The Well-Founded Semantics

—

Characterizations and Computation

Stefan Brass

University of Hildesheim
On leave from: University of Hannover

Based on joint work with:
Jürgen Dix, Ulrich Zukowski, Burkhard Freitag

Introduction (1)

Nonmonotonic Negation:

- Prolog's „Negation as Failure“:
  If A is not provable, assume **not** A as proven.
- The specified positive knowledge is complete (everything else is false).

Example:

book(1, "Ullman", "DBS").
book(2, "Lloyd", "LP").

borrowed(1).

available(Author, Title) $\leftarrow$
    book(Book, Author, Title) $\wedge$
    not borrowed(Book).

NOT is useful/necessary:

- Already the specification of finite relations (as in relational databases) is quite complicated in first order logic.
- The transitive closure cannot be defined in first order logic.

Introduction (2)

Problem:

- There are about 20 proposals for the exact semantics of nonmonotic negation.
- Which one is natural and free of surprises?
- Are there good semantics which we do not know yet?
- Efficient computation.

Stratified Programs:

- Semantics of negation is clear, but stratified programs are not enough in practice.
- Negation is a special case of aggregation: "bill of materials"-Problem not stratified.
- The SQL3 standard proposal requires stratification, but IBM DB2 allows more.
- "Runtime stratification" inconvenient.

Abstract Semantics

Semantics for Logic Programs:

- A semantics is a mapping $\mathcal{S}$, which assigns to every program P the set of derivable positive and negative ground literals.
- $\mathcal{S}(P) = \mathcal{S}\big(\text{ground}(P)\big)$.
- If $A \leftarrow \text{true} \in P$, then $A \in \mathcal{S}(P)$.
- If A is not ground instance of any rule head, then **not** $A \in \mathcal{S}(P)$.

Program-Transformation:

- A program-transformation is a relation $\mapsto$ between ground logic programs.
- A semantics $\mathcal{S}$ allows a transformation $\mapsto$ iff

$$P_1 \mapsto P_2 \implies \mathcal{S}(P_1) = \mathcal{S}(P_2).$$

A Normal Form (1)

Deletion of Tautologies:

$P_1 \mapsto_T P_2$ iff P_1 contains a rule of the form

$$A \leftarrow \ldots \land A \land \ldots,$$

and P_2 is the result of deleting this rule from P_1.

Unfolding (Partial Evaluation):

- Replace a positive body literal B by the bodies of all rules about B.
- P_1:

$$\begin{aligned} p &\leftarrow q \land \textbf{not}\, r. \\ q &\leftarrow s \land \textbf{not}\, t. \\ q &\leftarrow u. \end{aligned}$$

- P_2:

$$\begin{aligned} p &\leftarrow s \land \textbf{not}\, t \land \textbf{not}\, r. \\ p &\leftarrow u \land \textbf{not}\, r. \\ q &\leftarrow s \land \textbf{not}\, t. \\ q &\leftarrow u. \end{aligned}$$

A Normal Form (2)

Deletion of Nonminimal Rules:

- A rule $A \leftarrow L_1 \wedge \cdots \wedge L_n$ can be deleted if there is another rule $A \leftarrow L_{i_1} \wedge \cdots \wedge L_{i_k}$ such that $\{L_{i_1}, \ldots, L_{i_k}\} \subset \{L_1, \ldots, L_n\}$.

Normal Form:

P_0 is a normal form of P wrt $\mapsto$ iff

- $P \mapsto^* P_0$ and
- there is no P_1 with $P_0 \mapsto P_1$.

Theorem:

- The rewriting system $\mapsto$ consisting of the above three transformations is terminating, i.e. every program has a normal form.
- The rewriting system $\mapsto$ is also confluent (if $P_1 \mapsto^* P_2$ and $P_1 \mapsto^* P_3$, then there is P_4 such that $P_2 \mapsto^* P_4$ and $P_3 \mapsto^* P_4$).
- So every program has a unique normal form.

Conditional Facts (1)

Conditional Fact:

Ground rule with only negative body literals:

$$A \leftarrow \textbf{not}\, B_1 \wedge \cdots \wedge \textbf{not}\, B_n.$$

Direct Consequence Operator T_P:

$$\mathsf{p(a)} \leftarrow \textbf{not}\, \mathsf{s(b)} \wedge \textbf{not}\, \mathsf{r(b)}.$$

$$\uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

$$\boxed{\mathsf{p(X)} \leftarrow \mathsf{q_1(X)} \wedge \mathsf{q_2(X,Y)} \wedge \textbf{not}\, \mathsf{r(Y)}.}$$

$$\uparrow \qquad\qquad \uparrow$$

$$\mathsf{q_1(a)} \qquad \mathsf{q_2(a,b)} \leftarrow \textbf{not}\, \mathsf{s(b)}.$$

Theorem:

lfp(T_P) (without nonminimal cond. facts) is exactly the normal form of ground(P).

Conditional Facts (2)

Example:

book(1, "Ullman", "DBS").
book(2, "Lloyd", "LP").

borrowed(1).

available(Author, Title) $\leftarrow$
    book(Book, Autor, Titel) $\wedge$
    not borrowed(Book).

Normal Form:

book(1, "Ullman", "DBS").
book(2, "Lloyd", "LP").

borrowed(1).

available("Ullman", "DBS") $\leftarrow$
    not borrowed(1).
available("Lloyd", "LP") $\leftarrow$
    not borrowed(2).

Relation to Minimal Models

Model:

- Set I of positive and negative ground literals
- satisfying the rules.

Order Among the Models:

$I_1 \prec I_2$ iff

- $I_1 \subset I_2$, but
- I_1 and I_2 contain the same negative literals.

Theorem:

- A semantics $\mathcal{S}$ allows unfolding, elimination of tautologies and of nonminimal rules iff
- $\mathcal{S}(P_1) = \mathcal{S}(P_2)$ for all programs P_1 and P_2, which have the same set of minimal models.

WFS-Characterization (1)

Positive Reduction:

Replace a rule of the form

$$A \leftarrow L_1 \wedge \cdots \wedge L_{i-1} \wedge \textbf{not}\, B \wedge L_{i+1} \wedge \cdots \wedge L_n,$$

where B occurs in no rule head, by

$$A \leftarrow L_1 \wedge \cdots \wedge L_{i-1} \wedge L_{i+1} \wedge \cdots \wedge L_n.$$

Negative Reduction:

Delete a rule of the form

$$A \leftarrow L_1 \wedge \cdots \wedge \textbf{not}\, B \wedge \cdots \wedge L_n,$$

where $B \leftarrow$ true is given as a fact.

Theorem:

Also the rewriting system extended by these two transformations is terminating and confluent.

WFS-Characterization (2)

Residual Program:

The normal form of a program P is called the residual program res(P) of P.

Example:

book(1, “Ullman”, “DBS”).
book(2, “Lloyd”, “LP”).

borrowed(1).

available(“Ullman”, “DBS”) $\leftarrow$
  not borrowed(1).
available(“Lloyd”, “LP”) $\leftarrow$
  not borrowed(2).

Residual Program:

book(1, “Ullman”, “DBS”).
book(2, “Lloyd”, “LP”).

borrowed(1).

available(“Lloyd”, “LP”).

WFS-Characterization (3)

Example:

$\mathsf{odd(X)} \leftarrow \mathsf{succ(Y,X)} \wedge \textbf{not}\,\mathsf{odd(Y)}.$

$\mathsf{succ(0,1)}.$
$\mathsf{succ(1,2)}.$
$\ldots$
$\mathsf{succ}(n-1,n).$

Derivable Conditional Facts:

$\mathsf{odd(1)} \leftarrow \textbf{not}\,\mathsf{odd(0)}.$
$\mathsf{odd(2)} \leftarrow \textbf{not}\,\mathsf{odd(1)}.$
$\mathsf{odd(3)} \leftarrow \textbf{not}\,\mathsf{odd(2)}.$
$\ldots$

Residual Program:

$\mathsf{odd(1)}.$
$\mathsf{odd(3)}.$
$\ldots$
$\mathsf{succ(0,1)}.$
$\mathsf{succ(1,2)}.$
$\ldots$
$\mathsf{succ}(n-1,n).$

WFS-Characterization (4)

Example:

$p \leftarrow$ **not** p.

Theorem:

The well-founded semantics allows the above five transformations.

Theorem:

The well-founded model of P can be directly read from the residual program res(P):

- A is true in the well-founded model iff res(P) contains the fact $A \leftarrow$ true.
- A is false in the well-founded model iff res(P) contains no rule about A.
- All other ground atoms are undefined in the well-founded model.

WFS-Characterization (5)

Weaker Semantics:

A semantics $\mathcal{S}_1$ is weaker than (or equal to) a semantics $\mathcal{S}_2$ iff for all programs P:

$$\mathcal{S}_1(P) \subseteq \mathcal{S}_2(P).$$

Theorem:

The WFS is the weakest semantics which allows the above five transformations.

Remarks:

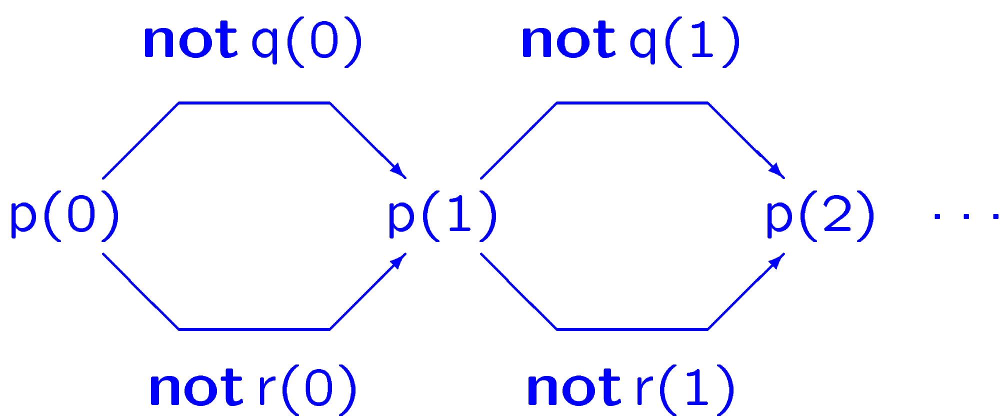
- There is such a weakest semantics for any set of transformations.
- Another parameter is the basic definition of a semantics. E.g. one can require that a semantics yields a set of models.

Delaying Positive Literals (1)

Problem:

The residual program can grow to exponential size:

$$
\begin{aligned}
&\mathsf{p}(0).\\
&\mathsf{p}(X) \leftarrow \mathsf{p}(Y) \wedge \mathsf{succ}(Y,X) \wedge \textbf{not}\ \mathsf{q}(Y).\\
&\mathsf{p}(X) \leftarrow \mathsf{p}(Y) \wedge \mathsf{succ}(Y,X) \wedge \textbf{not}\ \mathsf{r}(Y).\\
&\mathsf{q}(X) \leftarrow \mathsf{succ}(X,Y) \wedge \textbf{not}\ \mathsf{q}(X).\\
&\mathsf{r}(X) \leftarrow \mathsf{succ}(X,Y) \wedge \textbf{not}\ \mathsf{r}(X).\\
&\mathsf{succ}(0,1).\\
&\mathsf{succ}(1,2).\\
&\dots\\
&\mathsf{succ}(n-1,n).
\end{aligned}
$$



Delaying Positive Literals (2)

Solution:

- "Unfolding" is too powerful.
- Delay also the positive body literals (as in Chen/Warrens's SLG-Resolution).

Generalized Conditional Facts:

- Let $\bar{T}_P(F)$ be the set of ground instances

$$A\theta \leftarrow L_1\theta \wedge \cdots \wedge L_n\theta$$

  of rules in P, such that for every positive L_i there is a rule instance about $L_i\theta$ in F.
- "Intelligent Grounding"

Delaying Positive Literals (3)

Example:

book(1, “Ullman”, “DBS”).
book(2, “Lloyd”, “LP”).

borrowed(1).

available(“Ullman”, “DBS”) $\leftarrow$
    book(1, “Ullman”, “DBS”) $\wedge$
    not borrowed(1).
available(“Lloyd”, “LP”) $\leftarrow$
    book(2, “Lloyd”, “LP”) $\wedge$
    not borrowed(2).

“Success” (Simplification):

Replace a rule of the form

$$A \leftarrow L_1 \wedge \cdots \wedge L_{i-1} \wedge B \wedge L_{i+1} \wedge \cdots \wedge L_n,$$

where $B \leftarrow$ true is given as a fact, by

$$A \leftarrow L_1 \wedge \cdots \wedge L_{i-1} \wedge L_{i+1} \wedge \cdots \wedge L_n.$$

Delaying Positive Literals (4)

"Failure":

Delete a rule of the form

$$A \leftarrow L_1 \wedge \cdots \wedge B \wedge \cdots \wedge L_n,$$

where B does not appear in any rule head.

Remark:

The four transformations Success, Failure, positive and negative Reduction together correspond to the Fitting operator.

Example:

These transformations are not sufficient for computing the well-founded model:

$$\begin{array}{lll} p. & & \\ q & \leftarrow & \textbf{not}\, p. \\ q & \leftarrow & r. \\ r & \leftarrow & q. \end{array}$$

Loop Check (1)

Elimination of Positive Loops:

Let $\mathcal{A}$ be a set of ground atoms such that

For all rules $A \leftarrow \mathcal{B}$ in P:
If $A \in \mathcal{A}$, then $\mathcal{B} \cap \mathcal{A} \neq \emptyset$.

Then delete all rules $A \leftarrow \mathcal{B}$ with

$$\mathcal{B} \cap \mathcal{A} \neq \emptyset.$$

Implementation of Loop Check:

- The maximal $\mathcal{A}$ consists of all facts which are not derivable even if one assumes that all negative body literals are true.
- Can be computed in polynomial time.

Lemma:

- If a semantics allows unfolding and elimination of tautologies, it also allows loop check.
- ground$(P) \mapsto_L$ lfp$(\bar{T}_P)$.

Program Remainder

Theorem:

- The rewriting system consisting of these transformations (Success, Failure, pos/neg Reduction, Loop Elimination) is again terminating and confluent.
- We call the normal form under this rewriting system the "program remainder" of P.

Theorem:

- The program remainder is equivalent to the original program under WFS, STABLE, and may other semantics.
- The program remainder can be computed in polynomial time.
- The well-founded model can be read from the program remainder as from the residual program.
- The remainder of P results from the ground instantiation of P by evaluating all body literals known in WFS(P).

WFS-Computation (1)

Remark:

In order to turn a transformation system into an algorithm, one needs to specify

- in which order the transformations are applied
- which data structures are used to represent the conditional facts.

Strongly Connected Components:

- Partition program into sets of mutual recursive rules (or single nonrecursive rules).
- Do computation componentwise (in some topological order wrt the dependencies).

Componentwise Grounding:

- Like the above intelligent grounding, but only for a single component, and
- body literals defined in lower components and having a definite truth value are evaluated.

WFS-Computation (2)

Lemma:

After this intelligent grounding, an explicit application of "loop check" is only needed if a predicate in the component depends on itself positively as well as through negation.

Alternating Fixpoint:

Compute possibly true and surely true facts in alternating sequence.

Comparison:

- AFP reduces the bodies of the conditional facts to one bit and recomputes them when needed.
- We can simulate AFP (using loop check + negative reduction and success + positive reduction in alternating sequence).
- We beat AFP when components contain only negative recursion (like in the "odd number" example).

Conclusions (1)

The WFS is Important:

- Stratified programs are not enough, Runtime-stratification also problematic.
- The WFS has a unique model and is computable in polynomic time.
- The WFS is really very simple.
- Support for arbitrary programs under the WFS is announced for XSB and LOLA.

Comparison with Stable Semantics:

- In the stable semantics, non stratified negation is really used to specify problems which are beyond polynomial complexity.
- WFS = runtime stratification plus localized error messages.

Conclusions (2)

Computation:

- The presented method is faster than the alternating fixpoint procedure.
- It is much simpler to understand than SLG-resolution (however, it is not goal-directed).

Future Work:

- Complexity: quadratic or maybe linear?
- Extension to aggregations.
- Combination with SLDMagic technique.
- Construction of bottom-up machine with support for WFS and using DB techniques.