The SLP System: An Implementation of Super Logic Programs

Stefan Brass
Technical University of Clausthal, Germany

In part joint work with:
Jürgen Dix, Teodor C. Przymusinski

Super Logic Programs (1)

- Arbitrarily nested propositional connectives.
  - ◇ Includes disjunctive logic programs and
  - ◇ integrity constraints (rules with empty head).
- Rule form not required: Clear separation between default negation and logical/classical negation.
- Variables.
  - ◇ Allowedness: Every variable must appear at least once in negated context outside default negation (i.e. a positive body literal).

Super Logic Programs (2)

- Default negation can be used only in negated context (i.e. in the body).
  - ◇ The meaning of **not** is defined by a nonmonotonic semantics, and not by rules of the program.
- The static semantics permits $\wedge$ and $\vee$ inside **not**.
  - ◇ **not** $(\varphi_1 \vee \varphi_2)$ is equivalent to **not** $\varphi_1 \wedge$ **not** φ_2.
  - ◇ This does not hold for conjunction: **not** means falsity in all minimal models.

Probably no real expressiveness extension: Define new predicate.

SLP Interpreter (1)

Negation Semantics Currently Supported:

- Static Semantics: Query evaluation (disj. answers).
- Stable Models: Only model computation.

Future Plans (Easy Extensions):

- Query evaluation (cred./skept.) for stable models
- D-WFS (Brass/Dix)
- More efficient minimal model computation for programs without default negation.

SLP Interpreter (2)

- Written in C++, currently 21.000 lines of code
- Source code available under GNU public license
- `http://purl.oclc.org/NET/slp/`
  `http://www.informatik.uni-giessen.de/staff/brass/slp/`
- Web interface, local installation not necessary.

  Runs as CGI-Program. In development: Own HTTP server.
- Alternative: Classical console interface.

Basic Steps

- Transformation into clauses (rules and cond. facts).
- Computation of implied conditional facts (hyperresolution, does grounding).
- Evaluation of **not** in obvious cases: residual prog.
- For static semantics: Iterative computation of static interpretations for remaining default negations, evaluation of conditional disjunctive answers.
- For stable models: Model computation of a disjunctive version of Clark's completion.

Overview

1. Computation of Residual Program
2. Static Semantics
3. Minimal Model Computation
4. Query Evaluation
5. Stable Models

Conditional Facts

- Rule without positive body literals, i.e. of the form
  $$\mathsf{p(a,b)} \vee \mathsf{q(c)} \leftarrow \textbf{not}\, \mathsf{r(d)} \wedge \textbf{not}\, \mathsf{s(e,f)}.$$
- Because of the allowedness restriction, variables cannot appear.
- For the static semantics, conditional facts can contain negations of the form **not** $(\mathsf{q(a)} \wedge \mathsf{q(b)})$.

Hyperresolution

- Generalization of T_P-operator to conditional facts.
- Disjunctive context or condition of a fact matched with a body literal is appended to the derived fact:

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

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

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

$$\uparrow \qquad \uparrow$$

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

Hyperresolution Fixpoint

- Theorem: Let F be the least fixpoint of the hyperresolution operator for a program P. A Herbrand interpretation I is a minimal model of F iff it is a minimal model of the ground instantiation P* of P.
- Minimal model: Minimal given a fixed interpretation for the default negation literals.
- Theorem: If a semantics permits unfolding and elimination of tautologies, F is equivalent to P*.

Duplicate Elimination (1)

- Duplicate conditional facts must be detected in order to guarantee termination.
- Also non-minimal conditional facts are eliminated immediately in SLP.
- E.g. p(a) $\vee$ p(b) $\leftarrow$ **not** r(a) $\wedge$ **not** r(b) is eliminated if there is also the stronger fact p(a) $\leftarrow$ **not** r(b).
- Since the first conditional fact is implied by the second, this does not change the models.

Duplicate Elimination (2)

Algorithm (Elimination of Non-Minimal CFs):

```
Let a cond. fact C with m literals be generated.
Get new comparison number N (inc. counter);
for each literal L in C do
    for each existing CF C' that contains L do
        if C'.ComparNo ≠ N then
            C'.Overlap := 1; C'.ComparNo := N;
        else increment C'.Overlap;
        if C'.Overlap = C'.length then
            C is redundant
        else if C'.Overlap = m then
            C' is redundant
```

Duplicate Elimination (3)

- Seminaive evaluation: At least one body literal must be matched with a new conditional fact (derived in the previous iteration).
- Possible Improvements (not yet implemented):
  - ◇ It is possible to make the resolvable literal in a disjunctive fact unique (reduces the number of ways to derive the same disjunction).
  - ◇ It is possible to determine an evaluation sequence for the rules and iterate only selected rules.

Residual Program (1)

- Positive Reduction: If q appears in no head, then **not** q is obviously true, and e.g.

  $$p \leftarrow \textbf{not}\, q \wedge \textbf{not}\, r$$

  can be strengthened to $p \leftarrow \textbf{not}\, r$.
- Negative Reduction: If there is an unconditional fact $q \vee r$, then **not** q and **not** r cannot both be true, and e.g.

  $$p \leftarrow \textbf{not}\, q \wedge \textbf{not}\, r \wedge \textbf{not}\, s$$

  can be deleted.

Residual Program (2)

- The residual program is constructed from the hyperresolution fixpoint by evaluating default negation in obvious cases, i.e. applying positive and negative reduction (and elimination of non-minimal CFs).
- It is equivalent to the original program under e.g. the static and the stable model semantics.
- Positive reduction can be efficiently implemented with an counter for occurrences in the head and a linked list to occurrences in the body, negative reduction uses the above overlap technique.

Overview

1. Computation of Residual Program
2. Static Semantics
3. Query Evaluation
4. Stable Models

Note

- The current algorithm is quite inefficient, but
  - ◇ As far as I know, SLP is still the only implementation of the static semantics.

    A direct application of the definition of the static semantics is impossible. But further improvements of the algorithm are possible.
  - ◇ The static semantics is a very well-behaved generalization of the WFS to disjunctive programs.
  - ◇ Normally, very few negations remain in the residual program. The computation can be restricted to these "critical negations".

Static Model Computation (1)

- Static interpretations for the critical negations are computed as follows:
  - ◇ At first, all possible interpretations for the critical negations are tried.
  - ◇ For each such interpretation, the conditions of the conditional facts are evaluated, and minimal models of the remaining disjunctions are computed (only for ground literals that appear in negations).

Static Model Computation (2)

- An interpretation I of the default negations is only possible if there is a set of minimal models $J_1, \ldots, J_n$ such that for all atoms p, $I \models \mathbf{not}\, p$ iff $J_1 \models \neg p$ and ... and $J_n \models \neg p$ (and if I can be extended to a model).
- This condition can be checked without trying all subsets of the current set of minimal models:
  - ◇ One selects all models J_i that make no p true for which **not** p is true in I.
  - ◇ Then one only has to check that for each **not** p that is false in I, there is J_i with $J_i \models p$.

Static Model Computation (3)

- This condition restricts the possible interpretations for the default negation literals.
- But then the set of minimal models is reduced.
- And so on, until a fixpoint is reached.

Possible Improvement (not yet implemented):

- It is possible to avoid considering all possible interpretations for the default negation atoms at the beginning (one can start directly to construct minimal models).

Example (1)

- Consider the following program:

$$\begin{aligned} p \lor q &\leftarrow \textbf{not}\, r. \\ q &\leftarrow \textbf{not}\, q. \\ r &\leftarrow q. \end{aligned}$$

- Let the query be **not** p.
- The residual program is:

$$\begin{aligned} p \lor q &\leftarrow \textbf{not}\, r. \\ q &\leftarrow \textbf{not}\, q. \\ r &\leftarrow \textbf{not}\, q. \\ r \lor p &\leftarrow \textbf{not}\, r. \\ \$\text{answer} &\leftarrow \textbf{not}\, p. \end{aligned}$$

Example (2)

- Residual program (again):

$$\begin{aligned} p \lor q &\leftarrow \textbf{not}\, r. \\ q &\leftarrow \textbf{not}\, q. \\ r &\leftarrow \textbf{not}\, q. \\ r \lor p &\leftarrow \textbf{not}\, r. \\ \$\text{answer} &\leftarrow \textbf{not}\, p. \end{aligned}$$

- If **not** q and **not** r are both false, p, q and r must be false in a minimal model.
- If **not** q is true, q and r must be true, and p false.
- And so on.

Example (3)

- Minimal models
  (reduced to ground atoms appearing in negations):

No	p	q	r
1	false	false	false
2	true	false	false
3	false	true	true

- Still possible interpretations for default negations:

No	**not** p	**not** q	**not** r
1	true	true	true
2	false	true	true
3	true	false	false
4	false	false	false

Example (4)

- This reduces the set of minimal models (minimal model 2 is based on the interpretation that makes **not** r true and **not** q false: no longer possible).

No	p	q	r
1	false	false	false
3	false	true	true

- Remaining interpretations for the default atoms:

No	**not** p	**not** q	**not** r
1	true	true	true
3	true	false	false

Example (5)

- This means that **not** p is true in all static models.

$$\text{\$answer} \leftarrow \textbf{not}\ p.$$

- D-WFS (Brass/Dix) and the well-founded circumscriptive semantics (You/Yuan) do not imply **not** p in this example.

$$\begin{aligned} p \vee q &\leftarrow \textbf{not}\ r. \\ q &\leftarrow \textbf{not}\ q. \\ r &\leftarrow q. \end{aligned}$$

- Under the static semantics the given rule $r \leftarrow q$ implies **not** $r \rightarrow$ **not** q.

Overview

Minimal Model Generator (1)

- Input is a set $\mathcal{D}$ of minimal positive disjunctions.
- For each ground atom p that should be assigned a truth value (appears in critical negation):
  - ◇ If p does not appear in $\mathcal{D}$: p is false.
  - ◇ If p appears as a fact in $\mathcal{D}$: p is true.
  - ◇ If p appears in n proper disjunctions, $n + 1$ cases are considered (with backtracking):
    p is false or
    p is true required by one of the n disjunctions.

Minimal Model Generator (2)

- If e.g. the disjunction $p \vee q \vee r$ was selected to explain that p must be true, the ground atoms q and r are made false.
- Assumed truth values are used to the update the disjunctions:
  - ◇ True ground atoms are asserted as disjunction, which eliminates all disjunctions that contain the ground atom (they are non-minimal).
  - ◇ False ground atoms are removed from the disjunctions (which makes them stronger).

Minimal Model Generator (3)

Positive Aspects of the Algorithm:

- Never runs into dead ends.
- No additional minimality test required.
- Can generate partial minimal models.
- Space complexity polynomial in the size of the input disjunctions.

Negative Aspects:

- The same model may be generated more than once.

Overview

1. Computation of Residual Program
2. Static Semantics
3. Minimal Model Computation
4. Query Evaluation
5. Stable Models

Query Evaluation (1)

- A query ψ with answer variables $X_1, \ldots, X_n$ is added to the program as the formula

$$\$\text{answer}(X_1, \ldots, X_n) \leftarrow \psi.$$

- All derived conditional facts that directly or indirectly used this rule contain $answer in the head.
- Since the rule does not really belong to the program, these conditional facts must be ignored when
  - ◇ doing positive reduction and when
  - ◇ computing minimal models of the program.

Query Evaluation (2)

- Disjunctive answers are computed from conditional facts in the residual program that contain only $answer-literals in the head:
  - ◇ One simply checks whether the condition is true in all static interpretations.
  - ◇ In many cases, default negation was already evaluated by positive and negative reduction, so no condition remains in the residual program.

Overview

1. Computation of Residual Program
2. Static Semantics
3. Minimal Model Computation
4. Query Evaluation
5. Stable Models

Stable Models (1)

- Theorem: The transformation into the residual program does not change the set of stable models.
- Theorem: Let P be a disjunctive program without positive body literals. A Herbrand interpretation I is a stable model of P iff it is a model of Clark's completion comp(P).
- Clark's completion of a disjunctive program: Treat

  $p \vee q \leftarrow \textbf{not}\, r$ like $p \leftarrow \textbf{not}\, q \wedge \textbf{not}\, r.$
  $\qquad\qquad\qquad\qquad\quad q \leftarrow \textbf{not}\, p \wedge \textbf{not}\, r.$

Stable Models (2)

- SLP constructs Clark's completion as
  - ◇ a set of pure positive disjunctions (the rules),
  - ◇ and a set of pure negative disjunctions (the completion axioms).

    Only ground atoms are considered that still appear in the residual program (all other ground atoms are certainly false).
- E.g. $p \leftarrow \textbf{not}\, q,\ q \leftarrow \textbf{not}\, p$ has the completion $p \vee q,\ \neg p \vee \neg q$.
- Duplicate and non-minimal disjunctions are eliminated as before.

Stable Models (3)

Model Generation:

- If a positive or negative disjunction consists only of one element, the corresponding truth value is immediately assigned to the ground atom.
- Then the model generator loops over all ground atoms that still appear in the residual program.
- If an atom is not yet assigned a truth value, the model generator considers both possibilities (with backtracking).

Stable Models (4)

- Assigned truth values are used to update the disjunctions:
  - ◇ The current assumption is added as one-element disjunction. Eliminates non-minimal disjunctions.
  - ◇ The complement of the assumption is removed from the disjunctions (they become stronger).
- This might make the truth value of other ground atoms obvious. If an atom is assigned both true and false, an inconsistency is detected.

Conclusion

- SLP is the first implementation of the static semantics and one implementation of disjunctive stable models (with constraints).
- I plan to continue the work on SLP (HTTP server, more semantics, completion of documentation).
- For simple examples, the efficiency is more than sufficient.
- There are ideas for improving the efficiency, but that would require some motivation (e.g. users).