the type III.

(Type III) simulate (World, NewWorld, Goal, Result, Control, Channel)

Meaning: The process of solving a given goal (Goal) under given controls (Control) while exchanging messages between one world (World) and another worlds through the communication logical variables (Channel) is simulated. As a side-effect of the simulation, the old world (World) is updated to a new world (NewWorld), At the same time, the necessary meta information (Result) is derived from the goal proving process itself.

(Type III') demo (World, Goal, Result, Control)

Meaning: The process of solving a given goal (Goal) under given controls (Control) in a world (World) is demonstrated. At the same time, the necessary meta information (Result) is derived from the goal proving process itself.

The types II and I simulate predicates and the types II' and I' demo predicates are defined as specializations of the type III simulate predicate and the type III' demo predicate, respectively, and their syntax is described as follows. Their meanings are evident from the meanings of the types III and III' meta predicates.

(Type II) simulate (World, NewWorld, Goal, Result, Channel)
(Type II') demo (World, Goal, Result)
(Type I) simulate (World, NewWorld, Goal, Channel)
(Type I') demo (World, Goal)

Types I and I' provide a multi-world World model and types II and II', a meta information Result derivation model in addition to the World model.

5.3 Multi-world identification type

5.3.1 Multi-world identification

Basically, there are two methods for representing the structure of the universe composed of meta knowledge and object knowledge as follows:

(a) implicit (static) representation

This method statically presents the structure of the universe as knowledge, such as part-of relations or is-a relations. Suited for knowledge representation languages such as Smalltalk and KRL, it is used where the structure of the universe is known.

Specific examples of this method are inheritance-demo in Mandala. (10) As will be seen in these examples, a nested world can be realized simply by calling another world from one of specific worlds using the simulate/demo predicates.

(b) explicit (dynamic) representation

This method explicitly presents the structure as program arguments. For example, simulate/demo arguments are given as simulate(W1+W2+W3,...). Suited for views of dynamically changing worlds, it is used where the structure of the universe is unknown.

Specific examples of this method are the 'with' predicate in Prolog/KR (11) and the 'with' predicate used by Takeda. (12) Since this method can be used in flexible and diverse ways depending on how world-structuring operators, e.g., '+', are processed, it needs to be considered more formally in the future.

5.3.2 Application of multi-world type

○ Implementation of parallel worlds (logically independent multiple worlds) which can communicate with each other.
○ External I/O can be considered within this framework.
○ Stream type set expressions in PARLOG can also be implemented by using the external communication function of simulate.
○ Utilization of multi-world knowledge base
○ Application to distributed knowledge base

5.4 Meta Information Derivation Type

5.4.1 Meta information derivation

The simulate/demo predicates can execute the process of solving in a given goal, derive the necessary information from the process of generating the proof tree, and reflect the result in the argument Result. The following types of Result are often utilized.

(i) Result which returns the result of proof (by 'fail' analysis)
(1) Identify true, false, and error.
(2) Identify true, false, unknown, and error.

These can be implemented by managing a Positive World and a Negative world.

(ii) Result which returns the proving process
(1) Return a success pattern. (Easy to implement.)
(2) Return a fail pattern. (Difficult to implement.)
(3) Return the entire proof tree.

What is important to the meta predicate simulate is to utilize intermediate results of endless process computations.

(iii) Result which derives and returns illogical concepts
(1) Derive Result by communicating information to another subworld in the same world.
(2) Derive Result by communicating information to another (meta/object) world.

Various illogical but useful concepts can be implemented by communicating with logically unrelated worlds.

5.4.2 Applications of meta information derivation type

Type II has a wide range of applications. Some of them are given below according to the classification shown in 5.4.1.
(i) Debugger and knowledge acquisition including exception values (9)
(ii) Explanation/tracing function, editor (12), scheme/meta class management (10), and diagnosis
(iii) (1) Multiple inheritance (10) and intelligent problem solving

(2) Communications with external (physical) I/O and knowledge base management, and cooperative problem solving

5.5 Control Information Addition Type

5.5.1 Control information addition

Logic programming based on Horn logic originally had the philosophy 'Algorithm = Logic + Control'. (13)

It is therefore considered possible to add control information in a form separate from Logic. However, the existing DEC-10 Prolog or Concurrent Prolog bases its logic control too much on the control primitive cut '!' or commit '|'. This means that in certain applications the user must provide some awkward, unnatural control. We have therefore chosen to add control information as parameters to control the Control information part of type III (III') simulate (demo) predicate. Some useful control parameters are given below.

(i) Control to change goal evaluation methods
(1) lazy/eager evaluation
(2) deferred evaluation (14)
In addition, partial evaluation is effective, depending on the problem involved.

(ii) Control to change proof tree search methods
(1) limitation of the search space (8)
(2) specification of conditions for search termination (e.g. depth-bound, derivation number (8))
(3) depth-first/breadth-first search (4),(14)
(4) introduction of new control primitives
It is necessary to introduce this sort of strategy in solving practical/real-world problems

(iii) Control to change proving methods
(1) top-down/bottom-up inference
(2) sequential/parallel inference
This is necessary in interfacing with different types of machines or systems.

5.5.2 Applications of control information addition type

Type III also has a wide range of applications. Some of them are given below according to the classification in 5.5.1.

(i) Set abstraction, and interface between RDBN and SIM (4),(14)
(ii) Speedup of theorem proving search (8) and solution of puzzles and games
(iii) Bottom-up Parser and Compiler to Compiler

NOTE : This chapter has been written by S. Kunifuji.

[REFERENCES]

(1) ICOT(ed.) : Outline of Research and Development Plans for Fifth Generation Computer Systems, ICOT, May 1983.
(2) Chikayama, T., Yokota, M., Hattori, T. : Fifth Generation Kernel Language

Version-0, Proceedings of the Logic Programming Conference '83, Tokyo, May 1983.
(3) Bowen, K.A., Kowalski, R.A. : Amalgamating Language and Meta Language in Logic Programming, School of Computer and Information Sciences, University of Syracuse (1981).
(4) Kunifuji, S., Yokota, H., Kitakami, H., Miyachi, T., Furukawa, K. : How to Interface Logic Programming Language and Relational Data Base Management System, presented by CERT Workshop on 'Logic Base for Databases', CERT, DEC. 1982.
(5) Coelho, H., Cotta, J.C., and Pereira, L.M. : How to Solve It with PROLOG (2nd edition), Laborato'rio Nacional de Engenhaira Civil, Lisboa (1980).
(6) Miyachi, T., Kunifuji, S., Kitakami, H., Furukawa, K., Takeuchi, A., Yokota, H. : A Knowledge Assimilation Method for Logic Databases, ICOT TR-025, Aug. 1983.
(7) Kunifuji, S., Asou, M., Takeuchi,A., Sakai, K., Miyachi, T., Kitakami, H., Yokota, H., Yasukawa, H., Furukawa, K. : Amalgamation of Object Knowledge and Meta Knowledge in Prolog and its Applications, Information Processing Society of Japan, Preprints of WG on Knowledge Engineering and Artificial Intelligence, June 1983 (in Japanese).
(8) Kitakami, H., Kunifuji, S., Miyachi, T., Furukawa, K. : A Methodology for Implementation of a Knowledge Acquisition System, ICOT TM-0024, Aug. 1983.
(9) Kunifuji, S., Miyachi, T., Kitakami, H., Furukawa, K. : Knowledge Acquisition and Meta Inference, 27th National Conference of Information Processing Society of Japan, Nagoya University, Oct. 1983 (in Japanese).
(10) Furukawa, K., Takeuchi, A., Kunifuji, S. : Mandala: Knowledge Representation System in Concurrent Prolog, Information Processing Society of Japan, Preprints of WG on Knowledge Engineering and Artificial Intelligence, Nov. 1983 (in Japanese).
(11) Nakashima, H. : A Knowledge Representation System: Prolog/KR, Dissertation, METR83-5 of Mathematical Engineering, University of Tokyo, Feb. 1983.
(12) Takeda, M. : Design of Knowledge-based Language-oriented Editor, Proceeding of WG4 Workshop, ICOT, July 1983 (in Japanese).
(13) Kowalski, R.A. : Algorithm = Logic + Control, CACM, Aug. 1979.
(14) Yokota, H., Kunifuji, S., Shibayama, S., Miyazaki, N., Kakuta, T., Murakami, K. : An Enhanced Inference Mechanism for Generating Relational Algebra Queries, ICOT TR-026, Oct. 1983.

6. Conclusion

We discussed a framework of knowledge based software system, and introduced a part of 'Tell' project in Tokyo Institute of Technology and Fifth Generation Computer System project. Tell has a knowledge base facility, stepwise definition along implicit structure, and may be useful through software life cycle.

In general the sets of definitions declared in a static and itemized way such as relations and predicates do not supply an complete and efficient computing procedure directly. There are many implicit informations in order to obtain final computing results. Therefore it is important to make more clear how to organize the structure of information hiding (1), i.e., implicit structure. Making implicit for many things and giving only necessary specification suggest an importance of concept of service. Furthermore information should be classified from the view point of semantics.