A Semantics for Concurrent Logic Programming Languages Based on Multiple- Valued Logic

Marion Glazerman Ben-Jacob
Department of Mathematics and Computer Information
Science Mercy College
Dobbs Ferry, New York, USA

***Abstract*— In order to obtain an understanding of parallel logic thought it is necessary to establish a fully abstract model of the denotational semantics of logic programming languages. In this paper, a fixed point semantics for the committed choice, non-deterministic family of parallel programming languages, i.e. the concurrent logic programming languages is developed. The approach is from an order theoretic viewpoint. We rigorously define a semantics for a Guarded Horn Clauses-type of language because of the minimal restrictions of the language. The extension to other concurrent logic programming languages would be direct and analogous, based on their specific rules of suspension. Today's world is replete with multitasking and parallelism in general. The content of this paper reflects a paradigm of an application of multi-valued logic which is reflective of this.**

Keywords- concurrent logic programming; multiple-valued logic; denotational semantics.

I. INTRODUCTION

Parallelism in technology encourages us to examine the meaning of different logical operations being done concurrently. A question of reasonable complexity is how would one attempt to solve more than one quantitative problem at the same time or attempt to make more than one logical inference concurrently. We need to give meaning to the programs that are written in languages that possess the capability of concurrency.

In the case of implementing a logic program via resolution, it appears that not only will parallelism increase efficiency but that the underlying inference procedure actually lends itself in a natural way toward concurrency; for any selected clause, many different instantiations of a selected atom might be attempted at once, more than one atom might be chosen for possible resolution, and any give atom might even select several different clauses that contain an atom against with which it might be resolved. With all these non-deterministic possibilities, obviously, concurrency controls must also be guaranteed by parallel programming languages.

Given that sound results can be obtained, we need to specifically understand how logical inferences are made by machines that support parallelism. A fundamental goal becomes the understanding of parallel logic.

First order logic does not provide us with a significantly sophisticated basis for interpreting parallel logic thought. Perhaps, Kleene's [19] three-valued logic or Belnap's [2] four-valued logic is more appropriate. Upon closer inspection the situation is more involved than just several processors working independently. A reasonable model of what happens when pertinent information is spread over a number of sites that communicate with each other was investigated by Fitting [10, 11]. It was based on Belnap's four-valued logic, exhibited by a bi-lattice structure.

Existing results credit a greater complexity to parallel logic thinking than information just being distributed over a number of sites. Parallelism, accounting for hardware and supporting language, allows for shared memory, sometimes somewhat restricted, sometime global, and interaction among the processors prior to the final assignment of truth values; thus, a fixed point semantics based on a simple bi-lattice structure no longer seems adequate.

In order to obtain an understanding of parallel logic thought and the role played by classical logic, we need to establish a fully abstract model of the denotational semantics of logic programming languages [22]. In addition to its other merits, this type of model can serve as a theoretical foundation for debugging parallel logic programs.

In pursuit of the understanding of parallel logic programming, an analysis of the operational semantics of Concurrent Prolog and the Concurrent Constraint Programming (CCP) family of languages has been made by Saraswat [24]. Kok [20] has developed a purely topological model for the denotational semantics of Concurrent Prolog, and Gerth, Codish, Lichtenstein and Shapiro [14] have developed one for Concurrent Prolog based on sets of suspensions. We will examine the semantics of a parallel logic programming language from an order theoretic viewpoint.

In this paper fixed point semantics for the committed choice, non-deterministic and parallel (CCNAP) family of parallel programming languages, i.e. the concurrent logic programming languages, is developed. We start with GHC, Guarded Horn Clauses, and the simplest of the parallel logic programming languages.

GHC, with its minimal restrictions, few rules of suspension and global environments is an appropriate starting place for understanding how logical implications are formed in parallel. We assume the version of GHC which we are considering allows for failure; thus, it might perhaps be more accurate to say we are considering a GHC-type language. We rigorously define semantics for a parallel programming language. We

build this definition with the operational semantics of GHC in mind and so, this semantics will describe a formalized GHC. The extension to the other concurrent logic programming languages would be natural and based on the language-dependent suspension rules.

II. SYNTAX

The syntax of GHC and more generally, that of any member of the family of CCNAP languages is based on the syntax of the sequential logic programming languages. Let L be a first-order language over a non-empty domain D. A term is either a variable, or a constant or a function of terms, i.e. an element belonging to the domain D. If P( ) is an n-place predicate symbol and t_i , $1 \leq i \leq n$ are terms of L(D) then P($t_1,\ldots,t_n$) is an atom or atomic formula. A literal of L(D) is an atom or the negation of an atom belonging to L(D).

An expression of the form H: $-G_1 \ldots G_k \mid B_1 \ldots B_n$ (k, n>0) is a guarded program clause. H is called the clause head, the G_j's are guard goals and the B_i's are body goals. H, the G_j's and the B_i's are all atomic formulae. The commitment operator, $\mid$, is usually interpreted as conjunction and separates the clause's guard from its body, the former being written to the left of the operator and including the head, and the latter the right. The guard of a non-goal clause is never empty. The limiting case for GHC is when the predicate is the system predicate true, i.e. the clause if of the form h(x): - true $\mid$ b(x). Goal clauses are of the form : - $B_1 \ldots B_n$. A CCNAP program is a finite set of guarded program clauses.

III. SEMANTICS

Before giving formal definitions, we will attempt to provide the motivation for our choice of a five-valued logic and for the actual truth values chosen.

We want our denotational semantics to align with the operational semantics of the CCNAP family of languages as closely as possible, and so, we concluded that we need a four-valued logic, the truth values being true (t), false (f), undefined ($\perp$), and suspend (s). The fifth value, overdefined ($\top$), is included for topological facility.

The necessity of the first two truth values in our logic is obvious. Suspend is required because of the nature of the parallel programming languages. All the concurrent logic languages experience suspension of processing as a result of specific language dependent occurrences. According to Ueda [27], GHC's rules of suspension are:

"(a) Unification invoked directly or indirectly in the guard of a clause C called by a goal G cannot instantiate the goal G.

(b) Unification invoked directly or indirectly in the body of a clause C cannot instantiate the guard of C until that clause is selected for commitment.

A piece of unification that can succeed only by making such bindings is suspended until it can succeed without making such bindings."

The truth value suspend reflects the fact that work has been attempted to establish the truth value of the instantiated predicate in question, no (exact) precise truth valued has yet been assigned and at this point in time work must be stopped and recorded (suspended) so as not to interfere with the validity of the calculations of the other processors. Also, should there be a malfunction in the hardware allowing for one instantiated predicate to be assigned true by one processor and false by another, we will say the predicate has the truth value suspend, indicating that some work has been done on it. We are assuming all processor are working with all clauses.

After careful consideration, it will become clear that $\perp$ belongs in the scheme. We will be determining truth value assignments based on the operations of several processors and our intuition leads to the naturalness of assigning $\perp$ to an instantiated predicate in the following cases: if a processor has not even begun dealing with its values yet; if more "work" is needed before assigning it a truth value and this work can proceed without interfering or contradicting the operation of the other processors.

More formally, along the lines of Fitting and Ben-Jacob [12, 13] we get

Definition 1

FIVE is the space of truth values $\{\top, t, f, suspend, \perp\}$ with the ordering $<_5$ where $\perp <_5$ suspend $<_5$ f $<_5 \top$ and $\perp <_5$ suspend $<_5$ t $<_5 \top$. Figure 1 illustrates the ordering pictorially.

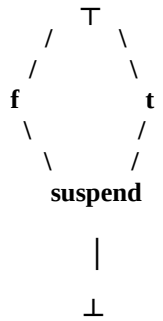

Figure 1

Clearly, the ordering $<_5$ is based on the amount of information or knowledge available. We note the existence of an alternative ordering $<_5^*$ where f $<_5^*$ suspend $<_5^*$ t,

f $<_5^* \top <_5^*$ t and f $<_5^* \perp <_5^*$ t. The ordering $<_5^*$ is based on the amount of truth available.

Definition 2

A five-valued interpretation is a mapping V from ground (variable-free) atomic formulae of L to FIVE. Obviously, in general, an interpretation can be trivially single-valued or many-valued, not necessarily five-valued. For our purposes, we need only consider four-valued interpretations.

Our interpretations are given the point wise ordering based on ground atomic formulae. The space FIVE is a complete lattice and by a generalization of the Knaster-Tarski Theorem, must have a least fixed point and a greatest fixed point [17].

The extension of interpretations from atomic formulae, e.g. A and B, to all closed formulae is governed by the truth tables, Table I and Table II.

TABLE I

A	B	A v B	A∩B
true	false	t	f
	true	t	t
	suspend	t	suspend
	⊥	t	⊥
false	false	f	f
	suspend	suspend	f
	⊥	⊥	f
suspend	suspend	suspend	suspend
	⊥	⊥	⊥
⊥	⊥	⊥	⊥

TABLE II

A	~A
true	false
false	true
suspend	suspend
⊥	⊥

With regard to Table I we note that both AvB and A∩B are monotonic with regard to the knowledge ordering $<_5$ of FIVE. Other patterns that exist include that A∩B takes the g.l.b. of its values with regard to the truth ordering $<_5^*$. A∩B takes the g.l.b. of its values with regard to the knowledge ordering if we disregard the argument being equal to false and lastly, in the knowledge ordering, if one of the arguments is false. A∩B always takes on the l.u.b. of the two values. Also, AvB takes on the l.u.b. of its values with regard to the truth ordering.

Given a clause of the generic type h ← g | b , by definition the guard includes those predicates to the left of the commit operator, including the head. By headless guard we mean only those predicates to the left of the commit operator on the right side of the implication arrow, i.e. the guard without the head of the clause. Consider the following truth table:

TABLE III

headless guard (g)	body b	g | b
t	t suspend f	t suspend f
suspend	t suspend f	suspend suspend
f	t suspend f	f f
⊥	t ⊥	⊥
	f	f
⊥	suspend	suspend

We clarify (truth) Table III by noting if the guard g is false and the body b of a clause suspends, g | b is false since it cannot be used in a proof that ascertains the truth value of a predicate. Also, according to Ueda [27], GHC was designed that with the two given rules of suspension (see page 7), anything can be done in parallel or even executed in a predetermined order provided the latter constraint does not change the meaning of the program; thus, we must allow the possibility of the truth values that appear in the last line of Table III. It is conceivable in theory that the body of a clause could get instantiated prior to the instantiation of the guard and the commitment and that this is the clause to be used in an attempted resolution process. If this takes place, the truth value of the body would be suspend. In this case, if a body, b, suspends its corresponding guard will never be instantiated; thus, the guard of this clause is undefined. Reiterating, if the situation is such that the instantiation of a predicate in the body forces the instantiation of a variable in the guard prior to commitment the truth value of the body would be suspend and so, the guard would never get assigned a truth value. We note that g | b is monotonic. We now define a conditional truth table for the natural interpretation of h ← g | b where g | b is of the form $g_1,\ldots,g_k \mid b_1,\ldots,b_n$ (k,n>0).

Table IV

g | b	h	h← g | b	Annotations
t	t	t	
f	t	t	
suspend	t	t	
⊥	t	t	
t	f	f	
f	f	t	
suspend	f	suspend	Suspended work on g | b caused lack of knowledge of outcome; so, h← g | b has value suspend.
⊥	f	⊥	Insufficient amount of information caused lack of knowledge of outcome, so h← g | b has value
t	suspend	suspend	
f	suspend	suspend	h← g | b takes on suspend to reflect some work was done but not enough to determine a final truth value.
suspend	suspend	suspend	Work has been done on g | b and h, but no value can be determined.
⊥	suspend	suspend	Reflects that work has been done but no conclusion has been reached.
t	⊥	⊥	It is the truth value of h that is responsible for the lack of a determined truth value of the entire clause.
f	⊥	⊥	
⊥	⊥	⊥	

After examining Table IV, we note the monotonicity with regard to knowledge of h←g | b. We also remark on the deviation from classical logic, e.g. a→b ≠ a v b as f → ⊥ is ⊥ and ¬ f v ⊥ = t v ⊥ is true.

Obviously, if the formula is of the form ∃ x A (*there exists an x such that A is true),* its truth value is determined by the truth tables and the relationship ∃ x A = $v_{x\varepsilon D}$ A. Similarly, if the formula is of the form *for- every-x A is true,* its truth value depends on the tables and the relationship *for- every-x A is true* =$\cap_{x\varepsilon D}$ A.

Next we will consider the assignment of truth values for heads of clauses in a logic program, first with regard to an individual processor and then, with respect to several processors working concurrently.

When we write h←g | b we will be using this shorthand notation to mean h(x)←g(x) | b(x). If we want to resolve clauses of the form $h_1(x)$←$g_1(x)$ | $b_1(x)$ and $h_2(y)$←$g_2(y)$ | $h_1(y)$ we would of course first have to unify the clauses. Using our shorthand notation, we would represent these two clauses by h_1←g_1 | b_1 and h_2←g_2 | h_1'.

Let p_1←g_1 | b_1 be the clause whose head predicate p_1 is attempting to unify with the goal clause via a unifier θ_0. Let p_2← g_2 | b_2 be the clause whose head predicate p_2 is attempting to unify with g_1 via a unifier θ_1 to solve g_1. In general, let θ_n be the unifier needed to unify guard g_n with head predicate p_{n+1} to solve the guard g_n. The process is nested j+1 levels until the guard of the j+1st clause is "true."

Definition 3

We will say p and p_i are unifiable if the composition of unifiers $\theta j \cdot \ldots \theta_1 \cdot \theta_0$ does not cause a suspension. The most general unifier will be the composition of most general unifers that do not cause a suspension. Unifiers that do not cause a suspension under composition will be said to be compatible.

Theorem 1

The rigorously defined parallel programming languages that has been defined on the previous pages correctly reflects the operational semantics of GHC.

Proof: Previous discussion, truth tables I-IV, and definitions 2-3.

Definition 4

Let P be a logic program defined over a domain D. A reserved relation symbol is a symbol that represents a given relation on the domain D, i.e. a given mapping from D to the appropriate or relevant space of truth values. Φ_{ip} will be the map on non-parallel interpretations by a given processor i, given by the following: Assume $\Phi_{ip}(V)$ = W. W is the interpretation such that

(i) if R is a reserved relation symbol, W(R(a)) = R(a).

For a non-reserved relation symbol h,

(ii) if ∃ a program clause in P(D) whose head is h(a) and whose guarded body g(a) | b(a) maps to true under V. then W(h(a)) = true.

(iii) if all clauses in P(D) of which h(a) is the head, have guarded bodies that V maps to false, then W(h(a)) = false.

(iv) If at least one clause in P(D) whose head is ha(a) and whose corresponding guarded body V maps to suspend while all other clauses of the form h(a)← $g_i(a)$ | $b_i(a)$ in P(D) have guarded bodies that V maps to false,
W(h(a)) = suspend.

(v) if all clauses in P(D) whose head is h(a) have bodies that V maps to suspend, then W(h(a)) = suspend.

(vi) in all other cases W(h(a)) = ⊥, i.e. the guarded bodies of the clauses in P(D) of which h(a) is the head are such that for at least one, V maps the guarded body to ⊥ and none are mapped to true by V, the W(h(a)) = ⊥.

To gain more comfortable with the previous definition, let us examine its effect on ground programs.

1. Since there does not exist any instantiation of variables, suspension does not play any role as a truth value.
2. Once the headless guard of a clause is true, the interpretation of the clause is equivalent to its "unguarded version." Consider the program

q←
p ←q | r

In this case, V(r) = ⊥, and if Φ(V) = W, then W(p) = ⊥. This program behaves like p←r. Consider the comparison of the following two programs, assuming negation in the body of a clause is allowed.

r←	r←
q←	q←
p←q | r	p←q | ¬ r

In the first program, W(p) = t and in the second program, W(p) = false.

1. If a headless guard is false, V(guarded body) = false; it is irrelevant which truth value V(body) takes on; thus, W(unguarded head) = false.
2. If V(guard) = ⊥, then the truth value of the body is of concern. Consider the following three programs:

h←g | b	b←	b←
	h←g | b	h←g | ¬ b

For the first program V(g | b) = ⊥ and W(h) = ⊥. For the second and third programs, W(h) = ⊥ and W(h) = f, respectively.

The aforementioned conclusions are based on (Truth) Table III. The definition of Φ_p is satisfactory when one processor is trying to interpret the true meaning of a program clause or several processors are concurrently determining the interpretation of one program clause. With parallel logic programming we must account for more than one

interpretation of a clause or a predicate being worked on concurrently, as well.

Definition 5

Let $p_i(a)$ denote the i^{th} interpretation of the instantiated predicate p(a). Then the concept of parallel interpretations is defined in a binary manner and $p_i(a)$ @ $p_j(a)$, the interpretation based on two, concurrent interpretations is given by Table V.

Table V			
$p_i(a)$	$p_i(a)$	$p_i(a)$ @ $p_i(a)$	Annotations
t	t	t	
t	suspend	t	One processor is forced to suspend but the other processor proved the predicate true.
t	f	$\top$	Overdefined- possibly by hardware malfunction; this is the only occurrence of $\top$.
t	$\perp$	t	One processor proved the predicate true; the other one may not even have examined $p_i(a)$.
f	suspend	f	The assumption is that one processor chose the wrong clauses to try to unify and so suspend; the other processor proved the other predicate false based on the program.
f	f	f	
f	$\perp$	f	(Same as other case). One processor did not deal with the truth value of $p_i(a)$ and the other got false as a value.
suspend	suspend	suspend	
suspend	$\perp$	suspend	Some work was done; cannot do more work on one processor and other processor did not work with the predicate.

The commutative and associative closure of @ are obvious. As we previously mentioned the set FIVE with its ordering is a complete lattice (See Figure 1). An interpretation is a map from atomic formulae into the above set. Interpretations with regard to an individual processor are given the point-wise ordering, and any order-preserving map has a least fixed point (lfp) and a greatest fixed point (gfp).

Definition 6

Let Φ denote a parallel-system operator on interpretations based on the maps on the interpretations from the n individual processors with @ defined between operators pointwise.

Lemma 1

Φ is independent of the order of the maps on the individual interpretations upon which Φ is based.

Proof: Referring to Table IV we see the values in the range of the @ operation depends merely on the truth values taken on by the operands under consideration and are order independent.

Interpretations on the system are given a pointwise ordering (i.e. $V_1 < V_2$ iff

$V_1(a) \leq V_2(a)$ with respect to the lattice FIVE for all atoms a) and so, any order-preserving map on parallel interpretations will have a least fixed point. The truth values determined by the lfp of $\Phi_{p,}$, an operator on a five-valued parallel interpretation supplies us with the truth valued determined by the program.

In general, a parallel interpretation is an interpretation that is achieved by parallel evaluation of sequential interpretations. Every sequential interpretation can be considered as (the limiting case of) a parallel interpretation. We now proceed to show the relationship between the parallel interpretations determined by elements in the range of Φ_p and the sequential interpretations determined by the elements in the range of Φ_{ip}, i=1,…n.

Definition 7

For processor i, we define the following family of maps on interpretations with regard to program P. Φ_{ip} is as defined in Definition 4.

(1) $\Phi_{ip}{}^0$ assigns corresponding truth values to reserved relation symbols and given relations; on unreserved relations symbols it is $\perp$.
(2) $\Phi_{ip}{}^{\alpha+1} \equiv \Phi_{ip}(\Phi_{ip}{}^{\alpha})$
(3) For a limit ordinal λ, $\Phi_{ip}{}^{\lambda} \equiv \sup \{ \Phi_{ip}{}^{\alpha} \mid \alpha < \lambda\}$

Definition 8

Let Φ_{ip} be as in Definition 4. Let X be an interpretation of a program P upon which Φ_{ip} is defined. The following are parallel-system interpretations:

$\Phi_p{}^0(X) = X$
$\Phi_p{}^1(X) = \Phi_{1p}{}^1(X) @ \Phi_{2p}{}^1(X) \ldots . @ \Phi_{np}{}^1(X)$
$\Phi_{p_p}{}^{\alpha+1}(X) = \Phi_p(\Phi_p{}^{\alpha}(X))$
$\Phi_p{}^{\lambda}(X) = \text{lub}\ \{\Phi_p{}^{\beta}(X) \mid \beta < \lambda\}$

The following shows $\Phi_p{}^{\alpha}(X)$ is well defined.

Lemma 2

Let μ be any function mapping {1,2,…n} to {0,1} such that lub μ(i) = 1.

Then $\Phi_p{}^1(X) = @_i \Phi_i{}^{\mu(i)}(X)$.

Proof: $\Phi_p{}^1(X)$ is a parallel-system interpretation based on the lub of interpretations implied by sequential interpretations.

Lemma 3:

$\Phi_p{}^{\alpha}(X) = @_i \Phi_i{}^{\mu(i)}(X)$ where μ is any mapping from {1,2…n} to {0,1,2…α} such that $\text{lub}_i(i) = \alpha$, α any ordinal.

Proof: From lemma 1 we see that $\Phi_p{}^{\alpha}(X)$ is a unique interpretation for α a successor ordinal; thus, $\Phi_p{}^{\lambda}(X)$ is well defined for λ a limit ordinal.

Lemma 4:

Let $\Phi_p = @\Phi_{ip}$ be the map from 5-valued parallel interpretations into 5-valued parallel interpretations as defined by Definition 8. Then

(1) Φ_p (@ $\Phi_{ip}^{\lambda i}(X)$) $_{i=1}^{n}$ = (2) (@ $\Phi_{ip}^{\lambda i+1}(X)$) $_{i=1}^{n}$ = (3) @[Φ_p ($\Phi_{ip}^{\lambda i}(X)$)] $_{i=1}^{n}$

Proof: (1) ↔ (2)

(2) is the n-parallel interpretation $\Phi_p^{\alpha}(X)$ where $\alpha = \text{lub}_i (\lambda_i + 1)$. (1) is the interpretation one gets from Φ operating on $\Phi^{\lambda}(X)$, $\lambda = \text{lub}_i \lambda_i$ which by definition = $\Phi^{\lambda+1}(X)$ where

$\lambda = \text{lub}_i \lambda_i$ and so is equal to (2).

(2) ↔ (3)

(3) is the resulting interpretation from Φ_p acting on each sequential interpretation $\Phi_{ip}^{\lambda i}(X)$ as the limiting case of a parallel interpretation, i.e. $\Phi_{ip}^{\lambda i}(X) = \Phi_{ip}^{\lambda i}(X)$ @ $\Phi_{jp}^{0}(X)$ (for all $j \neq i$). When we perform the @ operation we arrive at the parallel interpretation $\Phi_p^{\alpha}(X)$ where $\alpha = \text{lub}_i (\lambda_i + 1)$.

IV. Conclusion

We contend that the approach of defining the semantics of parallel logic programs that we have presented here has a strong relationship with the theory of powerdomains based on the topology established by Plotkin [24]. Additional future work includes an extension of the results of this paper to other concurrent logic programming language and their respective semantics [15].

References

[1] Apt, K.R., Bezem, M., Acrylic Programs, New Generation Computing, 9, pp. 335-363, (1995).

[2] Belnap Jr., N.D., A Useful Four-Valued Logic, Modern Uses of Multiple Valued Logic, (edited by Dunn and Epstein) Reidel, Dordrecht, pp. 8-37, (1977).

[3] Chen, W., Warren, D.S., A Goal-Oriented Approach to Computing the Well-Founded Semantics, Journal of Logic Programming, 17, pp. 279-300, (1993).

[4] Chen, W., Warren, D.S., Toward Effective Evaluation of General Logic Programs, Technical Report 93-CSE-11, Southern Methodist University, (1993).

[5] Della Croce, F., Tsoukiàs A., Moraïtis P., "Why is Difficult to Make Decisions under Multiple Criteria, Proceedings of the Sixth International Conference on AI Planning & Scheduling *(AIPS'02) Workshop on Planning and Scheduling with Multiple Criteria*, pp. 41-45, Toulouse, France, (2002).

[6] Derensart, P. Maluszynski, J. A Grammatical View of Logic Programming, MIT Press, (1993).

[7] Dung, P. an Argumentation Semantics for Logic Programming with Explicit Negation, Proceeding 10th International conference on Logic Programming, pp. 615-30, (1993).

[8] Fitting, M.C., The Family of Stable Models, Journal of Logic Programming, 17, pp. 197-225, (1993).

[9] Fitting, M.C., A Kripke-Kleene Semantics for Logic Programs, Journal of Logic Programming, vol.3 pp. 295-312, (1986).

[10] Fitting, M.C., Logic Programming on a Topological Bilattice, Fundamentica Informatica, vol. 11, pp.209-218, (1988).

[11] Fitting, M.C., Bilattices and the Semantics of Logic Programming, Journal of Logic Programming, vol.11, pp. 91-116, (1991).

[12] Fitting, M.C.,, Ben-Jacob, M., Stratified and Three-Valued Logic Programming Semantics, Logic Programming, Proceedings of the Fifth International Conference and Symposium, editors, Kowalski, R.A., and Bowen, K. S. pp.1054-1069, The MIT Press, (1988).

[13] [13] Fitting, M.C., Ben-Jacob, M., Stratified, Weak Stratified, and Three-valued Semantics, Fundamenta Informatica, vol.13, pp. 19-33, (1990).

[14] Gerth, R., Codish, M., Lichtenstein, Y., Shapiro, E., Fully Abstract Denotational Semantics for Flat Concurrent Prolog, Weizmann Institute Technical Report CS-8803, (1988).

[15] Hewitt, Carl, The repeated demise of logic programming and why it will be reincarnated; What Went Wrong and Why: Lessons from AI Research and Applications. Technical Report SS-06-08. AAAI Press. (March 2006).

[16] Huth, M., Jagadeesan, R., and Schmidt, D.A. Modal Transition Systems: a Foundation for Three-valued Program Analysis. Proceedings of the European Symposium on Programming, Springer LNCS 2028, pp. 155-169, (2001).

[17] Kakas, A., Mancarella, A., Dung, P., The Acceptability Semantics for Logic Programs, Proceedings of the 11th International Conference on Logic Programming, pp.504-519, (1994).

[18] Knaster, B. Une Theoreme sur les Fonctions d'Ensembles, Ann. Soc. Polon. Math. Vol. 6, pp. 133-134 (1928).

[19] Kleene, S.C., Introduction to Metamathematics, Van Nostrand, Princeton, (1952).

[20] Kok, J.N., A compositional Semantics for Concurrent Prolog, Symposium on Theoretical Aspects of Computer Science, pp. 373-388, (1988).

[21] Malfon, B., Characterization of Some Semantics for Logic Programs with Negation and Application to Program Validation, Rapport de Recherche Laboratoire d' Informatique Fondamentale d'Orleans, pp. 94-100, (1994).

[22] Milner, Robin, The Space and Motion of Communicating Agents. Cambridge University Press, (2009).

[23] Ross, K.A., A Procedural Semantics for Well-Founded Negation in Logic Programs, Journal of Logic Programming, 13, pp. 1-22, (1992).

[24] Saraswat, V.A., Concurrent Constrain Programming Languages, Ph.D. thesis, Carnegie-Mellon University, (January 1989).

[25] [25] Schmidt, David, Denotational Semantics, Wm. Brown, Iowa, (1988).

[26] Tarski, A., A Lattice-Theoretical Fixpoint Theorem and its Applications, Pacific Journal of Mathematics, vol.5, pp. 285-309, (1955).

[27] Ueda, K. Guarded Horn Clauses, ICOT Technical Report TR-103, June 1985.

[28] Ueda, K. and Kato, N., The Language Model LMNtal. Proceedings of the 19th International Conference on Logic Programming (ICLP'03), LNCS 2916, Springer-Verlag, pp.517-518, 2003.

[29] Ueda, K. A Pure Meta-Interpreter for Flat GHC, A Concurrent Constraint Language, *Computational Logic: Logic Programming and Beyond* (Essays in Honour of Robert A. Kowalski, Part I), A.C. Kakas, F. Sadri (Eds.), Lecture Notes in Artificial Intelligence 2407, Springer-Verlag, pp.138-161, (2002).

[30] Van Gelder, A. Ross, K.A., Schlipf, J.S., The Well Founded Semantics for General Logic Programs, Journal of the ACM, vol. 38, 3, pp. 620-650, (1991).

Authors Profile

Dr. Marion Ben-Jacob is a Professor in the Department of Mathematics and Computer Information Science at Mercy College for over 30 years. She teaches computer science, mathematics, and critical inquiry, both in the traditional classroom and online. She has recently written papers and spoken extensively on the topics of computer science, computer ethics, online teaching/distance education, collaborative learning, and global learning at numerous conferences. Dr. Ben-Jacob serves on the editorial board of the Journal of Educational Technology Systems. She is the editor and a contributing author of Integrating Computer Ethics across the Curriculum and a recently published e-book, Computer Ethics: Integrating across the Curriculum.