REVISE

Carlos Viegas Damásio
UNINOVA
Portugal

Wolfgang Nejdl
RWTH Aachen
Germany

Luís Moniz Pereira
UNINOVA
Portugal

To appear in KR'94
A semantics for logic programs with explicit negation (WFSX) was introduced in 1992 by Pereira and Alferes (ECAI’92). It stems from the observation that:

“Normal Logic Programs have no way of representing explicitly negated assertions, i.e. negative conclusions.”

Basically, WFSX follows from WFS [Gelder et al., 1991] plus one basic “coherence” requirement relating explicit and default negation:

¬L entails ∼ L for any literal L.
Logic Programming Basis (2)

- Objective Literal: an atom $A$ or its explicit negation $\neg A$.

- Default Literal: $\sim L$, $L$ an objective literal.

An extended logic program is a set of rules

$$H \leftarrow B_1, \ldots, B_n, \sim C_1, \ldots, \sim C_m$$

and integrity rules of the form

$$\bot \leftarrow B_1, \ldots, B_n, \sim C_1, \ldots, \sim C_m$$

with $m \geq 0, n \geq 0$, and $H, B_i, C_j$ objective literals.

For each atom $A$, the integrity rule $\bot \leftarrow A, \neg A$ is implicit in the program.
Because of integrity rules and explicit negation, extended logic programs may be contradictory. We handle contradiction by revising CWAs to true, by adding facts:

**Revisables** Let $\mathcal{R}_P$ be the set of all default literals $\sim A$ with no rules for $A$ in $P$. The revisable literals of $P$ are a subset of $\mathcal{R}_P$. A set $S \subseteq \sim \mathcal{R}_P$ is a set of positive assumptions.

**Revision of a program** A set of positive assumptions $A$ of $P$ is a revision of $P$ iff $(P \cup A) \not\models \bot$ (i.e. satisfies all integrity rules).

Without loss of generality, we can restrict ourselves to consider as revisables default literals for which there are no rules in the program.
Advantages of LP/Diagnosis Combination

Advantages for Diagnosis/Debugging Application:

- Clear semantics

- Enhance system representations by including meta (strategic) knowledge

- Greater applicability by using this multi-level representation architecture for both diagnosis and debugging applications
• Extend state of the art in model-based diagnosis in the areas
  – Hierarchical representations
  – Strategic knowledge (working with and switching between multiple representations and views)
  – Representation of monitoring, diagnosis and repair views

• Extend state of the art in debugging of logic programs and logic specification
Advantages for Extended Logic Programming:

- Test expressivity and adequacy of various semantics against a well-known application domain (diagnosis)

- Test against real-world examples like power distribution networks (as described in last year's IEEE Expert and AI-EDAM articles (in cooperation with Siemens))

- Extend or modify semantics and language of logic programming system as suggested by possible limitations discovered during in the diagnosis/debugging domains
Global preferences are used in diagnosis to:

- Represent preference relation between possible models;

- Enforce abstract properties that influence what can be derived after adding assumptions and new facts: transitivity, antisymmetry and irreflexivity (partial order), modularity, linearity, etc.

- Define arbitrary preferences for choosing models, switching between different views and representations.
Logic programs are not expressive enough to represent preferences among sets of literals. To express global preferences, i.e. preferences over the order of revisions, we use a directed acyclic and/or graph of the form:

\[ \text{Level}_0 \ll \text{Level}_1, \ldots, \text{Level}_n \quad (1) \]

The associated revisables of level \( L \) are denoted by \( \mathcal{R}(L) \).

The intuitive meaning of a set of preference rules like (1) for some \( \text{Level}_0 \) is

“\( \text{I’m willing to consider the Level}_0 \) revisions as good solutions only if for some rule body its levels have been considered and there are no revisions at any of those levels.”
The revisions of the special preference level bottom preference level are preferred to all other ones. So, level bottom must be accessible from every node in the preference graph:

**Preferred revisions:** Let $P$ be an ELP, and $\Pi$ a preference graph containing a preference level $Lev$.

The revision $R$ is preferred wrt $\Pi$ iff $R$ is a minimal revision of $P$ (in the sense of set inclusion), using revisables $\mathcal{R}(Lev)$, and there is no other preference level below with a revision.
Example (1)

Basic Representation

\[ \text{inv}(G, I, 1, T) \leftarrow \text{not} \ ab(G), \text{node}(I, 0, T). \]
\[ \text{inv}(G, I, 0, T) \leftarrow \text{not} \ ab(G), \text{node}(I, 1, T). \]

\[ \text{node}(b, B, T) \leftarrow \text{inv}(g1, a, B, T). \]
\[ \text{node}(c, C, T) \leftarrow \text{inv}(g2, b, C, T). \]

Fault Models

\[ \text{inv}(G, \_ , 0, T) \leftarrow ab(G), s\_at\_0(G'). \]
\[ \text{inv}(G, \_ , 1, T) \leftarrow ab(G), s\_at\_1(G'). \]

\[ s\_at\_0(G') \leftarrow \text{fault\_mode}(G, s0). \]
\[ s\_at\_1(G') \leftarrow \text{fault\_mode}(G, s1). \]
\[ \text{unknown}(G) \leftarrow \text{fault\_mode}(G, \text{unknown}). \]
Integrity Constraints

% Exclusive fault modes
← fault_mode(G, S1), fault_mode(G, S2), S1 ≠ S2.

% Exclusive values
← node(X, V1), node(X, V2), V1 ≠ V2.

% Assignment of fault modes
← ab(G),
not fault_mode(G, s0),
not fault_mode(G, s1),
not fault_mode(G, unknown).
Reasoning Modes

Basic Idea:

- Focus reasoning by concentrating on probable failures (simple views, high abstraction level etc.) first, to avoid reasoning in too large detail.

In this example, we’ll prefer single faults to multiple faults (i.e. more than one component is abnormal), fault mode “stuck at 0” to “stuck at 1” and the latter to the “unknown” fault mode.
Example (4)

One possible combination of these two preferences is expressed using the following integrity rules and preference graph:

\[ \bot \leftarrow \text{fault\_mode}(\_ , s1), \sim s0\_impossible. \]
\[ \bot \leftarrow \text{fault\_mode}(\_ , \text{unknown}), \sim s0\_impossible. \]
\[ \bot \leftarrow \text{fault\_mode}(\_ , \text{unknown}), \sim s1\_impossible. \]
\[ \bot \leftarrow \text{fault\_mode}(G1, F1), \text{fault\_mode}(G2, F2),
G2 \neq G1, \sim \text{single\_fault\_impossible}. \]

1 \ll \text{bottom} \qquad 2 \ll \text{bottom}

3 \ll 1 \qquad 4 \ll 1 \land 2

5 \ll 3 \land 4

\[ R(\text{bottom}) = \{ \sim ab(\_), \sim \text{fault\_mode}(\_ , \_) \}, \]
\[ R(1) = R(\text{bottom}) \cup \{ \sim s0\_impossible \}, \ldots \]
Preference Graph

- ~s0_impossible
- ~s1_impossible
- ~single_fault_impossible
- ~ab(_)
- ~fault_mode(_,_)

1 2 3 4 5

- ~s0_impossible
- ~s1_impossible
- ~ab(_)
- ~fault_mode(_,_)

- ~s0_impossible
- ~s1_impossible
- ~ab(_)
- ~fault_mode(_,_)

- ~s0_impossible
- ~ab(_)
- ~fault_mode(_,_)

bottom

- ~ab(_)
- ~fault_mode(_,_)
First Experiment:

\[ \text{node}(a, 0, t0) \quad \text{node}(b, 0, t0) \quad \text{node}(c, 0, t0) \]

Preferred revision at level 2:

\[ \{ ab(g1), ab(g2), \text{fault\_mode}(g1, s0), \text{fault\_mode}(g2, s0), \text{single\_fault\_impossible} \} \]
Second Experiment:

node(a, 0, t1) node(b, 1, t1) node(c, 0, t1)

Preferred revisions at level 5:

\[
\{ \text{s0_impossible, s1_impossible, single_fault_impossible, } \\
\text{ab(g1), fault_mode(g1, unknown), } \\
\text{ab(g2), fault_mode(g2, s0)} \} \\
\}

\[
\{ \text{s0_impossible, s1_impossible, } \\
\text{single_fault_impossible, } \\
\text{ab(g1), fault_mode(g1, unknown), } \\
\text{ab(g2), fault_mode(g2, unknown)} \} \\
\}
\]
Computing Preferred Revisions

In the paper we provide an algorithm to compute preferred revisions based on Reiter’s algorithm [Reiter, 1987].

The main new features of the algorithm are:

- Handles arbitrary sets of assumptions;

- Handles different sets of revisables at different levels;

- Provides a general caching mechanism for the computation of minimal “hitting sets” obtained at previous levels.

The revision algorithm co–routines with the theorem prover.
Example (7)

\[
\begin{align*}
\{\neg ab(g1)\} \\
\text{ab}(g1) \\
\{ab(g1),\neg fm(g1,s0),\neg fm(g1,s1),\neg fm(g1,uk)\} \\
\text{fm}(g1,s0) & \quad \text{fm}(g1,s1) \\
\{ab(g1),\text{fm}(g1,s0)\} & \quad \{ab(g1),\text{fm}(g1,s1)\} \\
\text{X} & \quad \text{X} \\
\{ab(g2),\neg fm(g2,s0),\neg fm(g2,s1),\neg fm(g2,uk)\} \\
\text{fm}(g2,uk) & \quad \text{fm}(g2,s0) & \quad \text{fm}(g2,s1) \\
\{\text{fm}(g1,uk),\text{fm}(g2,uk),\neg sfi\} & \quad \{\text{fm}(g1,uk),\neg s0i\} & \quad \{\text{fm}(g2,uk),\text{fm}(g2,s0),\neg sfi\} \\
\text{bottom, 2} & \quad \text{s0i} & \quad \text{sfi} \\
1,3 & \quad \text{sfi} & \quad \{\text{fm}(g1,uk),\text{fm}(g2,s0),\neg sfi\} \\
\text{4} & \quad \text{sfi} \\
\{\text{fm}(g1,uk),\neg s1i\} & \quad \{\text{fm}(g1,uk),\neg s1i\} \\
\text{s1i} & \quad \text{s1i} \\
\text{5} & \quad \text{\checkmark} & \quad \text{\checkmark}
\end{align*}
\]
Comparisons
Future Work

- Extend the preference graph language in order to represent structural refinement, behaviour refinement, and other strategies of [Böttcher and Dressler, 1994];

- Develop a meta-language for our revision framework, based on the *demo* predicate, where the preference graph can be “programmed”.

- Find more efficient algorithms based on the REVISE algorithm and the SLX proof procedure for WFSX [ADP,ECAI’94]

- Provide mechanisms to implement, “easily”, special preference relations.