On the Role of Negation in Choice Logic Programs

Marina De Vos* and Dirk Vermeir

Dept. of Computer Science
Free University of Brussels, VUB
Pleinlaan 2, Brussels 1050, Belgium
Tel: +32 2 6293308
Fax: +32 2 6293525
{marinadv,dvermeir}@tinf.vub.ac.be
http://tinf2.vub.ac.be

Abstract. We introduce choice logic programs as negation-free datalog programs that allow rules to have exclusive-only (possibly empty) disjunctions in the head. Such programs naturally model decision problems where, depending on a context, agents must make a decision, i.e. an exclusive choice out of several alternatives. It is shown that such a choice mechanism is in a sense equivalent with negation as supported in seminegative ("normal") datalog programs. We also discuss an application where strategic games can be naturally formulated as choice programs: it turns out that the stable models of such programs capture exactly the set of Nash equilibria. We then consider the effect of choice on "negative information" that may be implicitly derived from a program. Based on an intuitive notion of unfounded set for choice programs, we show that several results from (seminegative) disjunctive programs can be strengthened; characterizing the position of choice programs as an intermediate between simple positive programs and programs that allow for the explicit use of negation in the body of a rule.
Keywords: Logic programming, choice, unfounded sets, game-theory

1 Choice Logic Programs for Modeling Decision Making

When modeling agents using logic programs, one often has to describe a situation where an agent needs to make a decision, based on some context. A *decision* can be thought of as a single choice between several competing alternatives, thus naturally leading to a notion of nondeterminism. Using seminegative (also called "normal") programs, such a choice can be modeled indirectly by using stable model semantics, as has been argued convincingly before [10, 8]. E.g. a program such as

$$p \leftarrow \neg q$$
$$q \leftarrow \neg p$$

* Wishes to thank the FWO for their support.

has no (unique) total well-founded model but it has two total stable models, namely $\{p, \neg q\}$ and $\{\neg p, q\}$, representing a *choice* between p and q (note that his choice is, however, not exclusive, as e.g. p may very well lead to q in a larger program).

In this paper, we simplify matters by providing for explicit choice sets in the head of a rule. Using $p \oplus q$ to denote an exclusive choice between p and q, the example above can be rewritten as

$$p \oplus q \leftarrow$$

Intuitively, $\oplus$ is interpreted as "exclusive or", i.e. either p or q, but not both, should be accepted in the above program.

Definition 1. *A **choice logic program** is a finite set of rules* [1] *of the form $A \leftarrow B$ where A, the head, and B, the body, are finite sets of atoms.*

Intuitively, atoms in A are assumed to be xor'ed together while B is read as a conjunction. In examples, we often use $\oplus$ to denote exclusive or, while "," is used to denote conjunction. If we want to single out an atom in the head of a rule we sometimes write $A \oplus a$ to denote $A \cup \{a\}$.

The semantics of choice logic programs can be defined very simply.

Definition 2. *Let P be a choice logic program. The **Herbrand base** of P, denoted $\mathcal{B}_P$, is the set of all atoms occurring in the rules of P. A set of atoms $I \subseteq \mathcal{B}_P$ is **model** of P if for every rule $A \leftarrow B$, $B \subseteq I$ implies that $I \cap A$ is a singleton, i.e. $|A \cap I| = 1$*[2]*. A model of P is called **stable** iff it is minimal (according to set inclusion).*

Note that the above definitions allow for *constraints* to be expressed as rules where the head is empty.

Example 1 (Graph 3-colorability). Given a graph assign each node one of three colors such that no two adjacent nodes have the same color. This problem is know as graph 3-colorability and can be easily transformed in the following choice program:

$$\begin{aligned} col(X,r) \oplus col(X,g) \oplus col(X,b) &\leftarrow node(X) \\ &\leftarrow edge(X,Y), col(X,C), col(Y,C) \end{aligned}$$

The first rule states that every node should take one and only one of the three available colors (r, g or b). The second demands that two adjacent nodes have different colors. To this program we only need to add the facts (rules with empty body) that encode the graph to make sure that the stable models for this program reflect the possible solutions for this graph's 3-colorability. The facts look either as $node(a) \leftarrow$ or $edge(a,b) \leftarrow$.

[1] In this paper, we identify a program with its grounded version, i.e. the set of all ground instances of its clauses. This keeps the program finite as we do not allow function symbols (i.e. we stick to datalog).

[2] We use $|X|$ to denote the cardinality of a set X.

	Does not confess	*Confess*
Does not confess	$3, 3$	$0, 4$
Confess	$4, 0$	$1, 1$

Fig. 1. The prisoner's dilemma (Ex. 2)

The following example shows how choice logic programs can be used to represent strategic games[6].

Example 2 (The Prisoner's Dilemma). Two suspects of a crime (they jointly committed) are arrested and interrogated separately. The maximum sentence for their crime is four years of prison. But if one betrays the other while the latter keeps quiet, the former is released while the silent one receives the maximum penalty. If they both confess they are both convicted to three years of prison. In case they both remain silent, they are convicted for a minor felony and sent to prison for only a year. In game theory this problem can be represented as a strategic game with a graphical notation as in Fig. 1. One player's actions are identified with the rows and the other player's with the columns. The two numbers in the box formed by row r and column c are the players' payoff (e.g., the years gained with respect to the maximum sentence). When the row player chooses r and the column player chooses c, the first component represents the payoff of the row player. It is easy to see that the best action for both suspects is to confess because otherwise there is a possibility that they obtain the full four years. This is called a Nash equilibrium.
This game can be easily transformed to the following choice logic program where d_i stands for "suspect i does not confess" and c_i means "suspect i confesses":

$$
\begin{aligned}
d_1 \oplus c_1 &\leftarrow \\
d_2 \oplus c_2 &\leftarrow \\
c_1 &\leftarrow d_2 \\
c_1 &\leftarrow c_2 \\
c_2 &\leftarrow d_1 \\
c_2 &\leftarrow c_1
\end{aligned}
$$

The first two rules express that both suspects have to decide upon a single action. The last four indicate which action is the most appropriate given the other suspect's actions.
This program has a single stable model corresponding to the Nash equilibrium of the game, namely $\{c_1, c_2\}$.

In [3], it was shown that every finite strategic game can be converted to a choice logic program whose stable models correspond with the game's Nash equilibria.

Definition 3 ([6]). *A **strategic game** is a tuple $\langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ where*

1. *N is a finite set of **players**;*

	Head	*Tail*
Head	1, 0	0, 1
Tail	0, 1	1, 0

Fig. 2. Matching Pennies (Ex. 3)

2. *for each player $i \in N$, A_i is a nonempty set of* ***actions*** *that are available to her (we assume that $A_i \cap A_j = \emptyset$ whenever $i \neq j$) and*
3. *for each player $i \in N$, $\geq_i$ is a* ***preference relation*** *on $A = \times_{j \in N} A_j$*

An element $\mathbf{a} \in A$ is called a ***profile****. For a profile $\mathbf{a}$ we use $\mathbf{a}_i$ to denote the component of $\mathbf{a}$ in A_i. For any player $i \in N$, we define $A_{-i} = \times_{j \in N \setminus \{i\}} A_j$. Similarly, an element of A_{-i} will often be denoted as $\mathbf{a}_{-i}$. For $\mathbf{a}_{-i} \in A_{-i}$ and $a_i \in A_i$ we will abbreviate as $(\mathbf{a}_{-i}, a_i)$ the profile $\mathbf{a}' \in A$ which is such that $\mathbf{a}'_i = a_i$ and $\mathbf{a}'_j = \mathbf{a}_j$ for all $j \neq i$.*

A ***Nash equilibrium*** *of a strategic game $\langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ is a profile $\mathbf{a}^*$ satisfying*

$$\forall a_i \in A_i \cdot (\mathbf{a}^*_{-i}, \mathbf{a}^*_i) \geq_i (\mathbf{a}^*_{-i}, a_i)$$

Intuitively, a profile $\mathbf{a}^*$ is a Nash equilibrium if no player can unilaterally improve upon his choice. Put in another way, given the other players' actions $\mathbf{a}^*_{-i}$, $\mathbf{a}^*_i$ is the best player i can do[3].

Not every strategic game has a Nash equilibrium as demonstrated by the next example.

Example 3 (Matching Pennies). Two persons are tossing a coin. Each of them has to choose between Head or Tail. If the choices differ, person 1 pays person 2 a Euro; if they are the same, person 2 pays person 1 a Euro. Each person cares only about the amount of money that she receives. The game modeling this situation is depicted in Fig. 2. This game does not have a Nash equilibrium. The corresponding choice logic program would look like:

$$\begin{aligned} h_1 \oplus t_1 &\leftarrow \\ h_2 \oplus t_2 &\leftarrow \\ h_1 &\leftarrow h_2 \\ t_1 &\leftarrow t_2 \\ h_2 &\leftarrow t_1 \\ t_2 &\leftarrow h_1 \end{aligned}$$

This program has no stable model as the game has no Nash equilibrium. Notice that this would not have been the case if we would have used inclusive disjunctions instead of exclusive ones.

Theorem 1. *For every strategic game $G = \langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ there exists a choice logic program P_G such that the set of stable models of P_G coincides with the set of Nash equilibria of G.*

[3] Note that the actions of the other players are not actually known to i.

The choice logic program P_G obtained for a game, as one can see form the examples, consists of rules expressing that each player has to make a single choice out of her action set and rules expressing the best action for a player given the different actions of the other players.

2 Negation in Choice Logic Programs

While negation is not explicitly present in choice logic programs, it does appear implicitly. E.g. deciding on a in a rule $a \oplus b \leftarrow$ implicitly excludes b from any model; which can be read as "$\neg b$ is true". A similar effect can be observed for constraints: if e.g. a is true, then the presence of a rule $\leftarrow a, b$ implies that b must be false.

Still, there is a difference with seminegative programs because, although implicitly implied negative information may prevent the further application of certain rules, such information can never be used to *enable* the inference of further atoms. The latter is possible e.g. in seminegative logic programs or disjunctive logic programs where the body of a rule may contain negated atoms. Hence choice logic programs can be regarded as an interesting intermediate system in between purely positive logic programs, where a model can be computed without taking into account any negative information[4] and systems that allow for explicit negation in (the body of) a rule. In the remainder of this paper we will compare the role of negation in choice logic programs with both seminegative logic programs and seminegative disjunctive logic programs.

2.1 Simulating Seminegative Logic Programs

It turns out that choice logic programs can simulate semi-negative datalog programs, using the following transformation, which resembles the one used in [9] or [7] for the transformation of general disjunctive programs into negation-free disjunctive programs.

Definition 4. *Let P be a semi-negative logic program. The corresponding choice logic program $P_\oplus$ can be obtained from P by replacing each rule $r : a \leftarrow B, \neg C$ from P with $B \cup C \subseteq \mathcal{B}_P$ and $C \neq \emptyset$, by*

$$\begin{aligned} a_r \oplus K_C &\leftarrow B \quad (r'_1) \\ a &\leftarrow a_r \quad (r'_2) \\ \forall c \in C \ \cdot K_C &\leftarrow c \quad (r'_3) \end{aligned}$$

where a_r and K_C are new atoms that are uniquely associated with the rule r.

A model M for $P_\oplus$ is called ***rational*** *iff:*

$$\forall K_C \in M \cdot M \cap C \neq \emptyset$$

[4] Of course, as a last step, the complement of the positive interpretation can be declared false as a consequence of the closed world assumption.

Intuitively, K_C is an "epistemic" atom which stands for "the (non-exclusive) disjunction of atoms from C is believed". If the positive part of a rule in the original program P is true, $P_\oplus$ will choose (rules r'_1) between accepting the conclusion and K_C where C is the negative part of the body; the latter preventing rule application. Each conclusion is tagged with the corresponding rule (r'_2), so that rules for the same conclusion can be processed independently. Finally, the truth of any member of C implies the truth of K_C (rules r'_3).

Intuitively, a rational model contains a justification for every accepted K_C.

Proposition 1. *Let P be a semi-negative datalog program. M is a rational stable model of $P_\oplus$ iff $M \cap \mathcal{B}_P$ is a (total) stable model of P.*

The rationality restriction is necessary to prevent K_C from being accepted without any of the elements of C being true. For positive-acyclic programs, we can get rid of this restriction.

Definition 5. *A semi-negative logic program P is called* ***positive-acyclic*** [5] *iff there is an assignment of positive integers to each element of $\mathcal{B}_P$ such that the number of the head of any rule is greater than any of the numbers assigned to any non-negated atom appearing in the body.*

Proposition 2. *Let P be a semi-negative positive-acyclic logic program. There exists a choice logic program P_c such that M is a stable model of P_c iff $M \cap \mathcal{B}_P$ is a stable model of P.*

The reverse transformation is far less complicated.

Proposition 3. *Let $P_\oplus$ be a choice program. There exists a semi-negative datalog program P (containing constraints) such that M is a stable model of $P_\oplus$ iff M is a stable model of P.*

2.2 Unfounded Sets and Seminegative Disjunctive Programs

In this section, we formalize implicit negative information by defining an appropriate notion of "unfounded set" for choice logic programs and we investigate its properties and usefulness for the computation of stable models.

It turns out that many of the results of [5] remain valid or can even be strengthened:

1. For choice logic programs, the greatest unfounded set is defined on any interpretation, which is not the case for disjunctive programs.
2. Contrary to disjunctive programs, the results for choice programs remain valid in the presence of constraints.
3. For choice logic programs, the $\mathcal{R}_{P,I}$ (see Definition 9) operator, when repeatedly applied to $\mathcal{B}_P$, always yields the greatest unfounded set w.r.t. I.

[5] In [5] a similar notion is called "head-cycle free".

4. Because of (1) above, the $\mathcal{W}_P$ (see Definition 8) operator can be used in the computation of a stable model. For disjunctive programs, this is not possible because there is no guarantee that an intermediate interpretation has a greatest unfounded set.

Definition 6. *Let P be a choice logic program . An **interpretation** is any consistent[6] subset of $(\mathcal{B}_P \cup \neg\mathcal{B}_P)$. We use $\mathcal{I}_P$ to denote the set of all interpretations of P. An interpretation I is **total** iff[7] $I^+ \cup I^- = \mathcal{B}_P$. A total interpretation M is called a (**stable**) **model** iff M^+ is a (stable) model of P. A set $X \subseteq \mathcal{B}_P$ is an **unfounded set** for P w.r.t. an interpretation I iff for each $p \in X$ one of the following three conditions holds:*

1. *$\exists r : A \oplus p \leftarrow B \in P$ such that $A \cap I \neq \emptyset$ and $B \subseteq I$, or*
2. *$\exists r : \leftarrow B, p \in P$ such that $B \subseteq I$, or*
3. *$\forall r : A \oplus p \leftarrow B \in P$ at least one of the following conditions is satisfied:*
   (a) $B \cap \neg I \neq \emptyset$, or
   (b) $B \cap X \neq \emptyset$, or
   (c) $A \cap B \neq \emptyset$

*The set of all unfounded sets for P wrt I is denoted $\mathcal{U}_P(I)$. The **greatest unfounded set** wrt I, denoted $\mathcal{GUS}_P(I)$, is defined by $\mathcal{GUS}_P(I) = \bigcup_{X \in \mathcal{U}_P(I)} X$. I is called **unfounded-free** iff $I \cap \mathcal{GUS}_P(I) = \emptyset$.*

Condition (1) above expresses the fact that choice is exclusive and thus, alternatives to the actual choice are to be considered false. Condition (2) implies that any atom that would cause a constraint to be violated may be considered false. Condition (3) resembles the traditional definition of unfounded set by expressing when a rule cannot be used to infer a new atom: in case (a), the rule is "blocked" by the current interpretation; in case (b), the rule's application depends on an unfounded literal while case (c) indicates that the rule is useless[2] since the body contains one of the choices in the head.

The next proposition shows that the name "greatest unfounded set" is well-chosen for the union of all unfounded sets, $\mathcal{GUS}_P(I)$.

Proposition 4. *Let I be an interpretation for the choice logic program P. Then, $\mathcal{GUS}_P(I) \in \mathcal{U}_P(I)$. Moreover, $\mathcal{GUS}_P$ is a monotonic operator; i.e. if $I_1 \subseteq I_2$, then $\mathcal{GUS}_P(I_1) \subseteq \mathcal{GUS}_P(I_2)$.*

Note that the above proposition is false for disjunctive logic programs [5]. In fact, for such programs, $\mathcal{GUS}_P(I) \in \mathcal{U}_P(I)$ is only guaranteed if I is unfounded-free or d-unfounded-free[2].

Proposition 5. *Let M be a model for the choice logic program P. Then $M^- \in \mathcal{U}_P(I)$.*

---

[6] For X a set of literals, we use $\neg X$ to denote $\{\neg p | p \in X\}$ where $\neg\neg a = a$ for any atom a. X is consistent iff $X \cap \neg X = \emptyset$.

[7] For a subset $X \subseteq (\mathcal{B}_P \cup \neg\mathcal{B}_P)$, we define $X^+ = X \cap \mathcal{B}_P$ and $X^- = \neg(X \cap \neg\mathcal{B}_P)$.

Unfortunately, the converse does not hold, as can be seen from the interpretation $\{a, b\}$ of the single-rule program $a \oplus b \leftarrow$ which is not a model, although its complement (the empty set) is trivially unfounded. For seminegative disjunctive logic programs, the converse does hold[5].

Proposition 6. *Let P be a choice logic program . A total interpretation is a stable model iff it is unfounded-free.*

Combining Propositions 5 and 6 yields a characterization of stable models in terms of unfounded sets which also holds for disjunctive programs.

Corollary 1. *Let P be a choice logic program. An interpretation M is a stable model for P iff $\mathcal{GUS}_P(M) = M^-$.*

Definition 7. *Let P be a choice logic program. The* ***immediate consequence operator****, $\mathcal{T}_P : 2^{(\mathcal{B}_P \cup \neg\mathcal{B}_P)} \rightarrow 2^{\mathcal{B}_P}$, is defined by*

$$\mathcal{T}_P(I) = \{a \in \mathcal{B}_P \mid \exists A \oplus a \leftarrow B \in P \cdot A \subseteq \neg I \wedge B \subseteq I\}$$

This operator adds those atoms that are definitely needed in any model extension of I. It is clearly monotonic.

The $\mathcal{W}_P$ operator, which uses the same intuition as the one defined in [4], uses $\mathcal{T}_P$ to extend I^+ and $\mathcal{GUS}_P$ to extend I^-.

Definition 8. *Let P be a choice logic program. The operator $\mathcal{W}_P : \mathcal{I}_P \rightarrow 2^{(\mathcal{B}_P \cup \neg\mathcal{B}_P)}$ is defined by*

$$\mathcal{W}_P(I) = \mathcal{T}_P(I) \cup \neg\mathcal{GUS}_P(I)$$

Note that $\mathcal{W}_P$ is monotonic and skeptical as it only adds literals that *must* be in any model extension of I. The following result also holds for disjunctive programs (without constraints).

Proposition 7. *Let P be a choice logic program and let M be a total interpretation for it. M is a stable model iff M is a fixpoint of $\mathcal{W}_P$.*

The least fixpoint $\mathcal{W}_P^\omega(\emptyset)$ of $\mathcal{W}_P$ can, if it exists[8], be regarded as the "kernel" of any stable model.

Proposition 8. *Let P be a choice logic program . If $\mathcal{W}_P^\omega(\emptyset)$ exists then $\mathcal{W}_P^\omega(\emptyset) \subseteq M$ for each stable model M. If $\mathcal{W}_P^\omega(\emptyset)$ does not exist then P has no stable models.*

Because $\mathcal{W}_P$ is deterministic, and contrary to the case of e.g. seminegative (disjunctive-free) programs, $\mathcal{W}_P^\omega(\emptyset)$ may not be a model, even if it is consistent.

Corollary 2. *Let P be a choice logic program . If $\mathcal{W}_P^\omega(\emptyset)$ is a total interpretation, then it is the unique stable model of P.*

[8] The fixpoint may not exist because $\mathcal{W}_P^n(I)$ may not be consistent, i.e. outside of the domain of $\mathcal{W}_P$, for some $n > 0$.

The following monotonically decreasing operator can be used to check the unfounded-free property of total interpretations.

Definition 9. *Let P be a choice logic program and let I be an interpretation for it. The operator $\mathcal{R}_{P,I} : 2^{\mathcal{B}_P} \rightarrow 2^{\mathcal{B}_P}$ is defined by*

$$\mathcal{R}_{P,I}(X) = \left\{ a \in X \mid \begin{array}{ll} \exists r : A \oplus a \leftarrow B \in P \cdot A \cap I \neq \emptyset \wedge B \subseteq I & or \\ \exists \leftarrow B, a \in P \cdot B \subseteq I & or \\ \forall r : A \oplus a \leftarrow B \cdot \overline{B} \cap (\neg I \cup X) \neq \emptyset, \, or & \\ \qquad (A \cup \{a\}) \cap B \neq \emptyset & \end{array} \right\}$$

Intuitively, $\mathcal{R}_{P,I}(J)$ gathers all atoms that are contained in both J and some unfounded set of I.

Proposition 9. *Let I be a total interpretation for a choice logic program P. Then, $\mathcal{R}^{\omega}_{P,I}(I^+) = \emptyset$ iff I is unfounded-free.*

Moreover $\mathcal{R}_{P,I}$ can be used to compute the greatest unfounded sets $\mathcal{GUS}_P(I)$.

Proposition 10. *Let P be a choice logic program and let I be an interpretation for it. Then, $\mathcal{R}^{\omega}_{P,I}(\mathcal{B}_P) = \mathcal{GUS}_P(I)$.*

The above result does not hold for disjunctive logic programs.

3 Computing Stable Models

With the help of the above results, an intuitive and relatively efficient "backtracking fixpoint" algorithm can be designed to compute the stable models of a choice logic program.

Essentially, the algorithm of Fig. 3 keeps a "current interpretation" (which is initialized to the empty set) and a stack of choice points (initially empty). It consists of a loop which itself consists of two stages:

1. In the first stage, $\mathcal{W}_P$ is applied on the current interpretation until a fixpoint interpretation is reached or an inconsistency is detected. In the latter case, the algorithm backtracks to the previous choice point (if any) and tries a different choice.
2. In the second stage, a choice is made from the applicable rules (that have a true body in the current interpretation) that are not yet applied. If there are no such rules, the current interpretation is a stable model. For the selected rule, a choice is made for a literal from the head to be added to the current interpretation, thus making the rule applied (the choice must be such that the new interpretation remains consistent). The other literals are immediately assumed false. Such a combination of literals is called is a "possibly-true conjunction"[5]. We use $PT_P(I)$ to denote the set of such choices that are available, given the interpretation I.

Given the results of the previous section, it is clear that this algorithm will find all stable models of a given choice logic program. It generalizes on a corresponding algorithm in [5] because it also handles constraints. In addition, it can afford to be more skeptical than the algorithm in [5] (checking consistency at each step in stage 1) because of Proposition 4.

Input: A choice logic program P.
Output: The stable models of P.

Procedure *Compute-Stable*(I_n:SetOfLiterals);
var X, I'_n, I'_{n+1} : SetOfLiterals;
begin
if $PT_P(I_n) = \emptyset$ (* no choices available *)
then output "I_n is a stable model of P";
else for each $X \in PT_P(I_n)$ **do**
$I'_{n+1} := I_n \cup X$; (* Assume the truth of a possibly-true conjunction *)
repeat
$I'_n := I'_{n+1}$;
$I'_{n+1} := \mathcal{T}_P(I'_n) \cup \neg\mathcal{R}^{\omega}_{P,I'_n}(\mathcal{B}_P)$; (* $= \mathcal{W}_P(I'_n)$ *)
until $I'_{n+1} = I'_n$ or $I'_{n+1} \cap \neg I'_{n+1} \neq \emptyset$;
if $I'_{n+1} \cap \neg I'_{n+1} = \emptyset$ (* I'_{n+1} is consistent *)
then *Compute-Stable*(I'_{n+1})
end-if
end-for
end-if
end-procedure

var I,J : SetOfLiterals;
G : SetOfAtoms;
begin (*Main *)
$I := \emptyset$;
repeat (* Computation of $\mathcal{W}^{\omega}_P(\emptyset)$ if it exists *)
$J := I$;
$G := \mathcal{GUS}_P(J)$; (* by means of $\mathcal{R}^{\omega}_{P,J}(\mathcal{B}_P)$ *)
if $G \cap J \neq \emptyset$ (* J not unfounded-free *)
exit
end-if;
$I := \mathcal{T}_P(J) \cup \neg G$; (* $= \mathcal{W}_P(J)$ *)
until $I = J$;
if $PT_P(I) = \emptyset$
then output "I is the unique stable model of P";
else *Compute-Stable*(I)
end-if
end.

Fig. 3. Algorithm for the Computation of Stable Models for choice logic programs.

4 Conclusions and Directions for Further Research

We introduced choice logic programs as a convenient and simple formalism for modeling decision making. Such programs can e.g. be used to model strategic games. We investigated the implicit support for negation that is present in such programs, due to the exclusive nature of the choices and the support for constraints. It turns out that choice programs can reasonably simulate seminegative

logic programs. On the other hand, many results that are known for (semi-negative) disjunctive programs (without constraints) can be carried over (or even strengthened) to choice programs (with constraints), resulting in a simple algorithm to compute the stable models of a choice program.

It is worth noting that, although [1] introduces constraints for disjunctive logic programs, these are checked only after the usual algorithm (for programs without constraints) finishes, while our algorithm uses constraints directly, which should result in a more eager pruning of candidate interpretations.

Future research will attempt to extend the notion of choice programs to allow for the expression of epistemic restrictions. At present, all the knowledge of decision making agents is stored in a single program which is visible to each agent (this fact lies at the basis of Theorem 1); an assumption which is often not realistic.

References

1. Francesco Buccafurri, Nicola Leone and Pasquale Rullo. Strong and Weak Constraints in Disjunctive Datalog. In Jurgen Dix and Ulrich Furbach and Anil Nerode, editors, *4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, volume 1265 of Lecture Notes in Computer Science, pages 2–17. Springer.
2. Marina De Vos and Dirk Vermeir. Forcing in Disjunctive Logic Programs. In Kamal Karlapalem, Amin Y. Noaman and Ken Barker, editors, *Proceedings of the Ninth International Conference on Information and Computation*, pages 167–174, Winnipeg, Manitoba, Canada, June 1998.
3. Marina De Vos and Dirk Vermeir. Choice logic programs and Nash equilibria in strategic games. Accepted at *Annual Conference of the European Association for Computer Science Logic (CSL99)*, September 20-25, 1999, Madrid, Spain. Published in *Lecture Notes in Computer Science*, Springer.
4. Allen Van Gelder, Kenneth A. Ross and John S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the Association for Computing Machinery*, **38(3)** (1991) 620–650.
5. Nicola Leone, Pasquale Rullo and Francesco Scarello. Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Journal of Information and Computation* **135(2)** (1997) 69–112.
6. M. J. Osborne and A. Rubinstein. *A Course in Game Theory*, MIT Press,1994.
7. Carolina. Ruiz and Jack. Minker. Computing Stable and Partial Stable Models of Extended Disjunctive Logic Programs. *Lecture Notes in Computer Science*, **927**(1995). Spinger
8. D. Sacca. Deterministic and Non-Deterministic Stable Models. *Logic and Computation*, **5** (1997) 555–579.
9. Chiaki Sakama and Katsumi Inoue An Alternative Approach to the Semantics of Disjunctive Logic Programs and Decductive Databases. *Journal of Automated Reasoning*, **13** (1994) 145–172.
10. D. Sacca and C. Zaniolo. Stable Models and Non-Determinism for Logic Programs with Negation. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 205-218. Association for Computing Machinery, 1990.