2000

Nonmonotonic Behavior Demonstrated by Stable

James D. Jones
University of Arkansas at Little Rock, james.d.jones@acm.org

NONMONOTONIC BEHAVIOR DEMONSTRATED BY STABLE{}

James D. Jones

Computer Science
College of Information Science and Systems Engineering
University of Arkansas at Little Rock
Little Rock, AR. 72204-1099
james.d.jones@acm.org

I. Abstract

Nonmonotonic reasoning is a critical feature that robust reasoning systems must possess. Another important feature that robust reasoning systems must possess is the ability to reason about collections of objects (i.e., sets.) Logic programming presents us with a very powerful paradigm within which to represent and reason about knowledge. Developments in this field over recent years allow us to reason nonmonotonically, and allow us to reason about sets. Of the semantics introduced over recent years to allow logic programs to reason about sets, *Stable*{} is the newest and most expressive (i.e., comprehensive) approach to date. This paper examines the nonmonotonicity of *Stable*{}.

II. Introduction

Nonmonotonic reasoning is that ability humans possess which allow us to jump to conclusions which are not logically sound, but which nonetheless may be very reasonable. What makes reasoning nonmonotonic is the ability to retract former conclusions in the light of new, contradictory information. Both abilities are needed in order for automated reasoning systems to be robust: the ability to jump to conclusions (even in the face of uncertain, distorted information), and the ability to Achange one=s mind@.

Logic programming presents a very natural paradigm within which to represent and reason about knowledge. Much current research centers around extending the semantics of logic programming to allow us to reason more correctly (that is, to reason about new classes of problems.) With respect to extending the semantics, the semantics analyzed here allows us to reason about sets. Granting intelligent systems the ability to reason about sets is an important feature, since we as humans commonly reason about collections of objects.

The smallest piece of information in a logic program is an *atom* and is of the form

$$P(t_1, ..., t_n)$$

where P is an n-ary predicate constant , the t_i are terms, $n \$ 0$. Terms represent individual objects (concrete or abstract) that we care to reason about. A predicate constant names a property or a relationship that exists among these objects. An example could be something such as

married(*john*, *mary*, *1990*, *texas*)

which may represent something such as the fact that AJohn and Mary were married in Texas in 1990".

A *term* is defined inductively as follows:

1. an object constant is a term
2. an object variable is a term.
3. if f is an n-ary function symbol and $t_1, ..., t_n$ are terms, then $f(t_1, ..., t_n)$ is a term.

An example of an object constant would be something like *john*. An example of an object variable would be something like AX@, as in Athere exists an X to whom John is married@ as in

married(*john*, *X*).

An example of a functional expression would be something like A*first_child_of*(*bill, hillary*)@. In this example (using the Clintons as an example), the functional expression is equivalent to the object constant *chelsea*. That is, the formula

on_deans_list(*chelsea*)

is equivalent to the formula

on_deans_list(*first_child_of*(*bill, hillary*))

In practice, there is a special case of term type 3, used to represent a list, and is of the form:

4. if $t_1, ..., t_n$ n\$0 are terms, then $[t_1, ..., t_n]$ is a term.

The purpose of the semantics analyzed in this paper is to introduce a new kind of term: sets. Such terms will be of the form:

5. if $t_1, ..., t_n$, n\$0 are terms, then $\{t_1, ..., t_n\}$ is a term.

Notice that such an inductive definition of terms allows for arbitrarily complex terms.

III. Definition of *Stable*{}

2.1 entailment with respect to a belief set

Let P be an *extended logic program with setof.* Let *Sf*L*it* be a set of literals such that for every literal *l* 0 *S* appearing in a formula in P of the form *setof* or 5*setof*, the definition of *l* is finite.

Definition: *Entailment w.r.t. S.* Let *Q* be some ground formula.

case 1: *Q* is of the form *setof*($q(\bar{X})$,*F* $\bar{X}$(),*Y*). *S* Ö *Q* iff *Y* is {q($\bar{t}$): *S* Ö *F*($\bar{t}$) where q($\bar{t}$) is an arbitrary term formed from the tuple of ground terms which are among the ground terms appearing in *F*}.

case 2: *Q* is of the form 5*setof*($q(\bar{X})$,*F*($\bar{X}$),*Y*). *S* Ö *Q* iff *Y* is not {q($\bar{t}$): *S* Ö *F*($\bar{t}$) where q($\bar{t}$) is an arbitrary term formed from the tuple of ground terms which are among the ground terms appearing in *F*}.

case 3: *Q* is a literal. *S* Ö *Q* iff *Q* 0 *S.*

case 4: *Q* is an extended literal, that is, of the form *not P*. S Ö *Q* iff *P* ó *S.* ~

Stable{} (Jones 97, Jones 99) is the newest among the few semantics that have been developed that allow one to reason about sets. Future work will demonstrate that this semantics completely subsumes the other semantics, is more expressive than the other semantics, and is simpler than the other semantics. Other work in progress also demonstrates that the level of nonmonotonicity within *Stable*{} is greater than that of other semantics. (Space limitations preclude the proof of such claims here.)

1. Syntax

Terms follow the standard definitions (as outlined and enhanced in the introduction). Formula also follow the standard definitions, with the addition that there are two pre-defined forms: if $F(\bar{X})$ is a formula, then

setof($q(\bar{X})$,$F(\bar{X})$,*Y*)

and

5setof($q(\bar{X})$,$F(\bar{X})$,*Y*)

are formulae that can appear only in the bodies of rules where

$\bar{X}$ is an n-tuple of variables appearing in $F(\bar{X})$, n\$1

$q(\bar{X})$ and *Y* are arbitrary terms.

2. Semantics

The semantics is defined with respect to entailment. This is the fundamental difference between our approach to a definition of semantics for sets, and other approaches. Further, our semantics clearly distinguishes between that which is entailed by a belief set, and that which is entailed by a program. This is a crucial distinction, and one that is lacking in the other semantics.

Definition: *S satisfies* a rule iff for every formula *F* in the body of the rule, if *S* Ö *F*, then *L*, the head of the rule, is an element of *S*. That is, *L* 0 *S*.

Definition: *S* is a *belief set* of P iff *S* is a minimal set satisfying all the rules of P.

2.2 entailment with respect to a program

Negation-as-failure creates the possibility that a program may have more than one belief set. To distinguish between that which is entailed by a belief set, versus that which is entailed by all belief sets (that is, a program), we introduce the predicate symbol *p_set_of* (which is of the same syntactical form as *setof* and is introduced only as a convenience for the reader.)

Definition: *Entailment w.r.t.* P. Let *Q* be some ground formula. Let a(P) = {*S*: *S* is a belief set of P}.

case 1: *Q* is of the form *p_set_of*(*q*($\bar{X}$),*F*($\bar{X}$),*Y*). P Ö *Q* iff *Y* is {q($\bar{t}$): *S* Ö *F*($\bar{t}$) for all *S* 0 a(P), where q($\bar{t}$) is an arbitrary term formed from the tuple *t* of ground terms appearing in *F*}.

case 2: *Q* is of the form 5*p_set_of*(*q*($\bar{X}$),*F*($\bar{X}$),*Y*). P Ö *Q* iff *Y* is not {q($\bar{t}$): *S* Ö *F*($\bar{t}$) for all *S* 0 a(P), where q($\bar{t}$) is an arbitrary term formed from the tuple *t* of ground terms appearing in *F*}.

case 3: *Q* is a literal. P Ö *Q* iff *Q* 0 *S* for all *S* 0 a(P).

case 4: *Q* is an extended literal, that is, of the form *not P*, where *P* is a ground literal. P Ö *Q* iff *P* ó *S* for some *S* 0 a(P).

~

Definition: *answers to queries*. Let *Q* be some sequence of ground formulae, *q* 0 *Q* be a ground formula. For any such *Q* posed as a query to P, P answers *yes* iff P Ö *q* for all *q* 0 *Q*, *no* iff P Ö 5*q* for some *q* 0 *Q*, and *unknown* otherwise. (That is, P Ö *Q* iff P Ö *q* for all *q* 0 *Q*; P Ö 5*Q* iff P Ö 5*q* for some *q* 0 *Q*; otherwise P *Q* and P 5*Q*.)

IV. Nonmonotonicity of *Stable*{}

There are multiple forms of nonmonotonicity that *Stable*{} is capable of producing. There is nonmonotonicity induced by negation-as-failure. In the cases where inferences are made on the basis of missing information, as that missing information is added to our knowledge base, those previous inferences need to be withdrawn. This same mechanism augmented by the ability to represent and reason about sets produces additional nonmonotonicity in that the intensional sets may differ with the addition of new information. There are yet other forms of nonmonotonicity demonstrated by *Stable*{} which are not demonstrated by other semantics for sets. These forms of nonmonotonicity are due to: 1) the fact that *Stable*{} allows for multiple belief sets, 2) the fact that *Stable*{} allows more arbitrary intensional sets, and 3) the fact that *Stable*{} distinguishes between that which is entailed by a belief set (i.e., local), and that which is entailed by a program (i.e., global). With regard to point 2, *Stable*{} allows an unspecified level of nesting of intensional set definitions, it allows sets to be defined over a tuple of variables, and it allows negation-as-failure as part of the intensional set definitions. None of the other set semantics allow any of these features. Length restrictions do not allow us to examine all these subtleties.

Example 1

This example demonstrates nonmonotonicity induced by negation-as-failure.

r(*a*) 7
p(*X*) 7 *not q*(*X*)

Using the domain closure assumption (as will be used throughout this paper), the model for this program[1] is {*r*(*a*), *p*(*a*)}. Let us add the atom *q*(*a*) to our program. The model for this modified program is {*r*(*a*), *q*(*a*)}. The atom *p*(*a*) belongs to the model of the original program, but not to the model of the modified program. Clearly, this is nonmonotonic behavior. This is also very standard for logic programs, and is not unique to these semantics.

~

There is yet a slight twist to the notion of nonmonotonicity that is induced by the construction of sets. Strictly speaking, it is still nonmonotonic behavior as before. However, what is new is that the set that is constructed

[1] In this paper, "program", "theory", and "database" are synonymous.

differs. Without a semantics for sets, such a form of nonmonotonicity would not be possible.

Example 2

r(*a*) 7
r(*b*) 7
p(*S*) 7 *setof*(*X*, (*r*(*X*), *not* s(*X*)), *S*)

A model for this program would be: {*r*(*a*), *r*(*b*), *p*({*a*, *b*}) }. Let us add the atom *s*(*b*) to our program. The model for our modified program is {*r*(*a*), *r*(*b*), *p*({*a*}), *s*(*b*)}. The atom p({a, b}) belongs to the model of our original program, while the atom p({a}) belongs to the model of our modified program. This again is nonmonotonic behavior. The set that is entailed by the remainder of the program differs. Originally, the program entailedthe set {*a*, *b*}. The modified program entails the set {*a*}. ~

Example 3

The semantics defines sets that are entailed by a belief set, and sets that are entailed by the program. This distinction is important when there are multiple belief sets. This example demonstrates the nonmonotonicity induced by p_set_of. Let P be the following program:

p(*a*) 7
p(*b*) 7 *not p*(*c*)
p(*c*) 7 *not p*(*b*)
p(*d*) 7 *not p*(*e*)

This program has two belief sets: {*p*(*a*), *p*(*b*), *p*(*d*)} and {*p*(*a*), *p*(*c*), *p*(*d*)}. Limiting *p_set_of* to the language of the program, the only atoms entailed by this program are *p*(*a*), *p*(*d*), and *p_set_of*(*X*, *p*(*X*),{*a*, *d*}). If we add the rule

p(*e*) 7 *not p*(*b*)

the program yields the two belief sets {*p*(*a*), *p*(*b*), *p*(*d*)} and {*p*(*a*), *p*(*c*), *p*(*e*)}. Again, limiting *p_set_of* to the language of the program, the only atoms entailed by this program are *p*(*a*), and *p_set_of*(*X*, *p*(*X*),{*a*}). Previous to the addition of this rule, the program entailed *p*(*d*). Now, it no longer does. Further, the set defined by *p_set_of* was previously {*a*, *d*}, whereas it is now {*a*}. So, both the atoms that are inferred, and the intensional set have been reduced by new information. ~

As an interesting aside, the addition of new information could reduce the number of belief sets. While reducing the number of atoms entailed by the program would be considered nonmonotonic, reducing the number of belief sets is not considered so. (In fact, the reduction of belief sets may be an increase in information.) What is significant about this observation is that *Stable*{} could easily be extended to be based upon epistemic specifications. Within the context of epistemic specifications (Baral, Gelfond 94) where modal operators allow one to reason among belief sets, it is quite possible that this very action could become nonmonotonic. Expanding *Stable*{} to the language of epistemic specifications is an area of future work. The additional nonmonotonicity induced by epistemic specifications (in particular, as it relates to sets) is an area for further investigation. The following example demonstrates the reduction in belief sets.

Example 4

Let P be the following program:

p(*a*) 7
p(*b*) 7 *not p*(*c*)
p(*c*) 7 *not p*(*b*)

This program has two belief sets: {*p*(*a*), *p*(*b*)} and {*p*(*a*), *p*(*c*)}. Limiting p_set_of to the language of the program, the only atoms entailed by this program are *p_set_of*(*X*, *p*(*X*),{*a*}), and *p*(*a*). If we add the atom *p*(*b*) to our program, we reduce the number of belief sets yielding only one belief set: {*p*(*a*), *p*(*b*)}.~

Example 5

This example demonstrates the nonmonotonicity induced by using negation-as-failure in the intensional set definition. Let P be the following program:

p(*a*) 7
r(*b*) 7
r(*c*) 7
set(*S*) 7 *setof*(*X*, *not p*(*X*), *S*)

The belief set for this program is {*p*(*a*), *r*(*b*), *r*(*c*), *set*({*b*, *c*}). If we add the atom *p*(*b*) to our program, the belief set becomes {*p*(*a*), *p*(*b*), *r*(*b*), *r*(*c*), *set*({*c*}). That is, the new information causes the atom *set*({*b*, *c*}) to no longer be entailed by the program, and it causes the set defined by the intensional definition to be different. Similarly, the original program entailed *p_set_of*(*X*, *not p*(*X*), {*b*, *c*}), whereas the modified program entails *p_set_of*(*X*, *not p*(*X*), {*c*}). ~

Example 6

This example produces the same results as does example 5 in the context of multiple belief sets. Let our program be the following:

p(*a*) 7
r(*b*) 7 *not r*(*c*)
r(*c*) 7 *not r*(*b*)
set(*S*) 7 *setof*(*X*, *not p*(*X*), *S*)

This program yields two belief sets: {*p*(*a*), *r*(*b*), *set*({*b*, *c*})} and {*p*(*a*), *r*(*c*), *set*({*b*, *c*})}. If we add the rule *p*(*b*) 7 *not r*(*b*), the resulting two belief sets are: {*p*(*a*), *r*(*b*), *set*({*b*, *c*})}, and {*p*(*a*), *p*(*b*), *r*(*c*), *set*({*c*})}. The original program entailed *set*({*b*, *c*}) and *p_set_of*(*X*, *not p*(*X*), {*b*, *c*}), neither of which is entailed by the modified program. There is not an atom with the predicate constant *set* entailed by the modified program (which differs from example 5), and yet it entails *p_set_of*(*X*, *not p*(*X*), {*c*}). ~

V. SUMMARY AND CONCLUSIONS

The semantics presented here demonstrates nonmonotonic behavior in multiple fashion. There is the nonmonotonicity induced by negation as failure. There is the nonmonotonicity induced by intensional set terms. This form of nonmonotonicity has at least two forms: a set that is entailed by a program may no longer be entailed by the addition of new information, and the set that is entailed by the formulae can differ when adding new data to a program. That is, at one level we have nonmonotonicity in the atoms that are entailed by a program, and (simultaneously) at another level we have nonmonotonicity in the set that is defined by the formulae. That is, w.r.t. sets, there is nonmonotonicity induced by negation-as-failure, and nonmonotonicity induced by the intensional sets. Further, there is the additional nonmonotonicity induced by the predicate *p_set_of*, since this predicate concerns itself with sets that are entailed by the program, rather than sets entailed by a particular belief set. A peculiar fact is that it is possible that with the introduction of a single fact, we may have nonmonotonic behavior (in that previously entailed formulae many no longer be entailed), and yet simultaneously, there is an increase in the information that is entailed by a program (in that the reduction of belief sets yields more facts that are entailed by the program.) Many of these forms of nonmonotonicity are not present in other semantics for sets.

References

(Apt, Bol 94) Apt, Krzysztof R., and Roland N. Bol: Logic Programming and Negation: A Survey, *Journal of Logic Programming*, vol 19/20 May/July 1994.

(Baral, Gelfond 94) Baral, Chitta, and Michael Gelfond: Logic Programming and Knowledge Representation, *Journal of Logic Programming*, vol 19/20 May/July 1994.

(Beeri, et. al. 91) Beeri, S. Naqvi, O. Shmueli, and S. Tsur: Set Constructors in a Logic Database Language, *Journal of Logic Programming*, 10(3):181-232, 1991.

(Dovier et al. 95) Dovier, Agostino, Enrico Pontelli, and Gianfranco Rossi: The *CLP* Language {log}, and the relation between Intensional Sets and Negation, New Mexico State University technical report NMSU-CSTR-9503, March 95.

(Dovier et al 96) Dovier, Agostino, Enrico Pontelli, and Gianfranco Rossi: {log}: A language for programming in logic with finite sets, *Journal of Logic Programming*, 28(1):1-44, 1996.

(Gelfond 92) Gelfond, Michael: Logic Programming and Reasoning with Incomplete Information (to appear in *The Annals of Mathematics and Artificial Intelligence*, 1994)

(Gelfond, Lifschitz 88) Gelfond, Michael and Vladimir Lifschitz: The Stable Model Semantics for Logic Programming, *5th Intl Conference on Logic Programming* 1988

(Gelfond, Lifschitz 90) Gelfond, Michael, and Vladimir Lifschitz: Logic Programs with Classical Negation. In D. Warren and Peter Szeredi, editors, *Logic Programming: Proceedings or the 7th Int=l Conf*, 1990.

(Gelfond, Lifschitz 91) Gelfond, Michael, and Vladimir Lifschitz: Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing*, No. 9 1991

(Gervet 94) Gervet, Carmen.: Constraint Logic Programming with Finite Set Domains, *Proceedings of International Logic Programming Symposium*, 1994

(Gervet 97) Gervet, Carmen: Interval propagation to reason about sets: Definition and implementation of a practical language, *Constraints*, 1(3):191-244, 1997.

(Jayaraman 92) Jayaraman, B.: Implementation of Subset-Equational Programs, *Journal of Logic Programming*, 12(4):299-324, 1992

(Jayaraman, Plaisted 89) Jayaraman, B., and D. A. Plaisted: Programming with Equations, Subsets and Relations, *Proceedings of NACLP89*, MIT Press, 1989

(Jones 97) Jones, James D.: *Stable Model Semantics for Sets*, Ph.D. dissertation, New Mexico State University, Department of Computer Science, July 1997

(Jones 99) Jones, James D.: AA Declarative Semantics For Sets Based on the Stable Model Semantics@, *Declarative Programming with Sets* (*DPS >99*), in conjunction with *Principles, Logics, and Implementations of high-level programming languages* , Paris, France, Paris, France, 1999

(Kuper 90) Kuper, Gabriel: Logic Programming with Sets, *Journal of Computer and System Science*, 1990

(Lloyd 87) Lloyd, J.W.: *Foundations of Logic Programming*, Berlin, Germany: Springer-Verlag