Logic Programming FGCS (DRAFT)

Kazuhiro Fuchi
Faculty of Engineering, University of Tokyo

1 Introduction

Although logic programming appeared in the 1970s, for a while it was scarcely considered in Japan, and it was treated as a minor topic in the international research community. Perhaps, this is because the idea germinated in (a corner of) Europe and it ran counter to notions held by the majority, which was then led by the research community in America. In those days (and now), it was American researchers who were leading the research in this area.
In the '80s, logic programming gradually came to gain some acceptance. As a result, the scientific journal "Journal of Logic Programming" was published and an international academic conference for logic programming research was founded. Even though still in the minority, some researchers had established a research domain. It had become an attractive area of inquiry for younger researchers in Japan, in spite of the older researchers' (unreasonable) rejection.
I think the necessity for logic programming has stimulated this research area. In addition, a big (coincident) event, the start up of the "Fifth Generation Computer Systems Project" (FGCS) in Japan, gave impetus to this area of investigation. (The FGCS project has played a role in contributing to this research field in its own way.)

2 Encounter with Logic Programming

First experience: I remember I first encountered something like logic programming around 1960. What I read was probably a paper in a journal published by ACM, in which the author spoke of trying to translate logical formula (axioms and inference rules) into a programming language.
I was interested in that paper and tried to implement the language. But the attempt resulted in complete failure. Only a partial implementation was needed to prove that the execution speed was too slow for practical use.
The primary reason might have been my poor skill in programming. But, certainly, the second cause had to be the computer I used. It was a machine with 2,000 words of

storage, a cpu rate of 0.005 Mips, and a paper tape device for external storage. I had designed, hard-wired, and adjusted the thing, myself.
I soon recognized that our computer technology itself needed advancing, and we would have to wait until we could obtain advanced computers before conducting researches on interesting subjects.

2.1 Around 1970

Situations seem to change in 10 year cycles. The major computer technology research entities shifted from research institutes and universities to industry. After deciding to remain working at a research institute, I became keenly absorbed in the question of what kind of research themes our (national) research institutes should consider in the fields of computer technology and information processing.
Conditions at my research institute had gradually improved to the point where it had become possible to afford the purchase of a large general purpose computer. Although my mind did not seem particularly suited for a philosophy-oriented or theory-oriented research approach, I began to feel it was the time for a forward thinking research project to be "experimentally" conducted, making use of a computer in a practical way. What 's more, I felt it was time for us to take positive steps in that direction.
For this reason, I began surveying the current state of research on information processing in the "advanced" countries, and began a review of their work. I found the state of technology to be further advanced than I had expected.
Research on artificial intelligence conducted at MIT, as well as Stanford University, SRI, and the University of Edinburgh had already given birth to AI-oriented programming languages and natural language understanding systems resulting in prototypes of artificial robots.
On the other hand, new programming methodologies had been proposed and mathematical theories offered for programs. Software engineering had been proposed and database research had come to be popular, bolstered by the proposition of relational database models.

2.2 Inference Mechanism Laboratory

A new research laboratory with the curious name (at that time) "Inference Mechanism Laboratory" was established at the research institute where I was working. Although it was a small laboratory, it became the base for research in this field.
The first task for the Inference Mechanism Laboratory was to implement a Lisp processor, since I believed this would be necessary as the infrastructure for new research. Dr. Toshiaki Kurokawa had already implemented one of the world's most advanced Lisp processors at the time, corresponding to MacLisp.
Dr. Hozumi Tanaka immediately began research on a Japanese language understanding system for the processor. The starting point was converting Dr. Pratt's grammar

processor, Lingol, into a Japanese version. Lingol, itself an interesting system, skillfully uses the characteristics of Lisp and matches the syntax tree with the function tree.
One of the original purposes for implementing ETL-Lisp was to "re-examine" an AI-oriented language which had first been proposed at MIT. I was interested in this language and introduced it to Japan, referring to it as the "inference" programming language.
Research on "theorem proving" was very popular in the latter half of 1960s. However, there was also "criticism to logic" as a reaction to the fact that research results turned out to be contrary to expectation. Partly because of the philosophical background, the criticism pointed out that the logical framework, which in this case was the first order predicate logic, was too generic and inefficient.
Dr. Hewitt's "Planner" was said to translate logic to procedures. The language understanding system of Dr. Winograd, which utilized u-Planner, had gained attention, and "procedural semantics" was advocated. Together, these were recognized as realizing the ideology of the "criticism to logic" of MIT's AI group.
Although I remember having a reservation or two, I felt I had better follow the predecessors' direction. However, to my surprise, the forerunner whom I intended to follow with respect suddenly lost his way.
Planner was also too slow. Then, based on this criticism, Dr. Sussman proposed "Conniver", stating that incomplete "anti-logicization" had caused the problem. However, it was also too slow. And, the majority of the AI group had returned to Lisp.

2.3 Encounter with Prolog

I had heard much criticism to logic from the MIT group. However, I felt that I need not consent to their ideology. Although I had no intention of soliciting agreement from others, I personally favored inference-sovereignty, like (Pierce). (My taste was clearly reflected in the name of the laboratory.)
I took first notice of the concept of logic programming in a paper by Dr. Kowalski (1974). The idea interested me greatly. And though a thorough explanation of his idea is unwarranted here, essentially, it interprets Horn logic, a subset of first order predicate logic, as a procedure, and regards the node set as a "program".
I thought that it was useful for the basic form of program translation. So, I tried to change the parsing algorithms of Drs. Pratt and Early to a logic form.
I was quite impressed with the result. It could explicitly represent the filter function and the structure which were essential to the algorithms. (I was later surprised to learn that the same function was developed in the database field and given the fancy name, "magic set".)
In spite of my experience, I didn't think the idea could be directly converted into an implementation. Even though it might have been implemented, I felt it would be too slow to work.
At that time, Dr. Koichi Furukawa had returned from a period of overseas study with

a copy of the listing of Prolog, something SRI had once tried to implement. However, SRI had failed with the implementation and had moved away from it several years earlier.
Because of Dr. Furukawa's view that it was an interesting system, I decided we would try to implement it, though with little expectation for success. After several weeks of hard work, it began to work, and it surprised us with its fast execution speed. It was very fast provided that it was an interpreter written in Fortran. Actually, it was twice as fast as I expected it to be. A further analysis of the details of its program revealed that the techniques used for implementation were rather skillful.
Since I was originally interested in the idea of logic programming, the implementation of an efficient processor completely changed my view. It was commonly believed that a logic based system was very slow. And, although it was biased, I also agreed with that view. (Many American researchers have agreed with this bias since then.)
On the other hand, young researchers, not burdened by useless remembrance, also thought that Prolog was rather slow compared with Lisp. Since the implementation technique for Prolog was largely common to that for Lisp, I thought that it might be possible to directly implement Prolog. (Original listing was only ten pages long.) I expected that the directly implemented Prolog would be as fast as Lisp.
Back then, Dr. Warren had constructed a similar type of Prolog implementation on DEC-10 in England. His system was efficient enough for practical use, and well-prepared as a total processing system. This system was a milestone, sparking an increase in the numbers of Prolog users around the world, as well as around me.

3 The Fifth Generation Computer Systems Project and Logic Programming

3.1 Computer Science and Logic

The 1970s was the period in which computer science established its roots. It was the period in which a logical approach had resulted in establishing a foundation, although there were criticisms about the logic. In place of the term, "logic", we can speak of a mathematical foundation as having been set. Anyway, it inevitably resulted in the use of a mathematical and logical framework.

3.1.1 Programming

In addition to logic programming, which was a direct example of the use of logic, "semantics" was explored in the field of programming. Semantics (Denotational Semantics = Model Theory) was a direct advance of the logical framework. Scott's theory of lambda computation was associated with advances in the set theory of the day. Hoare's axiomatic semantics was one of the attempts on the Proof Theory for procedural languages, and evolved into one kind of modal logic, so-called dynamic logic.

Both functional programming and object-oriented programming can be regarded as kinds of logic programming, which have firm common bases as well as unique characteristics and subjects.

3.1.2 Database

In the field of software, database has the same importance as programming. The relational data model proposed by Dr. Codd in 1970 and based on a subset of first order predicate logic (and Horn logic) had great bearing on this field.
Furthermore, the "subset" was more than simply a part of the total set. It had a mechanical significance since it could be used to elaborate advanced logic based on the (specific) characteristics of each field.
In the field of database, logic had been a more direct theme of investigation than in the field of programming. The theme of "logic and database" had often been discussed and experiments were done with deductive databases.

3.1.3 The Harmonic Association of Both

Essentially, the fields of programming and database are not two mutually exclusive fields. They should be regarded as nothing more than two aspects of the same world. Up to the present, there have been continuous attempts to connect independently developed systems by force. But, I think these attempts are unnatural.
It is necessary to develop subsystems for both fields within an integrated framework, while mechanically taking advantage of each one's specific natures. The same thing can be said for programming paradigms.
We thought that "logic" could form the common basis, and it was the main reason why we had adopted logic programming as the basis for the Fifth Generation Computer Systems project.

3.2 Work Hypothesis for Semantics

There was another more specific reason why I adopted logic programming as the basis of the Fifth Generation Computer Systems project.

3.2.1 Semantics = Program

The "procedural semantics" of Dr. Winograd tried to comprehend the semantics of terms as "programs". Here, a conceptual word corresponds with a program module. Modules are associated with each other to compose a program which corresponds with a sentence. The program runs to perform the function which the original sentence implies. This framework was momentous for its implementation of a natural language

understanding system.
The programming language, u-Planner, was intended to be used for this purpose. But, as mentioned earlier, the language was not implemented in reality. Is there any alternative language ? (An alternative approach was to abandon the framework itself, as Dr. Winograd did.)

3.2.2 Semantics = Logical Formula

In those days, Montague theory had gained the attention of a specific group studying philosophy. This theory had a great impact on philology. And though the achievements of the work had not been intended for computer technology, I was deeply interested in the framework.
With the framework, the semantics of a word corresponds with a "logical formula." Composing words results in a logical formula corresponding to the original sentence. I understood this had long been a goal of (some) philosophers. It was indeed epoch making that this framework was implemented for a natural language, instead of for only part of a natural language.
The composition of the logical formula was based on the framework of lambda conversion, using a kind of high order modal logic called intentional logic. The logic was given denotational semantics. In addition to the (philosophical) notation, what kind of computation did the logical formula represent ? The similarity between the framework and Lingol was also an interesting discovery for me.

3.2.3 Logical Formula = Program

As mentioned earlier, logic programming tried to interpret a kind of logical formula as a program. In this context, a very attractive three terms relationship was recognized. I was aware that logic programming was the missing link which bridged the three entities.
The above consideration, lacking evidence, could not be discussed in an academic paper. Indeed, there were countless gaps in the research design, which should have been investigated as "research themes." I knew investigations on these themes were not simple tasks.
On the other hand, I thought logic programming could be used as a basic framework or a working hypothesis for advancing a research project.

3.3 A Working Hypothesis for Parallel Inference

3.3.1 Advancement of Semi-conductor Technology and Parallel Inference

The continuous, exponential advancement in semiconductor technology began in the 70s and is expected to last into the 21th century. Based on this expectation, few

people opposed the view that parallel computers could be realized in terms of their hardware.
However, difficulties were expected in the software area for parallel computers. In particular, many of the researchers who had experienced failure in the development of a parallel computer in the '60s were quite skeptical.
The majority of researchers held the view that special purpose machines, e.g. those used for graphic image processing or large scale numerical computation of regular pattern might be constructed. However, (software for) a general purpose parallel computer containing irregular and complicated advanced symbol processing was highly likely to appear anytime soon.
By the late 70s, I agreed with that view.

3.3.2 Dataflow Model

In those days, I had a chance to acquaint myself with the current state of the research on dataflow machines. I did not expect much from dataflow machines, since I had known of their basic idea ten years earlier.
Now, what most impressed me was the dataflow language. At the time, I had been examining a plan (for the software) often mentioned earlier. I happened to notice that dataflow language could be viewed as a kind of logical language.
This view had allowed me to enlarge the plan for the range of hardware (architecture) at each stretch. For me, logic programming was the missing link which bridged the parallel machine architecture and software.

3.3.3 Parallel Inference

Basically, logic programs are independent of the method of execution. They can be interpreted sequentially. Prolog relies on this fact.
On the other hand, Prolog is different from languages which are originally designed to run sequentially in that it can run in parallel. As described below, it has abundant (wasteful) parallelism.
Anyway, I expected that the restriction of sequential processing would be released and that logic programming would provide the (basis for) organizational principle of the software for parallel machines. This idea had been embodied in the keyword "parallel inference" by the time the project start.

3.4 Starting the Project

In 1982, the project based on the above research plan started. It was not a traditional (follow-on) development project but, on the contrary, it was an epoch-making project which began by creating basic technology.

The project was in sync with plans at the Ministry of International Trade and Industry (MITI), which had been wanting a project that could establish new patterns of research. Of course, the expected din of opposition rose, which inevitably accompanies a large national project.
Serious criticisms and discussions centered around the point of why the project had employed logic programming as its basis.

3.4.1 Why Prolog and Why Not Lisp

This was a typical first reaction. Prolog was nothing more than a starting point; what we had really employed was the "idea" of logic programming. However, to our surprise, this was not truly understood by many researchers.
The first reason for this misunderstanding had to do with the sponsor. Since the project was sponsored by MITI, both foreign and domestic researchers mistakenly believed the research was meant to directly aid industry, or create a standard for future aid, as traditional national projects were wont to do.
The second reason was it might be regarded as a "rebellion" against Lisp which was mainstream in the US at the time. I was told face to face that it was an audacious selection. Some overseas mass media interpreted it as a "theological debate." As I mentioned earlier it was not an issue of selecting a language. Furthermore, I understood the real value of functional programming which was the basis of Lisp.
I believed nothing to be more barren than theological debate or ideological debate. I hate these kinds of debate. And, I have always stated that the employment of logic programming or other such ideas was nothing more than using "working hypotheses." I believe I am a relativist.
Although there were curious reactions at first, our "basic" idea gradually grew to be understood. In particular, younger researchers did not have a strong reaction to logic programming, while some senior researchers (my age) resisted it.

3.4.2 Principles in the Project

I basically welcomed free and varied ideas, but I wasn't going to be a relativist without principles.
I intentionally and decidedly selected logic programming, although the selection seemed to be too simple for some people. After I had examined all choices, my decision became a thoroughly simple one.
I believed that it was better for a single (research) project to have as few principles as possible. Tens of provisions or exhaustive exceptions cannot be regarded as a principle. The larger project should have a smaller number of principles. After all, it is the first principle which leads the project to success.
It might seem contrary to the practical wisdom, but I believe that the major factors which lead a project to failure are a research plan which contains various approaches

and a compromise for running the project.
I did not think that employing logic programming as the basis for research restricted new advancements in the research. On the contrary, I tried to establish a new fertile environment for the research. It was nothing more than a hypothesis. However, I thought that I had to stick to it, since it was a working hypothesis.

4 Achievements of the Fifth Generation Computer Systems Project

During the first period of the project (three years), we designed theESP language which was an extension of Prolog and we implemented its prototype. In addition, we also designed and implemented the workstation PSI corresponding to ESP and having a (high level) machine language. The operating system Simpos for PSI written only in ESP was the world's first large scale software written in a logic language.
More than 400 PSI systems were constructed (including revised versions), and they provided a basis for later research.
We have concentrated on the research of parallel inference since then. The following sections describe an overview of the major achievements of the research, focusing on logic programming.

4.1 GHC/KL1

The first research theme was the parallel logic language. Investigating logic language for parallel machines had been a project subject from the beginning. This theme was investigated by the team led by Dr. Furukawa. In this field, the proposition of a language (later called Parlog) by Dr. Clark of England, as well as the proposition of Concurrent Prolog by Dr. Shapiro preceded our project.
Dr. Furukawa and his colleagues were carefully examining these languages and invited young and energetic researchers from abroad to join the investigation. After the investigation was deeply engaged, Dr. Kazunori Ueda proposed a new language model, GHC. This was at the end of the first period of the project.
The GHC model had a simple, yet sophisticated structure while inheriting the functionality of the preceding languages. After a variety of examinations, we decided to adopt this model as a basis for succeeding research.
Based on GHC, Dr. Takashi Chikayama designed a practical programming language, KL1. And, the hardware group decided to adopt KL1 as the basis for designing parallel machines from that time forward. (This decision was to result in the development of parallel inference machine, PIM, during the last period of the project.)
Dr. Chikayama and his colleagues had developed the KL1 implementation and the operating system Pimos for the parallel machine which used the implementation. This implementation had proved that a parallel logic language could be used in practice.

KL1/GHC was based on Horn logic and employed "constraints" for controlling parallel actions. This enabled the system to naturally build in functions such as dataflow synchronization.
While some approved, other researchers claimed (?) that adding constraints had distanced the system from pure logic. I thought that many researchers strongly believed that logic indicated nothing more than the "first order predicate logic" illustrated in a textbook (although the y were not explicitly conscious of the fact). Researchers who were opposed to logic were usually such purists.
It was surely one of the members in the family of logic languages. I think that it forms a structure for the kernel of logic programming, even if parallelism is excluded.

4.2 Quixote

Since GHC/KL1 was an attempt to pursue a "low level" structure of thelogic programming language, another question was how to up grade the structure to a higher level. This coincided with the direction for which a knowledge representation language was being sought.

4.2.1 CIL/Situation Semantics

We gained our first achievement at the end of the first period. It resulted from the research on the natural language understanding system of Drs. Kuniaki Mukai and Hideki Yasukawa.
In those days, Dr. Barwise and his colleagues proposed "situation semantics" in order to overcome the limitations in Montague's theory. They also included "non-standard" set theory which led to the extension of symbol structures (such as the rational tree expressions which were later proposed by the Colmerauer, the inventor of Prolog). Dr. Mukai and his colleagues invented the programming language, which represented the model after minutely examining the theory.
CIL also included advanced attempts, such as non-determinate terms and extensions of control structure, which would later lead to the "constraint logic programming."

4.2.2 Constraint Logic Programming

During the '80s, research on logic programming made rapid progress all over the world. One of the notable advances was the constraint logic programming proposed by Drs. Jaffer and Lassez. (Dr. Colmerauer was also a forerunner in this area).
It tried to comprehend the behavior of a logic program at the level of "constraint" which controlled inference, without reducing all behaviors into a level of basic inference. This mechanism provided the framework to build in solvers specific to each problem domain (such as the solvers for equations or inequalities within the real number domain). It can be regarded as an extension or advancement of control structure.

At ICOT, the CAL processing system was implemented and used for research during the middle period. In the last period, the language, GDCC, and its processing system, which was extended to cope with parallelism, were implemented.

4.2.3 The Non-regular Relational Data Model

ICOT adopted the relational database model as the basis for the knowledge base system.
During the project's middle period, a team led by Dr. Kazumasa Yokota developed the "non-regular model" by extending the standard relational model, and then implemented it on PSI. The team also developed a prototype of CRL which was a (query) logic language for deductive databases. (The non-regular database system was extended into a parallel version and implemented during the last period.)
Harmonizing logic with object orientation had been an outstanding issue from the project's inception. Early on, Dr. Yokota and his colleagues proposed DOOD (deductive object-oriented database) and proceeded in this direction with their work. Their efforts bore fruit in the last period.

4.2.4 Don's Intention

Actually, throughout the project's beginning and middle periods, the research conducted by Dr. Yokota's team had remained independent of the natural language research that Dr. Mukai's team was doing. However, they had each reached some very similar points, especially with regard to the programming languages.
So, I forced them to make a combined team in the last period (although I was a little worried that my demand appear unreasonable).
The team then produced more results than I expected. A new language was created by harmonizing their independent researches, CIL and CRL. It was a very natural harmonization rather than an unreasonable combination.
The language called, Quixote, organically integrated factors, such as a concurrent logic programming language, a situated programming language, and an object-oriented database programming language. It had a flexible term configuration like the non-determinant term, which represented an object. The set of terms inherited its attributes using the partial order relationship. The overall structure was a hierarchical modules structure.
Quixote has since been used to represent the natural language semantics, the molecule biological database, and the legal reasoning system.

4.3 MGTP

As you know, one of logic programming's roots can be found in theorem proving research (and another in the research of natural language processing). However, the

theorem proving research root made little progress during that period.
The original experience in the '60s had badly affected the progress of theorem proving research. It had historically taken the popular view that first order predicate logic was too general. From this, logic programming limited its arena to the field of programming (or database).
Indeed, although the prover of first order predicate logic does not correspond with theorem proving on a one-to-one basis, it is difficult to avoid dealing with the issue of the first order predicate logic prover.
ICOT also dealt with this issue only in the form of a (mathematical) proving "support" system. As a result, the system CAP was constructed. However, after using the system, we reach the opinion that a support system should contain the ability for proving within itself, even though it was a support system.
In the '60s, a variety (almost all) of provers had been proposed, and I believed their mathematical research to be complete.
However, in a sense, (except in Goedel's meaning) research on a prover does not have a mathematical value by itself. The prover is nothing more than a tool. This indicates that it is basically an issue of engineering, though it requires mathematical support.
In the '80s, many techniques had been accumulated through logic programming. The technology for computers also advanced. We had the world's fastest (parallel) computer for symbol processing. It could not turn mathematical infinity to finiteness, but I thought it was the time to try again.
I still had not abandoned the hope of constructing a theorem prover; Dr. Furukawa hadn't either. And, at the same time, Dr. Ryuzo Hasegawa announced he intended to construct the world's fastest theorem prover.
Fortunately, we were able to make use of Dr. Bry's Satchmo. It was a model generation method, but at the first glance it seemed to be nothing more than a derivative of the Tableau method or Hyper Resolution. But, to our surprise, a program of several Prolog lines could be executed very fast. One of the keys to the fast execution was limitation of domains. Many problems could be reduced to the ground level (as databases had already done).
Examination of Satchmo revealed that it could be well adapted to parallel processing. Dr. Hasegawa and his colleagues constructed an ultra-fast prover on a parallel machine. They also constructed a non-ground version.
As a result, it could be used to solve some of the outstanding mathematical problems (including problems of Bennet's quasi-group). In addition to solving mathematical problems, it was used as the engine for legal reasoning systems. It was also employed as the base system for the hypothesis inference system and the modal logic system.
The focus of logic programming was being limited to Horn logic. But, at this stage, there arose a possibility to go back to the original background as a chance to enhance the technology.

4.4 Miscellaneous

Mentioned above are what I think are some of the typical achievements. Beyond that, we have too many achievements, which are valuable as research results, to be listed here.
Some of these include logic researches, non-monotonic inference, inference by analogy, partial evaluation, program translation, etc.
They also include large-scale application problems, the system which generates a plant control program of some thousands of lines using temporal (modal) logic, the natural language analysis system which works in parallel, the LSI-CAD system for hard-wiring, disposition and logical simulation, the legal reasoning system which uses past cases and provisions, the genetic information analysis system, etc.
They were all developed on the basis of logic programming. During the last period of the project, almost all of the programs were developed in the parallel logic language KL1. While they mainly aimed at examining new technologies, rich experience was obtained for logic programming.
Impressions on the Results: If there had been a research project with the same objective as the Fifth Generation Computer Systems project, logic programming might not have been the only selection, having no alternative. Of course, several choices might have been considered other than ICOT's choice, even if logic programming had been adopted as the basis.
On the other hand, we think the employment of logic programming had proven to be the most effective working hypothesis, given the results.

5 The Future of Logic Programming

5.1 Dissemination of the Research Results

I think the research on logic programming has now reached a turning point, since a fairly large amount of the results have been obtained.
One of the remaining issues is just how to take full advantage of the research results in the field of practical information processing.
Certainly, logic programming leaves a great deal of room for further investigation. However, since it is more than a prototype, its "idea" will be disseminated in the research society.
I think the future of logic programming is worth noting. Sequential programming finally resulted in the combined use of a variety of paradigms. What will happen in the field of parallel (concurrent) programming, which has now reached the starting point for a practical use?
There may be approaches in which concurrent Lisp or functional programming are adopted as their starting point. Practical researchers may rely on the improvement of C language. I am not sure that they will succeed. I believe that concurrent logic languages are most hopeful. However, the tendency may be to adopt a somewhat more

useless approach in the real world.
From the viewpoint of the user, what is most important is integrating programming paradigms, and also integrating programming and database. Although room is left for further research, I think the researches of these ten years have combined to lay a firm groundwork for future advancement. It is also necessary that further efforts be made to make use of the research results in the practical world.

5.2 Various Logic Systems

Another important issue is the enlargement of logic programming's field of research. First order predicate logic is not the only logic system. There are also partial logic systems, intuitive logic systems, and modal logic, as well as other formalizations. Furthermore, mathematical logic suggests there may be an infinite number of logic systems. However, countless logic systems are organically related to each other, rather than isolated. Standard logic systems are usually used for the relationships.
I believe most of the various AI attempts up to this point can be rearranged as logic systems. Rearrangement can clearly point out the characteristics of each logic system and clarify the way to harmonize various logic systems with each other while maintaining each one's merit.
(Sequential Type) Semantics for programs can be rearranged as dynamic logic. The dynamic aspects of a program can be well-understood as a type of modal logic rather than flat predicate logic.
I believe modal logic should be examined again from the viewpoint of logic programming rather than simply using existing systems.

5.3 Deduction, Abduction, Induction

Classifying logic inference into three category is rather traditional. Roughly speaking, modern mathematical logic has mathematically defined the inference part of mathematical proof. It inevitably includes modeling of "computation." Logic programming is mainly concerned with this point.
On the other hand, in many cases, AI research can be seen as attempts to extend the area of inference.
Theorem proving is deductive inference. While diagnostic issues have been regarded as abducting inference, hypothesis inference can also be regarded as an extension of abducting inference. Learning is an issue of inductive reasoning.
Although logic programming is generally regarded as part of deduction, I do not think it is always true. In my view, symbol logic has clarified the symbolic elemental process, which lies in the basis of inference, rather than clarifying the structure of deduction.
In my view, overall structure is not clarified even for deduction (or proving). However, for example, compared with induction, deduction is relatively well-defined.
In the history (of philosophy), inductive inference could not have been understood as

a set of simple rules or a system of symbol logic. No other logic system exists for inductive inference.

In other words, the three kinds of inference mechanisms have common logical elemental processes as their bases which correspond with logic programming.

The research on the inductive inference system based on logic programming is expected to become an important field. Recent results of Muggleton's research have supported this expectation.

The future direction for logic programming is finding ways to cope with various types of inference.