Middle History of Logic Programming

Resolution, Planner, Prolog, and the Japanese Fifth Generation Project

Carl Hewitt
http://carlhewitt.info

Logic Programming can be broadly defined as "using logic to infer computational steps from existing propositions" (although this has been opposed by Kowalski; see below). The focus of this article is on the development of this idea. Consequently, it does not treat any other associated topics related to Logic Programming such as constraints, abduction, *etc.*

   The idea has a long development that went through many twists in which important questions turned out to have surprising answers including the following:

- Is computation reducible to logic?
- Did Logic Programming contribute to the failure of the Japanese Fifth Generation Project (ICOT)?
- What is Logic Programming?

Note: This article is about the middle history of Logic Programming. See "*A historical perspective on developing foundations for client cloud computing"* [ArXiv 0901.4934] for conceptual development of Logic Programming from its origins to the present

Contents

Uniform Proof Procedures based on Resolution

John Alan Robinson [1965] developed a deduction method called resolution which was proposed as a uniform proof procedure for proving theorems which
Converted everything to clausal form and then used a method analogous to modus ponens to attempt to obtain a proof by contradiction by the adding clausal form of the negation of the theorem to be proved.

Using resolution as the only rule of inference is problematical because it hides the underlying structure of proofs. Also using proof by contradiction is problematical because the axiomatizations of all practical domains of knowledge are inconsistent in practice. And proof by contradiction is not a sound rule of inference for inconsistent axiomatizations .

Resolution uniform proof procedures were used to generate some simple proofs [Wos 1965; Green 1969; Waldinger and Lee 1969; Anderson and Bledsoe 1970; *etc.*]. However, in the resolution uniform proof procedure theorem proving paradigm, the use of procedural knowledge was considered to be "*cheating*" [Green 1969].

Procedural Embedding of Knowledge redux

The two major paradigms for constructing semantics software systems were procedural and logical. The procedural paradigm was epitomized by Lisp [McCarthy *et. al.* 1962] which featured recursive procedures that operated on list structures. The logical paradigm was epitomized by uniform resolution theorem provers [Robinson 1965].

Planner

Planner [Carl Hewitt 1969] was a kind of hybrid between the procedural and logical paradigms in that it featured a procedural interpretation of logical sentences in that an implication of the form (P implies Q) can be procedurally interpreted in the following ways [Hewitt 1969]:
> Forward chaining
>> *When assert* P, *assert* Q
>> *When assert not* Q, *assert not* P
> Backward chaining
>> *When goal* Q, *goal* P
>> *When goal not* P, *goal not* Q

Planner was the first programming language based on the pattern-directed invocation of procedural plans from assertions and goals. The development of Planner was inspired by the work of Karl Popper [1935, 1963], Frederic Fitch [1952], George Polya [1954], Allen Newell and Herbert Simon [1956], John McCarthy [1958, *et. al.* 1962], and Marvin Minsky [1958].

Planner represented a rejection of the resolution uniform proof procedure paradigm.

A subset called Micro-Planner was implemented by Gerry Sussman, Eugene Charniak and Terry Winograd as an extension to Lisp primarily for pragmatic reasons since it saved memory space and processing time (both of which were scarce):

- Lisp was very well suited to the implementation of a Micro-Planner interpreter.
- The full functionality of Lisp libraries were immediately available for use by Micro-Planner programs.
- The Lisp compiler could be used to compile Lisp programs used by Micro-Planner applications to make them smaller and run faster. (It was unnecessary to first implement a Micro-Planner compiler.)

Micro-Planner was used in Winograd's natural-language understanding program SHRDLU [Winograd 1971], Eugene Charniak's story understanding work, work on legal reasoning [McCarty 1977], and some other projects. This generated a great deal of excitement in the field of AI.[i]

Efficiency Expedients

Computers were expensive. They had only a single slow processor and their memories were very small by comparison with today. So Planner adopted some efficiency expedients including the following:

- Backtracking [Golomb and Baumert 1965] was adopted to economize on the use of time and storage by working on and storing only one possibility at a time in exploring alternatives.
- A unique name assumption was adopted to save space and time by assuming that different names referred to different objects. For example names like Peking and Beijing were assumed to refer to different objects.
- A closed world assumption could be implemented by conditionally testing whether an attempt to prove a goal exhaustively failed. Later this capability was given the misleading name "negation as failure" because for a goal G it was possible to say: "if attempting to achieve G exhaustively fails then assert (*Not* G)." [Davies 1974] has a system for maintaining consistency in a closed world. (See the discussion below concerning negation as failure in Prolog.)

Control structure controversy

In several ways, backtracking proved unwieldy helping to fuel the great control structure debate.[ii] Hewitt investigated some preliminary alternatives in his thesis. Using program schemas, Hewitt in collaboration with Mike Paterson proved that recursion is more powerful than iteration and parallelism more powerful than sequential recursion [Patterson and Hewitt 1970].

Hairy control structure

Peter Landin had introduced an even more powerful control structure using his **J** (for Jump) operator that could perform a nonlocal goto into the middle of a procedure invocation [Landin 1965]. In fact the **J** operator enabled a program to jump back into the middle of a procedure invocation even after it had already returned! Drew McDermott and Gerry Sussman called Landin's concept "*Hairy Control Structure*" and used it in the form of a nonlocal goto for the Conniver programming language [McDermott and Sussman 1972]. Hewitt and others were skeptical about hairy control structure. Pat Hayes [1974] remarked: *Their* [Sussman and McDermott] *solution, to give the user access to the implementation primitives of Planner, is however, something of a retrograde step (what are Conniver's semantics?)*

However, there was there germ of a good idea (previously emphasized in Polya [1957] and "progressive deepening" [de Groot 1965]) in Conniver; namely, using co-routines to computationally shift focus to another branch of investigation while keeping alive the one that has been left Scott Fahlman used this capability of Conniver to good effect in his in his planning system for robot construction tasks [Fahlman 1973] to introduce a set of higher-level control/communications operations for its domain. However, the ability to jump back into the middle of procedure invocations didn't seem to be what was needed as the foundation to solve the difficulties in communication that were a root cause of the control structure difficulties.

The difficulties using backtracking in Planner and Conniver were useful in that they provoked further research into control structures for the procedural embedding of knowledge.

Control structures are patterns of passing messages

In 1972 Alan Kay visited MIT and gave an inspiring lecture that explained some of his ideas for Smalltalk-72 building on the message-passing of Planner and Simula [Dahl and Nygaard 1967] as well as the Logo work of Seymour Papert with the "little person" model of computation used for teaching children to program (cf. [Whalley 2006]). However, the message passing of Smalltalk-72 was quite complex [Ingalls 1983]. Also, as presented by Kay, Smalltalk-72 (like Simula before it) was based on co-routines rather than true concurrency.

The Actor model [Hewitt, Bishop, and Steiger 1973] was a new model of computation that differed from previous models of computation in that it was inspired by the laws of physics.[iii] It took some time to develop programming methodologies for the Actor model. Hewitt reported
> ... *we have found that we can do without the paraphernalia of "hairy control structure" (such as possibility lists, non-local gotos, and assignments of values to the internal variables of other procedures in CONNIVER.)...* ***The conventions of ordinary message-passing seem to provide a better structured, more intuitive foundation for constructing the communication systems needed for expert problem-solving modules to cooperate effectively.***[iv] (emphasis in original)

Planner aimed to extend what could be programmed using logical methods but did not take a stand about the theoretical limits of these methods. However, once the Actor model was invented in late 1972[v], it became clear that logical inference alone would not suffice for

computation because the order of Actor message arrival could not always be logically inferred. And so work on Planner ceased in favor of intensive investigation of the Actor model.

The Genesis of Prolog

Gerry Sussman and Seymour Papert visited Edinburgh spreading the news about Micro-Planner and SHRDLU casting doubt on the resolution uniform proof procedure approach that had been the mainstay of the Edinburgh Logicists. According to Maarten van Emden [2006]

> *The run-up to the workshop* [Machine Intelligence 6 organized by Donald Michie in 2001] *was enlivened by telegrams from Seymour Papert at MIT announcing on alternating days that he was (was not) coming to deliver his paper entitled "The Irrelevance of Resolution", a situation that caused Michie to mutter something about the relevance of irresolution. The upshot was that a student named Gerry Sussman appeared at the appointed time. It looked as if this was going to be his first talk outside MIT. His nervousness was compounded by the fact that he had been instructed to go into the very bastion of resolution theorem proving and tell the assembled experts how totally misguided they were in trying to get anything relevant to AI with their chosen approach.*
>
> *I had only the vaguest idea what all this was about. For me theorem proving was one of the things that some people (including Kowalski) did, and I was there for the programming. If Bob and I had anything in common, it was search. Accordingly I skipped the historic Sussman lecture and arrived late for the talk scheduled to come after Sussman's. Instead, I found an unknown gentleman lecturing from a seat in the audience in, what I thought a very English voice. It turned out that a taxi from the airport had delivered Seymour Papert after all, just in time for the end of Sussman's lecture, which was now being re-done properly by the man himself.*
>
> *The effect on the resolution people in Edinburgh of this frontal assault was traumatic. For nobody more so than for Bob Kowalski. Of course there was no shortage of counter objections, and the ad hoc creations of MIT were not a pretty sight. But the occasion hit hard because there was a sense that these barbarians had a point.*

Previously at Edinburgh, Pat Hayes and Bob Kowalski had collaborated on resolution theorem proving. Then Hayes visited Stanford where Bruce Baumgart published his *Micro-Planner Alternate Reference Manual* in April 1972. Hayes says that from the time that he learned about Micro-Planner it seemed obvious to him that it was based on controlled deduction.[vi]

When he returned to Edinburgh, he talked about his insight with anyone who would listen and gave internal seminars at two of the major departments at Edinburgh concerned with logic. In the third department, Hayes point seemed irrelevant because they were busy getting their hands on the latest "magic machinery" for controlling reasoning using Popler [Davies 1973], a derivative of Planner. Hayes wrote a joint paper with Bruce Anderson on "*The Logicians Folly*" against the resolution uniform proof procedure paradigm [Anderson and Hayes 1972].

The above developments generated tension among the Logicists at Edinburgh. These tensions were exacerbated when the UK Science Research Council commissioned Sir James Lighthill to write a report on the AI research situation. The resulting report [Lighthill 1973; McCarthy 1973] was highly critical although SHRDLU [Winograd 1971] was favorably mentioned. "*Resolution theorem-proving was demoted from a hot topic to a relic of the misguided past. Bob* [Kowalski] *doggedly stuck to his faith in the potential of resolution theorem proving. He carefully studied Planner.*" [Bruynooghe, Pereira, Sickmann, and van Emden 2004]

Kowalski [1988] states "*I can recall trying to convince Hewitt that Planner was similar to SL-resolution.*" But Planner was invented for the purposes of the procedural embedding of knowledge and was a rejection of the resolution uniform proof procedure paradigm for proving theorems. Furthermore, Planner's use of forward chaining reasoning did not fit within SL resolution.

van Emden [2006] recalled:

> *Kowalski's apparent research program narrowed to showing that the failings so far of resolution inference were not inherent in the basic mechanism. He took great pains to carefully study PLANNER and CONNIVER. And painful it was. One of the features of the MIT work was that it assumed the audience consisted of LISP programmers. For anybody outside this circle (Kowalski most definitely was not a LISP programmer), the flavour is repellent.*

Kowalski [2008] later recalled:

> *In the meanwhile, critics of the formal approach, based mainly at MIT, began to advocate procedural representations of knowledge, as superior to declarative, logic-based representations. This led to the development of the knowledge representation and problem-solving languages Planner and micro-Planner. Winograd's PhD thesis (1971), using micro-Planner to implement a natural language dialogue for a simple blocks world, was a major milestone of this approach. Research in automated theorem-proving, mainly based on resolution, went into sharp decline.*
>
> *The battlefield between the logic-based and procedural approaches moved briefly to Edinburgh during the summer of 1970 at one of the Machine Intelligence Workshops organized by Donald Michie (van Emden, 2006). At the workshop, Papert and Sussman from MIT gave talks vigorously attacking the use logic in AI, but did not present a paper for the proceedings. This created turmoil among researchers in Edinburgh working in resolution theorem-proving. However, I was not convinced that the procedural approach was so different from the SL resolution system I had been developing with Donald Kuehner (1971).*
>
> *During the next couple of years, I tried to reimplement Winograd's system in resolution logic and collaborated on this with Alain Colmerauer in Marseille. This led to the procedural interpretation of Horn clauses (Kowalski 1973/1974) and to Colmerauer's development of the programming language Prolog.*

Hayes was astonished when Kowalski wrote back from Marseilles saying that he and Colmerauer had a revolutionary idea that logic could be used for programming. Feeling that his ideas were being unfairly appropriated by Kowalski, Hayes complained to the head of their unit Bernard Meltzer. Feeling that he wasn't getting satisfaction, Hayes wrote a summary and exegesis of his ideas in a paper for the proceedings of a summer school in Czechoslovakia with the idea of recording the priority of his ideas [Hayes 1973].

However, Kowalski felt that his work with Colmerauer bore little resemblance to anything that had been discussed previously in Edinburgh by Hayes. His claim was that Hayes' ideas (and the paper that he published) were based on using equations for computation (in the spirit of the work in Aberdeen) that are very different from both the Planner and Prolog views of logic

programming. [van Emden and Kowalski 1974] developed a procedural semantics of Horn clauses that was derived from backward chaining in Planner. [Hayes 1974] complained that van Emden and Kowalski "*alien semantics*" for the predicate calculus was analogous to "*This is throwing out the baby and keeping the bathwater.*" Subsequent to Edinburgh, Hayes took up a position at Essex where he pursued his ideas in the Golux[vii] Project with Bruce Anderson. Many years later Kowalski and Hayes patched things up, *e.g.*, see [Kowalski 1988].

Colmerauer and Roussel recalled their reaction to learning about Planner in the following way:
> *While attending an IJCAI convention in September '71 with Jean Trudel, we met Robert Kowalski again and heard a lecture by Terry Winograd on natural language processing. The fact that he did not use a unified formalism left us puzzled. It was at this time that we learned of the existence of Carl Hewitt's programming language, Planner [Hewitt, 1969]. The lack of formalization of this language, our ignorance of Lisp and, above all, the fact that we were absolutely devoted to logic meant that this work had little influence on our later research.* [Colmerauer and Roussel 1996]

Kowalski developed SLD resolution at Marseille in the summer of 1972, which he maintains provides a logical framework for the backward chaining of Micro-Planner. On the other hand, the direct procedural interpretation of implication originally developed for Planner [Hewitt 1969, 1971] provides a simpler logical framework for backward chaining that is compatible with direct inference [Hewitt 2008b] (unlike resolution). [viii]

In the fall of 1972, Roussel implemented a language called *Prolog* (an abbreviation for "**PRO**grammation en **LOG**ique" (French for *programming in logic*)). Prolog programs are generically of the following form (which is a special case of the backward-chaining in Planner):
> *When goal* Q*, try goal* P_1 *and ... and goal* P_n

Prolog was basically a subset of Planner that restricted programs to clausal form using backward chaining and consequently had a simpler more uniform syntax. (But Prolog did not include the forward chaining of Planner.) Like Planner, Prolog provided the following:
- o An indexed data base of pattern-directed procedures and ground sentences.
- o The Unique Name Assumption, by means of which different names are assumed to refer to distinct entities, *e.g.,* Peking and Beijing are assumed to be different.

Prolog implemented a number of non-logical computational primitives for input-output, *etc.* Like Planner, for the sake of efficiency, it used backtracking. Prolog also had a non-logical computational primitive like the one of Planner to control backtracking by conditionally testing for the exhaustive failure to achieve a goal by backward chaining. However, Prolog was incapable of expressing strong "Negation as Failure" because it lacked both the assertions and true negation of Planner and thus it was impossible in Prolog to say "if attempting to achieve the goal G exhaustively fails then assert (*Not* G)." Prolog extended Planner by using unification (but not necessarily soundly because for efficiency reasons it omitted use of the "occurs" check).

Prolog omitted a number of logical features of Micro-Planner including:
- o Pattern-directed invocation of procedural plans from assertions (*i.e.*, "forward chaining")
- o Logical negation, *e.g.*, (not (human Socrates)).

A consequence of the above shortcuts and compromises was that Prolog, like Planner, lacked elegance. McCarthy [2000] noted "*Micro-Planner was a rather unsystematic collection of tools, whereas Prolog relies almost entirely on one kind of Logic Programming, but the main idea is the same.*"[ix]

In summary, Prolog was basically a subset of Planner that restricted programs to clausal form using backward chaining.

In 1980, Drew McDermott gave his take on the situation as follows:
> At about the time the [Planner-like] *AI languages were dying here, several Europeans, notably Alain Colmerauer [Roussel 75] and Robert Kowalski [van Emden 76], rediscovered the procedural interpretation of deduction. This was embodied in a language called PROLOG (for PROgramming in LOGic") that seemed remarkably like PLANNER. Most Americans probably thought this was just the beginning of a delayed version of the events here, and expected disillusionment to set in fairly quickly.*
> *   This has not happened. PROLOG has attracted and held a user community that is as fanatically devoted as most Americans are to LISP.*

Controversies

There are a number of controversies involved in the history of logic programming including "What is logic programming?", "Is Logic Programming computationally universal?" and "Did Logic Programming contribute to the failure of the Japanese Fifth Generation Project (ICOT)?"

Is Computation Subsumed by Deduction?

The notion of computation has been evolving for a long time. One of the earliest examples was Euclid's GCD algorithm. Next came mechanical calculators of various kinds. These notions were formalized in the Turing Machines, the lambda calculus, etc. paradigm that focused on the "state" of a computation that could be logically inferred from the "previous" state.

The invention of digital computers caused a decisive paradigm shift when the notion of an interrupt was invented so that input that arrived asynchronously from outside could be incorporated in an ongoing computation. The break was decisive because asynchronous communication cannot be implemented by Turing machines etc. because the order of arrival of messages cannot be logically inferred. Message passing has become the foundation of many-core and client-cloud computing.

Kowalski [1979] published the thesis that "*Computation = controlled deduction*" which he states was first proposed by Hayes [1973].[x] However, Kowalski published the thesis 6 years after the invention of Actors [Hewitt, Bishop, and Steiger 1973] at a time when concurrency was already well established. The gauntlet was officially thrown in *The Challenge of Open Systems* [Hewitt 1985] to which [Kowalski 1988b] replied in *Logic-Based Open Systems*. This

was followed up with *Guarded Horn clause languages: are they deductive and logical?* [Hewitt and Agha 1988] in the context of the Japanese Fifth Generation Project (see section below). All of this was against Kowalski who stated "*Looking back on our early discoveries, I value most the discovery that computation could be subsumed by deduction.*" [Kowalski 1988a].

According to Hewitt *et. al.* and contrary to Kowalski and Hayes, computation in general cannot be subsumed by deduction and contrary to the quotation (above) attributed to Hayes computation in general is not controlled deduction. Hewitt and Agha [1991] and other published work argued that mathematical models of concurrency did not determine particular concurrent computations as follows: The Actor Model makes use of arbitration for determining which message is next in the arrival order of an Actor that is sent multiple messages concurrently. For example Arbiters can be used in the implementation of the arrival order of messages sent to an Actor which are subject to indeterminacy in their arrival order. Since arrival orders are in general indeterminate, they cannot be inferred from prior information by mathematical logic alone. Therefore mathematical logic cannot implement concurrent computation in open systems.

In concrete terms for Actor systems, typically we cannot observe the details by which the arrival order of messages for an Actor is determined. Attempting to do so affects the results and can even push the indeterminacy elsewhere. Instead of observing the internals of arbitration processes of Actor computations, we await outcomes. Indeterminacy in arbiters produces indeterminacy in Actors. The reason that we await outcomes is that we have no alternative because of indeterminacy.

It is important to be clear about the basis for the published claim about the limitation of mathematical logic. The claim is that because of the indeterminacy of the physical basis of communication in the Actor model, that no kind of deductive mathematical logic can always infer which message will arrive next and the resulting computational steps.

What does the mathematical theory of Actors have to say about this? A closed system is defined to be one that does not receive communications from outside. Actor model theory provides the means to characterize all the possible computations of a closed Actor system in terms of the Concurrency Representation Theorem [Hewitt 2006]: The denotation Denote_s of an Actor system S represents all the possible behaviors of S as

$$\mathsf{Denote}_s = \sqcup_{i \in \omega} \, \mathsf{Progression}_s^{\,i}(\perp_s)$$

$\mathsf{Progression}_s$ is an approximation function that takes a set of approximate behaviors to their next stage and $\perp_s$ is the initial behavior of S. Consequently, Logic Programming can represent but not in general implement concurrent systems.

Thus the Procedural Embedding of Knowledge paradigm is strictly more general than the Logic Programming paradigm.

The Japanese 5th Generation and Logic Programming

Beginning in the 1970's, Japan took the DRAM market (and consequently most of the integrated circuit industry) away from the previous US dominance. This was accomplished with the help of the Japanese VLSI project that was funded and coordinated in good part by the Japanese government Ministry of International Trade and Industry (MITI) [Sigurdson 1986]. MITI hoped to repeat this victory by taking over the computer industry. However, Japan had come under criticism for "copying" the US. One of the MITI goals for ICOT was to show that Japan could innovate new computer technology and not just copy the Americans.

According to Kowalski [2004a],
> *The announcement of the FGCS Project in 1981 triggered reactions all over the world. It gave rise to the Alvey Programme in the UK, to the joint research institute ECRC in Munich and to a similar institute MCC in Austin Texas. It may even have been one of the main triggers for the European Union research programme, ESPRIT.*
>
> *Logic Programming was virtually unknown in mainstream Computing at the time, and most of its research activity was in Europe. So it came as a big shock - nowhere more so than in North America - when it eventually became obvious that logic programming was to play a central, unifying role in the FGCS Project.*

The FGCS project (named ICOT), partly influenced by Logic Programming enthusiasts, tried to go all the way with Logic Programming. Kowalski later recalled "*Having advocated LP* [Logic Programming] *as a unifying foundation for computing, I was delighted with the LP focus of the FGCS project."* [Fuchi, Kowalski, Ueda, Kahn, Chikayama, and Tick 1993].

By making Logic Programming (which was mainly being developed outside the US) the foundation, MITI hoped that the Japanese computer industry could leapfrog the US.

This meant that ICOT had to deal with concurrency and consequently developed concurrent programming languages based on clauses that were loosely related to logic [Shapiro 1989]. However, it proved difficult to implement clause invocation in these languages as efficiently as procedure invocation in object-oriented programming languages. Simula-67 originated a hierarchical class structure for objects so that message handling procedures (methods) and object instance variables could be inherited by subclasses. Ole-Johan Dahl [1967] invented a powerful compiler technology using dispatch tables that enabled message handling procedures in subclasses of objects to be efficiently invoked. The combination of efficient inheritance-based procedure invocation together with class libraries and browsers (pioneered in Smalltalk) was better than the slower pattern-directed clause invocation of the FGCS programming languages. Consequently, the ICOT programming languages never took off and instead concurrent object-oriented message-passing languages like Java and C# became the mainstream.

The technical managers at ICOT were aware of some of the pitfalls that had tripped up previous Artificial Intelligence (AI) researchers. So they deliberately avoided calling ICOT an AI Project. Instead they had the vision of an integrated hardware/software system [Uchida and Fuchi

1992]. However, the Logic Programming paradigm turned not to be a suitable foundation because of poor modularity and lack of efficiency by comparison with direct message passing [Hewitt and Agha 1988].

 "The [ICOT] *project aimed to leapfrog over IBM, and to a new era of advanced knowledge processing applications."* [Sergot 2004] But the MITI strategy backfired because the software wasn't good enough (as explained above) and so the Japanese companies refused to productize the ICOT hardware.

Thus the way that ICOT used Logic Programming was a principle contributing cause to its failure.

What is logic programming?

Kowalski recalls that
> *The term "logic programming" was first used at a workshop organized by Robert Kowalski at Imperial College London in 1976. It was subsequently used to describe the general area associated with the use of logic as a programming language, extending the procedural interpretation of Horn clauses with negation as failure, and their implementation in languages such as Prolog*

The ALP (Association of Logic Programming) has provided the following Background statement:
> *Logic Programming was born circa 1972, presaged by related work by Ted Elcock, Cordell Green, Pat Hayes and Carl Hewitt on applying theorem proving to problem solving and to question-answering systems. It blossomed from Alan Robinson's seminal contribution, the Resolution Principle, all the way into a practical programming language with automated deduction at its core, through the vision and efforts of Alain Colmerauer and Bob Kowalski. Their work was followed up by the Pioneers of the field and many others, until it took strong hold in the academic community and became the basis of important scientific projects such as Fifth Generation Computing.*

Recently Kowalski remarked:
> *One issue is whether logic programming is to be understood ... as: "based upon the fact that a backwards reasoning theorem-prover applied to declarative sentences in the form of implications*
> B_1 and ... and B_n implies H
> *treats the implications as goal-reduction procedures*
> to show/solve H, show/solve B_1 and ... and B_n."
> *I believe that this is the generally accepted view of logic programming. It is a very restricted form of logic, but it is sufficient for Turing-computability.* [Kowalski 2007]

Kowalski's approach leads directly to the conclusion that Logic Programming is not computationally universal (see section above). On the other hand since 1969, Hewitt *et. al.* have conceived "*logic programming*" in very general terms (starting with McCarthy's Advice Taker proposal) as "*what can be programmed in mathematical logic*."   Of course, what can be programmed in mathematical logic is exactly "*the logical inference of computational steps."*

Even with this general definition and even allowing the full power of Direct Logic, computation is not reducible to Logic Programming [Hewitt 2008b].

Nevertheless, Logic Programming provides an important contribution to computing with its own valuable properties. Recently Hassan Aït-Kaci has pointed out:

> *It would be like saying Prolog and SLD-Resolution is the only way to do Logic Programming. To some extent, the LP* [Logic Programming] *community's insistence on clinging to this "exclusive method" has contributed to the relative disinterest in LP following its development in the 1980's and 1990's.* [Aït-Kaci 2009]

If the more general definition of Logic Programming (*i.e.* "*the logical inference of computational steps*") is accepted, then the field has a very long history with contributions by Bruce Anderson, Bruce Baumgart, Fischer Black, Eugene Charniak, Alonzo Church, Alain Colmerauer, Julian Davies, Jan Derksen, Ted Elcock*,* Scott Fahlman, Richard Fikes, Frederick Fitch, Gerhard Gentzen, Cordell Green, Pat Hayes, Carl Hewitt, Robert Kowalski, Bill Kornfeld, Drew McDermott, Zohar Manna, John McCarthy, Nils Nilsson, Alan Robinson*,* Philippe Roussel, Jeff Rulifson, Earl Sacerdoti, Erik Sandewall, Gerry Sussman, Richard Waldinger, Terry Winograd, *etc.*[xi]

Keith Clark, Alain Colmerauer, Pat Hayes, Robert Kowalski, Alan Robinson, Philippe Roussel, etc. deserve a lot of credit for promoting the concept of logic programming and helping to build the logic programming community. And the traditions of this community should not be disrespected. At the same time, the term "logic programming" (like "functional programming") is highly descriptive and should mean something. Over the course of history, the term "functional programming" has grown more precise and technical as the field has matured. Logic Programming should be on a similar trajectory. Accordingly, "*Logic Programming*" should have a more precise characterization, *e.g.,* "*the logical inference of computational steps*."

Kowalski's approach serves to avoid conclusions that are considered by some proponents to be unflattering to Logic Programming, *e.g.,* the conclusion that Logic Programming is not computationally universal (see section above). However, the fact that Functional Programming is also not computationally universal is not considered to be unflattering.

Also, Kowalski has pursued an *advocacy* approach towards logic programming. Recently he remarked:

> *Admittedly, I have been messianic in my advocacy of Logic, and I make no apologies for it. Pushing Logic as hard as I could has been my way of trying to discover its limits.* [Kowalski 2006]

In contrast, Hewitt and his colleagues have pursued an *exploratory* approach to the limits of Logic.

In summary, Kowalski's approach was to advocate Logic Programming in terms of the traditions of a community centered around the Association of Logic Programming building on the resolution uniform proof procedure paradigm for proving theorems. **In contrast, the Hewitt *et. al.* approach is to reject the resolution uniform proof procedure paradigm and to explore**

Logic Programming defined by a principled criterion, namely, "*the logical inference of computational steps*".

Subsequently Kowalski has championed, not only logic programming, but also computational logic, and logic-based agents more generally. His 1979 book "Logic for Problem Solving" advocated this more general use of logic, highlighted the role that logic can serve in open systems, and even eluded to the useful role of inconsistency. More recently, he has argued that logic programming and computational logic are too limited, both as a basis for Artificial Intelligence and for computing more generally. [Kowalski 2009]

In our work, we have identified concurrency as a reason why Logic Programming is not universal. In contrast Kowalski has identified the need to simulate semantic structures, which change destructively, concurrently and perhaps spontaneously.[xii] He believes that the need to simulate semantic structures is compatible with logic as a language for computation and sees the need to augment inference with an appropriate model theory. **Kowalski's approach contrasts with our own work on Direct Logic [Hewitt 2008b] in which we have moved towards proof theory and away from Tarskian set-theoretic model theory [Tarski and Vaught 1957 a as foundation for Logic Programming.**

Acknowledgment

Scott Fahlman made important suggestions and comments that materially improved this article. Richard Waldinger made an important suggestion for improving the definition of Logic Programming and otherwise greatly helped to improve the paper. Jeremy Forth made helpful suggestions. Bob Kowalski and Pat Hayes made important contributions and suggestions that greatly improved the article. I especially thank Bob for sharing his view of his recent research. Of course, any remaining errors are entirely my own.

References

- Hal Abelson and Gerry Sussman *Structure and Interpretation of Computer Programs (2nd edition)* MIT Press. 1996.
- Robert Anderson and Woody Bledsoe (1970) "A Linear Format for Resolution with Merging and a New Technique for Establishing Completeness" *JACM* 17.
- Hassan Aït-Kaci. *Children's Magic Won't Deliver the Semantic Web* CACM. March 2009
- Bruce Anderson. *Documentation for LIB PICO-PLANNER* School of Artificial Intelligence, Edinburgh University. 1972.
- Bruce Anderson and Pat Hayes. *The logician's folly* DCL Memo 54. University of Edinburgh. 1972.
- Bruce Baumgart. *Micro-Planner Alternate Reference Manual* Stanford AI Lab Operating Note No. 67, April 1972.
- L. Bertossi and J. Chomicki. *Query answering in inconsistent databases* Logics for Emerging Applications of Databases. 2003.

- Maurice Bruynooghe, Luís Moniz Pereira, Jörg Siekmann, Maarten van Emden. *A Portrait of a Scientist as a Computational Logician* Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part I Springer. 2004.
- Eugene Charniak *Toward a Model of Children's Story Comprehension* MIT AI TR-266. December 1972.
- Alonzo Church *A Set of postulates for the foundation of logic* Annals of Mathematics. Vol. 33, 1932. Vol. 34, 1933.
- Paolo Ciancarini and Giorgio Levi. *What is Logic Programming good for in Software Engineering?* International Conference on Software Engineering. 1993.
- Will Clinger. *Foundations of Actor Semantics* MIT Mathematics Doctoral Dissertation. June 1981
- Jacques Cohen. *A view of the origins and development of Prolog* CACM. January 1988.
- Alain Colmerauer and Philippe Roussel. *The birth of Prolog* History of Programming Languages. ACM Press. 1996.
- Ole-Johan Dahl and Kristen Nygaard. *Class and subclass declarations* IFIP TC2 Conference on Simulation Programming Languages. May 1967.
- Julian Davies. *Popler 1.6 Reference Manual* University of Edinburgh, TPU Report No. 1, May 1973.
- Julian Davies. *Representing Negation in a Planner System AISB'74*.
- Adriann De Groot *Thought and choice in Chess* Mouton De Gruyter. 1965.
- Maarten van Emden. *The Early Days of Logic Programming: A Personal Perspective* The Association of Logic Programming Newsletter. August 2006.
- Scott Fahlman. *A Planning System for Robot Construction Tasks* MIT AI TR-283. June 1973.
- Frederic Fitch *Symbolic Logic: an Introduction* Ronald Press, New York, 1952.
- J.M. Foster and E.W. Elcock. *ABSYS: An Incremental Compiler for Assertions* Machine Intelligence 4. Edinburgh University Press. 1969.
- Kazuhiro Fuchi, Robert Kowalski, Kazunori Ueda, Ken Kahn, Takashi Chikayama, and Evan Tick. *Launching the new era* CACM. 1993
- Cordell Green. *Application of Theorem Proving to Problem Solving* IJCAI'69.
- Steve Gregory. *Concurrent Logic Programming Before ICOT: A Personal Perspective* August 15, 2007.
- Pat Hayes *Computation and Deduction* Mathematical Foundations of Computer Science: Proceedings of Symposium and Summer School, Štrbské Pleso, High Tatras, Czechoslovakia, September 3-8, 1973.
- Pat Hayes. *Semantic Trees* Ph.D. thesis. Edinburgh University. 1973.
- Pat Hayes *Some Problems and Non-Problems in Representation Theory* AISB'74.
- Irene Greif. *Semantics of Communicating Parallel Professes* MIT EECS Doctoral Dissertation. August 1975.
- Carl Hewitt *PLANNER: A Language for Proving Theorems in Robots* IJCAI'69.
- Carl Hewitt *Procedural Embedding of Knowledge In Planner* IJCAI 1971.
- Carl Hewitt *Description and Theoretical Analysis (Using Schemata) of Planner, A Language for Proving Theorems and Manipulating Models in a Robot* AI Memo No. 251, MIT Project MAC, April 1972.
- Carl Hewitt, Peter Bishop and Richard Steiger. *A Universal Modular Actor Formalism for Artificial Intelligence* IJCAI 1973.

- Carl Hewitt *Stereotypes as an Actor Approach Towards Solving the Problem of Procedural Attachment in Frame Theories* Theoretical Issues In Natural Language Processing. 1975.
- Carl Hewitt and Henry Baker *Actors and Continuous Functionals* Proceeding of IFIP Working Conference on Formal Description of Programming Concepts. August 1-5, 1977.
- Carl Hewitt *The Challenge of Open Systems* Byte Magazine. April 1985.
- Carl Hewitt and Jeff Inman. *DAI Betwixt and Between: From 'Intelligent Agents' to Open Systems Science* IEEE Transactions on Systems, Man, and Cybernetics. Nov. /Dec. 1991.
- Carl Hewitt and Gul Agha. *Guarded Horn clause languages: are they deductive and Logical?* International Conference on Fifth Generation Computer Systems, Ohmsha 1988. Tokyo. Also in Artificial Intelligence at MIT, Vol. 2. MIT Press 1991
- Carl Hewitt (2006b) *What is Commitment? Physical, Organizational, and Social* COIN@AAMAS'06.
- Carl Hewitt (2008a) *A historical perspective on developing foundations for client cloud computing* (revised version of Development of Logic Programming: What went wrong, What was done about it, and What it might mean for the future *Proceedings of What Went Wrong and Why: Lessons from AI Research and Applications* edited by Mehmet Goker and Daniel Shapiro. AAAI Press. AAAI'08.) Google Knol. http://perspective.carlhewitt.info
- Carl Hewitt (2008b), *Common sense for concurrency and inconsistency tolerance using Direct Logic*^{TM} ArXiv 0907.3330
- Carl Hewitt *ActorScript*^{TM}*: Industrial strength integration of local and nonlocal concurrency for Client-cloud Computing* ArXiv 0907.3330. 2009.
- Matthew Huntbach and Graem Ringwood. *Agent-Oriented Programming: From Prolog to Guarded Definite Clauses* Springer. 1999.
- Daniel Ingalls. *The Evolution of the Smalltalk Virtual Machine* Smalltalk-80: Bits of History, Words of Advice. Addison Wesley. 1983.
- Michael Kassoff, Lee-Ming Zen, Ankit Garg, and Michael Genesereth. *PrediCalc: A Logical Spreadsheet Management System* 31st International Conference on Very Large Databases (VLDB). 2005.
- Stephen Kleene and John Barkley Rosser *The inconsistency of certain formal logics* Annals of Mathematics Vol. 36. 1935.
- William Kornfeld and Carl Hewitt *The Scientific Community Metaphor* MIT AI Memo 641. January 1981.
- William Kornfeld (1981a) *The Use of Parallelism to Implement a Heuristic Search* IJCAI'81.
- William Kornfeld (1981b) *Parallelism in Problem Solving* MIT EECS Doctoral Dissertation. August 1981.
- William Kornfeld. *Combinatorially Implosive Algorithms* CACM. 1982.
- Robert Kowalski and Pat Hayes. *Semantic trees in automatic theorem-proving* Machine Intelligence 4. Edinburgh Press. 1969.
- Robert Kowalski *Predicate Logic as Programming Language* IFIP'74.
- Robert Kowalski *Logic for problem-solving* DCL Memo 75. Dept. of Artificial Intelligence. Edinburgh. 1974.
- Robert Kowalski *Algorithm = Logic + Control* CACM. July 1979.
- Robert Kowalski. *Response to questionnaire* Special Issue on Knowledge Representation. SIGART Newsletter. February 1980.
- Robert Kowalski *The Limitations of Logic* Proceedings of the ACM Annual Conference on Computer Science. 1986.

- Robert Kowalski (1988a) *The Early Years of Logic Programming* CACM. January 1988.
- Robert Kowalski (1988b) *Logic-based Open Systems* Representation and Reasoning. Stuttgart Conference Workshop on Discourse Representation, Dialogue tableaux and Logic Programming. 1988.
- Robert Kowalski. *Logic Programming* MIT Encyclopedia of Cognitive Science. MIT Press. 1999.
- Robert Kowalski. *Logic Programming and the Real World* Logic Programming Newsletter. January 2001.
- Robert Kowalski (2004a) *History of the Association of Logic Programming* October 2004.
- Robert Kowalski (2004b) *Directions for Logic Programming* Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part I Springer. 2004.
- Robert Kowalski *Re: Which Logicists?* Email to Carl Hewitt, Pat Hayes, Michael Genesereth, Richard Waldinger, and Mike Dunn. December 20, 2006.
- Robert Kowalski "Reversion by Ruud Koot" in *Talk: Logic Programming* on Wikipedia. April 4, 2007.
- Robert Kowalski "Reasoning with Conditionals in Artificial Intelligence" *The Psychology of Conditionals* Oxford University Press. 2008.
- Robert Kowalski *Email to Carl Hewitt* April 2009.
- Peter Landin *A Generalization of Jumps and Labels Report* UNIVAC Systems Programming Research. August 1965. Reprinted in Higher Order and Symbolic Computation. 1998.
- Bruno Latour *Science in Action: How to Follow Scientists and Engineers through Society* Harvard University Press. 1988.
- James Lighthill *Artificial Intelligence: A General Survey* Artificial Intelligence: a paper symposium. UK Science Research Council. 1973.
- Donald MacKenzie *Mechanizing Proof* MIT Press. 2001.
- E. Mayol and E. Teniente. *A survey of current methods for integrity constraint maintenance and view updating* ER Workshops. 1999.
- John McCarthy *Programs with common sense* Symposium on Mechanization of Thought Processes. National Physical Laboratory, UK. Teddington, England. 1958.
- John McCarthy, Paul Abrahams, Daniel Edwards, Timothy Hart, and Michael Levin. *Lisp 1.5 Programmer's Manual* MIT Computation Center and Research Laboratory of Electronics. 1962.
- John McCarthy *Review of Artificial Intelligence: A General Survey* Artificial Intelligence: a paper symposium. UK Science Research Council. 1973.
- John McCarthy. *Sterile Containers* *www.ai.sri.com/~rkf/designdoc/sterile.ps* September 8, 2000.
- L. Thorne McCarty. *Reflections on TAXMAN: An Experiment on Artificial Intelligence and Legal Reasoning* Harvard Law Review. Vol. 90, No. 5, March 1977.
- Drew McDermott and Gerry Sussman *The Conniver Reference Manual* MIT AI Memo 259A. January 1974.
- Drew McDermott *The Prolog Phenomenon* ACM SIGART Bulletin. Issue 72. July, 1980.
- Marvin Minsky (ed.) *Semantic Information Processing* MIT Press. 1968.
- Marvin Minsky and Seymour Paper. *Progress Report, Artificial Intelligence MIT* AI Memo 252. 1972.
- Allen Newell and Herbert Simon *The logic theory machine: A complex information processing system* IRE Trans. Information Theory IT-2:61-79. 1956.

- Nils Nilsson *Artificial Intelligence: A New Synthesis* San Francisco: Morgan Kaufmann, 1998.
- L. Orman. *Transaction repair for integrity enforcement* TKDE, 13(6). 2001.
- George Polya (1957) *Mathematical Discovery: On Understanding, Learning and Teaching Problem Solving Combined Edition* Wiley. 1981.
- Karl Popper (1935, 1963) *Conjectures and Refutations: The Growth of Scientific Knowledge* Routledge. 2002.
- John Alan Robinson *A Machine-Oriented Logic Based on the Resolution Principle.* CACM. 1965.
- Jeff Rulifson, Jan Derksen, and Richard Waldinger *QA4, A Procedural Calculus for Intuitive Reasoning* SRI AI Center Technical Note 73, November 1973.
- Mike Paterson and Carl Hewitt. *Comparative Schematology* MIT AI Memo 201. August 1970.
- Philippe Rouchy. *Aspects of PROLOG History: Logic Programming and Professional Dynamics* TeamEthno-Online Issue 2. June 2006.
- Earl Sacerdoti, *et al., QLISP A Language for the Interactive Development of Complex Systems* AFIPS. 1976.
- Erik Sandewall. *From Systems to Logic in the Early Development of Nonmonotonic Reasoning* CAISOR. July, 2006.
- Gary Saxonhouse *What's All This about Japanese Technology Policy?* Cato Institute. August 17, 2001
- Marek Sergot. *Bob Kowalski: A Portrait* Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part I Springer. 2004.
- Jon Sigurdson *Industry and state partnership: The historical role of the engineering research associations in Japan* 1986
- Ehud Shapiro *The family of concurrent logic programming languages* ACM Computing Surveys. September 1989.
- Gerry Sussman, Terry Winograd and Eugene Charniak *Micro-Planner Reference Manual (Update)* AI Memo 203A, MIT AI Lab, December 1971.
- Gerry Sussman and Guy Steele *Scheme: An Interpreter for Extended Lambda Calculus* MIT AI Lab Memo 349. December 1975.
- Shunichi Uchida and Kazuhiro Fuchi *Proceedings of the FGCS Project Evaluation Workshop* Institute for New Generation Computer Technology (ICOT). 1992.
- Martin van Emden and Robert Kowalski. *The semantics of predicate logic as a programming language* Edinburgh TR 1974. JACM'76.
- Alfred Tarski and Robert Vaught (1957). "Arithmetical extensions of relational systems" *Compositio Mathematica* 13.
- Terry Winograd *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language* MIT AI TR-235. January 1971.

End Notes

i Being a hybrid language, Micro Planner had two different syntaxes, variable binding mechanisms, *etc.* So it lacked a certain degree of elegance. In fact, after Hewitt's lecture at IJCAI'71, Allen Newell rose from the audience to remark on the lack of elegance in the language! However, variants of this syntax have persisted to the present day.

ii One implementation decision in Micro Planner had unfortunate consequences. Lisp had adopted the programming pun of identifying NIL, the empty list with logical false (at memory location 0) because testing for 0 was faster than anything else. Because of the pun, testing for NIL was extremely common in Lisp programs. The implementers of Micro Planner extended this pun also to use NIL as a signal to begin backtracking. In Micro Planner, it was common to write programs to perform some operation on every element of a list by using a loop to process the first element of a list, take the rest of the list, and then jump back to the top of the loop to test if the list was empty. If the list tested empty, then the program would go on to do other things. Such a program never made it to testing the empty list after processing all the elements because when the last element was processed and the rest of the list was taken, NIL was returned as a value. The Micro Planner interpreter took this as the signal to begin backtracking and began undoing all the work of processing the elements of the list! People were dumbfounded. [Fahlman 1973]

iii Sussman and Steele [1975] misunderstood Actors and mistakenly concluded "*we discovered that the 'Actors' and the lambda expressions were identical in implementation.*" The actual situation is that the lambda calculus is capable of expressing some kinds of sequential and parallel control structures but, in general, not the concurrency expressed in the Actor model. On the other hand, the Actor model is capable of expressing everything in the lambda calculus and more.

Sussman and Steele noticed some similarities between Actor customers and continuations introduced by [Reynolds 1972] using a primitive called escape that was a further development of hairy control structure. In their programming language Scheme, they called their variant of escape by the name "*call with current continuation.*"   Unfortunately, general use of escape is not compatible with usual hardware stack disciple introducing considerable operational inefficiency. Also, using escape can leave customers stranded. Consequently, use of escape is generally avoided these days and exceptions are used instead so that clean up can be performed. [Hewitt 2009]

iv Hewitt [1976].

v First published in IJCAI-73.

vi There was somewhat similar work that Hayes had discussed with the researchers at Aberdeen on ABSYS/ABSET [Foster and Elcock 1969].

vii the Golux is a character in a Thurber fairytale who declared "I am the Golux, and not a mere device."

viii Resolution makes use of proof by contradiction: In order to prove a goal G, $\neg G$ (the negation of G) is placed in the data base and a contradiction is derived. However, resolution can easily prove any false proposition for from an inconsistency. For example, suppose that there is a simple inconsistent data base with just P and $\neg P$. To prove the false proposition TheMoonIsMadeOfCheese, first prove the lemma (TheMoonIsMadeOfCheese $\vee$ P), which is easily done. Now with the lemma in the data base, it is easy to prove TheMoonIsMadeOfCheese.

ix Abelson and Sussman [1996] provided their version of the history as follows:

> *Logic programming has grown out of a long history of research in automatic theorem proving. Early theorem-proving programs could accomplish very little, because they exhaustively searched the space of possible proofs. The major breakthrough that made such a search plausible was the discovery in the early 1960s of the* unification algorithm *and the* resolution priciple *(Robinson 1965). Resolution was used, for example, by Green and Raphael (1968) (see also Green 1969) as the basis for a deductive question-answering system. During most of this period, researchers concentrated on algorithms that are guaranteed to find a proof if one exists. Such algorithms were difficult to control and to direct toward a proof. Hewitt (1969) recognized the possibility of merging the control structure of a programming language with the operations of a logic-manipulation system…. At the same time that this was being done, Colmerauer, in Marseille, was developing rule-based systems for manipulating natural language (see Colmerauer et al. 1973). He invented a programming language called Prolog for representing those rules. Kowalski (1973; 1979), in Edinburgh, recognized that execution of a Prolog program could be interpreted as proving theorems (using a proof technique called linear Horn-clause resolution). The merging of the last two strands led to the logic-programming movement. Thus, in assigning credit for the development of logic programming, the French can point to Prolog's genesis at the University of Marseille, while the British can highlight the work at the University of Edinburgh. According to people at MIT, logic programming was developed by these groups in an attempt to figure out what Hewitt was talking about in his brilliant but impenetrable Ph.D. thesis. For a history of logic programming see Robinson 1983.*

x In justification, Kowalski [2006] later said:

> *I think that Pat* [Hayes] *and I had in mind the Church-Turing notion of computability, in which such diverse models of computation as recursion equations, production systems, Turing Machines, and lambda calculus have all proved to be equivalent. Horn clause logic is sufficient for computability in this sense.*

xi It might seem that this is not so different from the [Kowalski 1999] definition for the MIT Encyclopedia of Cognitive Science:

> *Logic Programming is the use of logic to represent programs and of deduction to execute programs in logical form. To this end, many different forms of logic and many varieties of deduction have been investigated.*

xii However these changes cannot be implemented using Logic Programming as defined in this article.