

SYSP : A New Programming Language to the Next Generation

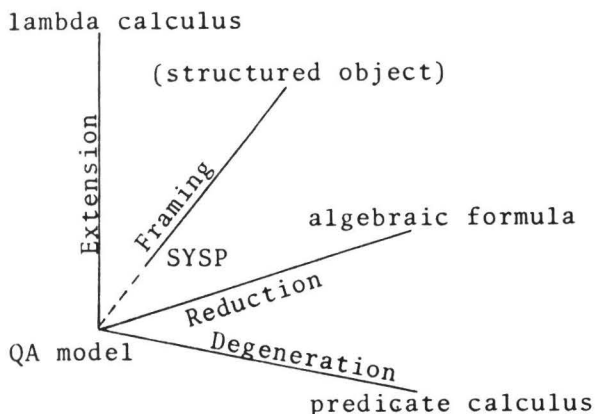
Toshio Yokoi, Shooichi Yokoyama, Taisuke Satoo, Fumio Motoyoshi
and Kazuhiro Fuchi

Machine Inference Section, Information Sciences Division
Electrotechnical Laboratory
2-6-1, Nagata-cho, Chiyoda-ku, Tokyo, Japan

Sysp(Structured Symbol System Processor) is a base programming language for knowledge representation, and also a non-procedural and non-von Neuman style programming language to the next generation. Logic programming, functional programming, object oriented programming etc. are integrated into a simple and transparent mechanism based on the message passing computation(QA model of computation). An overview of Sysp's framework is presented using simple examples.

1. INTRODUCTION

The research of knowledge representation demands a new type programming language for AI which is highly structured and is able to represent various aspects of knowledge naturally and uniformly. On the other hand, the one of automatic programming does new type programming(methodology and language) which is also highly structured and is able to make the process of programming more structured. These two trends aim at the same goal, for example, Sysp. It is a highly structured symbol manipulation system(in true sense, a general purpose computation system) based on Newell's physical symbol system hypothesis. Sysp consists of four co-ordinates.



In followings, Sysp is outlined along Extension, Degeneration and Reduction co-ordinates.

2. EXTENSION to QA model

Lambda calculus is a computation model which is able to express the higher order computation naturally. So, it is available so much for the basic mechanism and semantics description of various programming languages and is extended to make its ability more powerful for this purpose. Recursion, Landin's J-operator, Reynolds's escape expression, Steele & Sussman's tail transfer mechanism and continuation function were introduced. These extensions were arranged into message passing mechanism like Actor formalism etc.[1]. The distinctive features of the mechanism are as follows.

(a) By preserving execution environment on heap instead of stack, send and receive actions are able to express coroutine and parallel computation directly.

(b) Multiple-value return and naming without side effect become possible. The composition and decomposition of data structure are expressed in non-procedural manner.

(c) The history of computation is preserved completely and easily, so that the well fitted mechanism to the evaluation of nondeterministic program is gained.

(d) In co-operation with static binding mechanism, truly necessary execution environment for a computation is clearly specified.

Moreover, QA model of computation, puts its stress on well-formed communication(question answering) mechanism for the natural and rigid representation of a problem solving process(computation). So, appropriate synchronization mechanism etc. are included

inherently and the communication mechanism is made more structured.

One method of making the mechanism more structured is on message sent and received. In natural language, a sentence is classified into some types, declarative, interrogative, imperative or exclamative, so that a listener is able to understand the intention of a speaker so easily. And moreover, in interrogative sentence, it is expressed clearly what sort of answer is expected, yes-no or answer to interrogative(When... , Where... , How many... , What kinds of... etc.). Similarly, in QA model, various message types are provided. And a computation is represented as a natural "question-answering(QA) sequence".

The other method of making the communication mechanism more structured is on QA sequence. First, QA sequence is restricted to the following communication manners.

- (1) When a question is sent/received, the answer must be received/sent.
- (2) To send/receive the answer to the latest received/sent question.

With these restrictions, sequences starting with an action on a question and ending in an action on the corresponding answer are nested with one another. Such a sequence is called "well-structured QA sequence". The simplest well-structured QA sequence is to send/receive a question and then to receive/send the answer. These are called "atomic QA sequence".

QA sequence(computation) is reduced into some sub-QA sequence(subcomputation). Therefore, the description among sub-QA sequences gives the structure of the original QA sequence. This description is called "compound QA sequence". For example, factorial function in Lisp,

```
FACT:(LAMBDA (N) (IF (= N 0) 1
                     (* N (FACT (- N 1)))))
```

,is represented as QA sequence.

```
Fact:=>(0 ?) Then 1.=>
=>(n ?) Then (n 1 ?)=>=>m.
      then (m ?)= Fact= m.
      then (n m ?)=>*=>m.
      Then m.->
```

3. DEGENERATION to predicate calculus

It is possible to make two actions, to receive a question and to send the answer, degenerate into one action, to assert the fact. In the same

manner, two actions, to send a question and to receive the answer, are made degenerate into one, to interrogate the fact. Furthermore, two actions, to assert a fact and to interrogate on the fact, are made degenerate into one object, a predicate(atomic formula).

By regarding assert and interrogate actions as atomic QA sequences, compound QA sequence is made degenerate. It is called "degenerate (compound) QA sequence". The factorial function is transformed into a partially degenerate QA sequence.

```
Fact:<=>(0 1)
=<=>(n ?m3) because (n 1 ?m1)<=>- then
                    (m1 ?m2)<=>Fact then
                    (n m2 ?m3)<=>*
```

In this sequence, variables in the messages have some sort of indication(input or output) and the evaluation order are fixed. Next, this is transformed into a fully degenerate QA sequence.

```
<=>(Fact: 0 1)
=<=>(Fact: n m3) because (-: n 1 m1)<=> and
                    (Fact: m1 m2)<=> and
                    (*: n m2 m3)<=>*
```

In this one, the variables have no indication and the fixation of the evaluation order is taken away. In order to make the meaning of Degeneration clearer, the factorial function is represented in Horn set,

```
+(Fact: 0 1)
+(Fact: n m3),-(-: n 1 m1),-(Fact: m1 m2)
                    ,-(*: n m2 m3)
```

Assert action corresponds to positive clause and interrogate action does to negative clause. And further, pattern directed invocation of assert action by interrogate action corresponds to the application of Resolution rule.

In the current research on logic programming[2], it has tried to make clear the correspondence between computation structures and proof strategies without any explicit construct for control structure. By the following reasons, however, it needs to introduce appropriate procedural aspects into logic programming.

(1) Many computation sequences are condensed into one logical formula by abstracting certain aspects of the sequences, but only some of them are able to have actual meanings. That is, it is hard to say that it represents computation exactly.

(2) Proof mechanism like Resolution is powerful as basic computation mechanism, but very primitive. So, higher structures of computation can not be represented by proof strategy only.

It is so easy to introduce procedural elements for claptrops, but never desirable. Sysp makes it possible to define strict meanings of procedural elements by introducing the successive Degeneration process from QA model to predicale calculus.

4. REDUCTION to algebraic formula

Along Extension co-ordinate, lambda calculus, an applicative computation system, is extended to include imperative features preserving the advantages as it is. And again, QA model is reduced into algebraic formula, a typical applicative system. But this never means that Extension co-ordinate is traced conversely.

Send action is reduced into the composite function of receiver and message. Receive action is reduced into environment modifier function. Namely, a message is regarded as a functional form and a variable is regarded as a function which yields the value from environment as well as in GEDANKEN. Receive action is regarded as a functional form which tests whether or not the actual argument matches the message pattern and, if matched, modifies the environment by naming. Continuation(next) relation is represented as the composition of functions. In this way, a simple QA sequence is reduced mechanically into an applicative form. And further, Reduction process goes on. The factorial function is partially reduced to

```
Fact=(1=>(0 ?);
      m:=>m.*(n m ?):=>m.Fact.(m ?):=>m.
      --(n 1 ?):=>(n ?))
```

Further, this is fully reduced into

```
Fact=(1=>0.I;
      *(I Fact.--(I 1):))
```

through the transformation which simplifies environment modifier function and eliminates variables by substitution (introducing selector function). Thus, a QA sequence is reduced into an algebraic formula, its semantics is formalized as laws of algebra and it becomes possible to prove its various properties by formula manipulation, as Backus etc. point out. For comparison, the example is represented in Backus's FP[3],

```
Def !=eq.[id, 0] -> I; *[id, !.--[id, I]]
```

For each type of compound QA sequence, corresponding functional forms are provided. The applicative form obtained in this way is called "reduced QA sequence".

Functional programming is related to QA model through successive Reduction process. By this way, its weakpoints are covered up and its ability is extended as follows.

(1) The appropriate and natural interpretation of functional form are obtained.

(2) Advantages of other computation models, including history sensitivity, are added to in a very natural way.

5. CONCLUSION

The method for knowledge representation must be powerful, natural and formal. It must be powerful in order to represent complicated and higher order knowledge, natural in order to make it easy to use and understand, and formal in order to make possible for a computer to process represented knowledge mechanically. Sysp satisfies these three conditions sufficiently over a wide range by integrating four representative methods into one transparent mechanism. For example, factorial function is able to give natural looks in Sysp,

```
Fact:<=>(0 1)
=>(n ?) Then (--(n 1): ?)=>Fact=>m.
      Then *(m n):=>
```

At present the main part of Sysp is implemented on EPICS-Lisp. In other papers[4], Sysp is described in more details.

AKNOWLEDGEMENTS: The authors wish to express their gratitude for the encouragement from the head of their division, Dr. H. Nishino.

REFERENCES

- [1] Hewitt, C. : "Viewing Control Structures as Patterns of Passing Messages," Artificial Intelligence 8, pp.323-364 (1977)
- [2] van Emden, M. H. : "Programming with Resolution Logic," in E. W. Elcock(ed.), Machine Intelligence 8, pp.266-299 (1977)
- [3] Backus, J. : "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," CACM, Vol.21, No.8, pp.613-641 (Aug. 1978)
- [4] Yokoi, T. : "SYSP I. Basic Mechanism & II. Various Structures for Representation & III. Combining Various Structure," Bul. Electrotech. Lab., Vol.42, No.4 & 9 & 6 (Apr. & Sept. & June 1978) (in Japanese)