A new semantics for logic programs
capturing the stable model semantics: the
extension semantics

Belaïd Benhamou and Pierre Siegel

Université Aix-Marseille, CMI Laboratoire des Sciences de l'Information et des
Systèmes (équipe INCA, LSIS)

JFPC’2012
Motivation

- Non-monotonic reasoning is human, very known in Artificial intelligence

- The stable model semantics [Gelfond and Lifschitz 88] is an operational and rational semantics for logic programming. How to capture it in a logic approach?

- Is it possible to extend it?

- SAT-based answer set systems works on the Clark[78] completion and usually need to add for non tight logic programs during search what is called loops formulas. This could be heavy and space consuming.

- Is it possible to get a more simple answer set procedure?
Outline

- Background on Logic programs: positive and general (normal) programs
- The stable model semantics.
- A weak negation as failure
- The notion of extension
- A CNF encoding for logic program.
- The extension semantics for general programs
- A new Answer set procedure based on the extension semantics.
- Conclusion and perspectives.
Background on Logic programs: positive programs

Positive logic programs: a positive logic program $\pi$ is a set of rules of the form $r : L_0 \leftarrow L_1, \ldots, L_m$, ($m \geq 0$) where $L_i$ ($0 \leq i \leq m$) is an atom.

In other words a positive program is a logic program that does not contain classic negations and negations as failure.

Here, we have $head(r) = L_0$ and $body(r) = L_1, \ldots, L_m$.

The meaning of the previous rule is “if we can prove the body $L_1, L_2, \ldots, L_m$ then we will conclude the head $L_0$”.

A positive logic program $\pi$ contains one canonical Herbrand model which is the single minimal Herbrand model $CM(\pi)$.
Background on Logic programs: positive programs

- The minimal Herbrand model of $\pi$ is the least set of atoms that is closed with respect to $\pi$.

- Formally, for a given logic program $\pi$ and a set of atoms $A$, the operator $T_\pi(A) = \{head(r)/r \in \pi, body(r) \subseteq A\}$ compute all the atoms that can be deduced from $A$ by using the rules of $\pi$.

- Consider the suite $T_\pi^0= T_\pi(\emptyset)$, $T_\pi^{k+1} = T_\pi(T_\pi^k)$, $\forall k \geq 0$.

- The canonical model is $CM(\pi) = \bigcup_{k \geq 0} T_\pi^k$.

- The minimal Herbrand model $CM(\pi)$ includes all the atoms that can be deduced from $\pi$.

- This model is exactly the minimal model expressed by a Prolog program formed by the rules of $\pi$ or the minimal model of the Horn clauses.
Background on Logic programs: general (normal) programs

- **General programs**: A general logic program $\pi$ is a set of rules of the form $r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n$, $(0 \leq m < n)$ where $L_i$ $(0 \leq i \leq n)$ are atoms, and $not$ is the symbol expressing negation as failure.

- Intuitively the rule $r$ means "if we can prove all of $\{L_1, L_2, \ldots, L_m\}$ and we can not prove any of $\{L_{m+1}, \ldots, L_n\}$, then we deduce $L_0$"

- The positive body of $r$ is denoted by $body^+(r) = \{L_1, L_2, \ldots, L_m\}$, and the negative body by $body^-(r) = \{L_{m+1}, \ldots, L_n\}$.

- The sub-rule $r^+ : L_0 \leftarrow L_1, L_2, \ldots, L_m$ expresses the positive projection of the rule $r$. 
Background on Logic programs: the stable model semantics

The reduct of the program $\pi$ with respect to a given set $A$ of atoms is the positive program $\pi^A$ where we delete each rule containing an expression $\text{not}L_i$ in its negative body such that $L_i \in A$ and where we delete the other expressions $\text{not}L_i$ in the bodies of the other rules.

More precisely, $\pi^A = \{ r^+/r \in \pi, \text{body}^-(r) \cap A = \emptyset \}$.

A set of atoms $A$ is a stable model (an answer set) of $\pi$ if and only if $A$ is identical to the minimal Herbrand model of $\pi^A$, that is if only if $A = CM(\pi^A)$.

The stable model semantics is based on the world closed assumption (WCA), an atom that is not in the stable model $A$ is interpreted to false.
A weak negation as failure

- Our semantics follows a logic approach that concerns a propositional logic (not modal) language $L$ where we distinguish two types of variables: a subset of classical variables $V = \{L_i : L_i \in L\}$ and another subset $nV = \{notL_i : notL_i \in L\}$.

- For each variable in $L_i \in V$ there exists a corresponding variable $notL_i \in nV$.

- We use the symbol $notL_i$ to express a weak negation as failure for logic programming.

- At this level both kind of variables $L_i$ and $notL_i$ are independent. That is, there is no semantics relation defined between both of them.
A weak negation as failure

This weak negation as failure semantics gives the relationship between both kind of variables in the framework of logic programming.

This is expressed by adding to the propositional logic language $L$ the set of negative clauses $ME = \{ (\neg L_i \lor \neg notL_i) : L_i \in V \}$.

This is a pseudo axiom which should be applied on the pairs of literals $L_i$ and $notL_i$.

The set $ME$ expresses only the mutual exclusion between $L_i$ and $notL_i$, but does not establish the equivalence between $\neg L_i$ and $notL_i$.

We can have $\neg notL_i$ without having $L_i$.

We can have both $\neg L_i$ and $\neg notL_i$ at the same time.
A weak negation as failure

- Intuitively, we can consider here that a literal $not L_i$ expresses a weak negation as failure.

- This weakness renders our approach original and different from almost all of the well-known semantics.

- Even the world closed assumption used in default logic did not capture this weakened negation.

- To obtain a classic negation as failure, it is sufficient to add to the logic system a property (an inference rule) that infers $L_i$ each time $\neg not L_i$ is deduced.
The notion of extension

Definition: Given a set of formulas $F$ of $L$, a sub-set $S$ of $nV$ and a sub-set $S'$ of $S$, an extension of $(F \cup ME, S)$ is a set $E = (F \cup ME) \cup S'$ such that the following conditions hold:

1. $E$ is consistent.
2. $\forall notL_i \in S - S', E \cup \{notL_i\}$ is inconsistent

Example 1: Let $F = \{(notb \land c) \rightarrow a, a \rightarrow b, notd \rightarrow c, a\}$ be a set of formulas of $L$, $ME = \{-a \lor \neg nota, \neg b \lor \neg notb, \neg c \lor \neg notc, \neg d \lor \neg notd\}$ and $S = \{notb, notd\}$ a sub-set of variables of $nV$. The pair $(F \cup ME, S)$ admits a single extension $E = (F \cup ME) \cup \{notd\}$. 
The notion of extension

- **Example 2:** Let $F = \{ (\neg b \land c) \rightarrow a, a \rightarrow b, \neg d \rightarrow c \}$ be a set of formulas of $L$,
  
  $ME = \{ \neg a \lor \neg \text{not}a, \neg b \lor \neg \text{not}b, \neg c \lor \neg \text{not}c, \neg d \lor \neg \text{not}d \}$, and
  
  $S = \{ \text{not}b, \text{not}d \}$ a sub-set of variables of $nV$. The pair
  
  $(F \cup ME, S)$ admits two extensions $E = (F \cup ME) \cup \{ \text{not}d \}$ and
  
  $E = (F \cup ME) \cup \{ \text{not}b \}$.

- **Proposition 1:** Let $F$ be a set of formulas of $L$ and $S$ a sub-set of $nV$. If $(F \cup ME)$ is consistent, then there exists at least an extension of $(F \cup ME, S)$ for-all sub-set $S \subset nV$.

- **Proposition 2:** If $F$ is a set of clauses that contain at least a positive literal of $V$ and do not include any positive literal $\text{not}L_i$ of $nV$, then the set of clauses $F \cup ME$ is consistent.
A CNF encoding for logic program.

- In our approach, each rule \( r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n, \) \((0 \leq m < n)\) of \( \pi \) is expressed by the propositional logic formula (clause) \( c : L_0 \lor \neg L_1 \lor \neg L_2, \ldots, \neg \lor L_m \lor \neg notL_{m+1} \ldots \neg notL_n. \)

- The set of literals \( STB = \{ notL_i : notL_i \in \pi \} \subset nV \) appearing in \( \pi \) is the its strong backdoor.

- A program \( \pi = \{ r : L_0 \leftarrow L_1, L_2, \ldots, L_m, notL_{m+1}, \ldots, notL_n \}, \) \((0 \leq m < n)\), is then expressed by its CNF encoding in \( L \):

\[
L(\pi) = \bigcup_{r \in \pi} (L_0 \lor \neg L_1 \lor \ldots, \neg \lor L_m \lor \neg notL_{m+1} \ldots \neg notL_n) \\
\bigcup_{L_i \in V} (\neg L_i \lor \neg notL_i)
\]

- The size of \( L(\pi) \) is at most equal to \( size(\pi) + 2 \times n \), where \( n \) is the number of variable of \( \pi \).
The extension semantics

Definition: Given the logical encoding $L(\pi)$ of a logic program $\pi$, its strong backdoor $STB$, and a sub-set $S' \subset STB$, then $E = L(\pi) \cup S'$ is an extension of $(L(\pi), STB)$ if the following conditions hold:

1. $E$ is consistent.
2. $\forall notL_i \in STB - S', E \cup \{notL_i\}$ is inconsistent

Definition: Consider the set $F$ of Example 1 as rules of a general logic program $\pi$. We give below both $\pi$ and its logic encoding $L(\pi) = L(\pi) - Rules \cup L(\pi) - ME$:

<table>
<thead>
<tr>
<th>$\pi$</th>
<th>$L(\pi)$-Rules</th>
<th>$L(\pi)$-ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \leftarrow c, notb$</td>
<td>$a \lor \neg c \lor \neg notb$</td>
<td>$\neg a \lor \neg nota$</td>
</tr>
<tr>
<td>$b \leftarrow a$</td>
<td>$b \lor \neg a$</td>
<td>$\neg b \lor \neg notb$</td>
</tr>
<tr>
<td>$c \leftarrow notd$</td>
<td>$c \lor \neg notd$</td>
<td>$\neg c \lor \neg notc$</td>
</tr>
<tr>
<td>$a \leftarrow$</td>
<td>$a$</td>
<td>$\neg d \lor \neg notd$</td>
</tr>
</tbody>
</table>

The STB set is $STB = \{notb, notd\}$ and pair $(L(\pi), STB)$ has a single extension $E = L(\pi) \cup \{notd\}$. 

A new semantics for logic programs... – JFPC’12 – p. 14
The extension semantics

Example 4: Take the logic program \( \pi = \{ a \leftarrow c, \neg b ; b \leftarrow a ; c \leftarrow \neg d ; a \leftarrow \} \) of Example 3 and drop the rule \( a \leftarrow \). We obtain the program \( \pi' = \pi - \{ a \leftarrow \} \) whose CNF encoding is \( L(\pi') = L(\pi) - \{ a \} \) and whose STB is the same the one of \( \pi \). There is two extensions for \( (L(\pi'), STB) \): \( E_1 = L(\pi') \cup \{ \neg d \} \) and \( E_2 = L(\pi) \cup \{ \neg b \} \).

Theorem: If \( X \) is a stable model (an answer set) of a logic program \( \pi \), then there exist an extension \( E \) of \( (L(\pi), STB) \) such that \( X = \{ L_i \in V : E \models L_i \} \) and which verify the discrimination condition \((\forall L_i \in V, E \models \neg \neg L_i \Rightarrow E \models L_i)\).

Example 5: Take the program \( \pi \) of Example 3 and its CNF encoding \( L(\pi) \). \( \pi \) has one stable model \( X = \{ a, b, c \} \) which matches with the single extension \( E = L(\pi) \cup \{ \neg d \} \) of \( (L(\pi), STB) \). After the assignment of the STB variables, we use unit resolution to show that \( E \models \{ a, b, c, \neg d, \neg \neg a, \neg \neg b, \neg \neg c \} \). We can see that \( E \) verifies the condition \((\forall L_i \in V, E \models \neg \neg L_i \Rightarrow E \models L_i)\) and \( X = \{ L_i \in V : E \models L_i \} = \{ a, b, c \} \).
The extension semantics

- **Remark 1:** $\pi$ is a general logic program $\Rightarrow$ an extension $E$ of $(L(\pi), STB)$ is a consistent set of Horn clauses $\Rightarrow$ all the positive literals $L_i$ that could be deduced form $E$ are inferred by unit resolution.

- For instance the literals $X = \{a, b, c\}$ forming the stable model of example 5 are inferred by unit resolution. This is important for reducing the complexity of the algorithm which compute the extensions that we present in Section 4.

- **Remark 2:** By using Proposition 1, we deduce that for each consistent logic program $L(\pi)$, there always exists an extension of $(L(\pi), STB)$. But, there is not always a stable model for $\pi$. It could exists some extra-extension of $(L(\pi), STB)$ which do not match any stable model of $\pi$.

- These extensions $E$ are exactly those that do not satisfy the discriminant condition.

- These extensions could be very important for knowledge representation, but are not captured by the stable models semantics. Our approach, extends the stable model semantics in this sens.
The extension semantics

- **Example 6:** Let \( \pi = \{ a \leftarrow nota \} \). \( \pi \) has no answer set. Its CNF encoding is \( L(\pi) = \{ a \vee \neg nota, \neg a \vee \neg nota \} \) and its \( STB = \{ nota \} \). The pair \((L(\pi), STB)\) has an extra-extension \( E = L(\pi) \) such that \( E \models \neg nota \), but \( E \not\models a \). \( E \) does not verify the discrimination condition of Theorem 1, then \( E \) does not match any stable model of \( \pi \). Our semantics captures the stable models semantics for this example.

- **Example 7:** Now take the program \( \pi = \{ a \leftarrow nota, a \leftarrow \} \). \( \pi \) has a single stable model \( X = \{ a \} \). Its corresponding CNF encoding is \( L(\pi) = \{ a \vee \neg nota, \neg a \vee \neg nota, a \} \) and its \( STB = \{ nota \} \). The pair \((L(\pi), STB)\) has an extension \( E = L(\pi) \), such that \( E \models \neg nota \) and \( E \models a \). \( E \) verifies the discrimination condition of theorem 1, and entails the single stable model \( X = \{ a \} \) of \( \pi \). Our semantics captures the stable models semantics for this example too.

- **Theorem 2:** If \( E \) is an extension of \((L(\pi), STB)\), such that the discrimination condition \((\forall L_i \in V, E \models \neg not L_i \Rightarrow E \models L_i)\) holds, then \( X = \{ L_i : E \models L_i \} \) is an answer set of \( \pi \).
The extension semantics

**Example 8:** Consider the program $\pi = \{a \leftarrow \neg b, b \leftarrow \neg a\}$. Its CNF encoding is $L(\pi) = \text{Rules} \cup \text{ME}$ where $\text{Rules} = \{a \lor \neg \neg b, b \lor \neg \neg a\}$ and where $\text{ME} = \{\neg a \lor \neg \neg a, \neg b \lor \neg \neg b\}$ and its $\text{STB} = \{\neg a, \neg b\}$. The pair $(L(\pi), \text{STB})$ has two extensions $E_1 = L(\pi) \cup \{\neg a\}$ and $E_2 = L(\pi) \cup \{\neg b\}$ such that $E_1 \models \{b, \neg a, \neg \neg b\}$ and $E_2 \models \{a, \neg b, \neg \neg a\}$. Both extensions verify the discrimination condition ($\forall L_i \in V, E \models \neg \neg L_i \Rightarrow E \models L_i$). We obtain from $E_1$ resp. $E_2$ the sets $X_1 = \{b\}$ resp. $X_1 = \{a\}$ which are the two stable models of the program $\pi$.

**Example 9:** Now consider $\pi = \{a \leftarrow \neg b, b \leftarrow \neg c, c \leftarrow \neg a\}$. Its CNF encoding is $L(\pi) = \text{Rules} \cup \text{ME}$ where $\text{Rules} = \{a \lor \neg \neg b, b \lor \neg \neg c, c \lor \neg \neg a\}$ and where $\text{ME} = \{\neg a \lor \neg \neg a, \neg b \lor \neg \neg b, \neg c \lor \neg \neg c\}$ and its $\text{STB} = \{\neg a, \neg b, \neg c\}$. The pair $(L(\pi), \text{STB})$ has three extensions $E_1 = L(\pi) \cup \{\neg a\}$, $E_2 = L(\pi) \cup \{\neg b\}$ and $E_3 = L(\pi) \cup \{\neg c\}$ such that $E_1 \models \{\neg a, c, \neg \neg c, \neg \neg b\}$, $E_2 \models \{\neg b, a, \neg \neg a, \neg \neg c\}$ and $E_3 \models \{\neg c, b, \neg \neg b, \neg \neg a\}$. We can see that all of the extensions do not verify the discrimination condition ($\forall L_i \in V, E \models \neg \neg L_i \Rightarrow E \models L_i$).
A procedure for extension and stable models

**Proposition:** Consider a program $\pi$, its CNF encoding is $L(\pi)$ and $E$ an extension of $(L(\pi), STB)$ where all the STB variable are assigned and where all the unit resolution reductions (subsumption of clauses and suppression of variables) are done, then:

1. The set $E$ resulting from the simplification is formed by the union of a subset of unit clauses $C_1$ and a subset of non-unit Horn clauses $C_2$ without STB variables. The sets $C_1$ and $C_2$ do not have common variables.

2. The assignment to false of all the remain variables \{not$L_i$\} and \{L$_i$\} which are not already fixed in $C_1$ does not affect the answer sets of the program $\pi$, and leads to the single minimal model $M$ of $E$ ($E$ is a set of Horn clauses). This minimal model $M$ is made by the part of positive literals that are entailed by the extension $E$ (by unit resolution). $M$ is a candidate for model stable checking.

3. The candidate $M$ is a stable model of $\pi$ if the corresponding extension $E$ of $(L(\pi), STB)$ verifies the discriminant condition
\[
(\forall L_i \in V, E \models \neg \text{not}L_i \Rightarrow E \models L_i).
\]
A procedure for extension and stable models

The method

1. Set \( E = L(\pi) \).

2. Assign the STB variables by using a DLL method where all the unit resolution propagations and consistency checking are done at each choice node of the search tree. Unit resolution is sufficient to maintain consistency at each node. We also mark at each node of the tree the variables whose assignment propagates negative literals \( \neg notL_i \) of the STB. Assign the value true in prior to the free STB variables and when a failure is detected the method backtracks only on the nodes of the STB that had been marked. After assigning all STB variables, we get automatically a maximal consistent partial interpretation \( I_{STB} \). The obtained set \( E = L(\pi) \cup I_{STB} \) is an extension of \( (L(\pi), STB) \).

3. Since \( E \) is an extension of \( (L(\pi), STB) \) by construction (step 1 and step 2), assign all the remain variables to false in order to get the minimal model \( M \) of \( E \).

4. If \( E \) verifies the discrimination condition, then the restriction to the positive atoms that are entailed by \( E \) forms an answer set of the program \( \pi \). Otherwise, \( M \) is an extra-answer set of \( \pi \).
A procedure for extension and stable models

- Our method has the advantage to perform on the CNF encoding $L(\pi)$ whose size is equal to $\text{size}(\pi) + 2n$ rather on the completion of $\pi$ and works in a constant space complexity.
- It can be used for tight and non-tight programs and is able to compute all the answer sets of the program $\pi$.
- It does not need to manage the heavy loop formulas set that SAT-based answer set solvers use.
- This method can be implemented by a slightly modified DLL procedure which performs an enumeration on the STB variables and maintains a unit resolution saturation process at each node of the search.
- This is followed by a simple and basic answer set verification step (step 4).
- Beside, when the discriminant condition does not hold our method could compute extra-stable models

**Complexity:** If $n$ is the number of the variables of $L(\pi)$, $k$ is the number of STB variables, and $m$ the number of clauses of $L(\pi)$, then the complexity of the algorithm is $O(nm2^k)$ with $k \leq n$ in the worst case.
Conclusion

Work done:
- We introduced the extension semantics which captures and extends the stable model semantics.
- We defined a weak negation as failure on which is based the extension semantics.
- We defined a new answer set procedure which simplifies the family SAT-based answer set systems.

Future works:
- We are first interested in implementing this new method and compare its performances to the ones of existing methods for answer sets computing.
- We are interested in extending this work to other frameworks like extended and disjunctive logic programs, and to more general frameworks like the default logic.