Stable-Unstable Semantics: Beyond NP with Normal Logic Programs

Bart Bogaerts\textsuperscript{1,2}, Tomi Janhunen\textsuperscript{1}, Shahab Tasharrofi\textsuperscript{1}

1) Aalto University, Finland
2) KU Leuven, Belgium

Computational Logic Day 2016, December 8, 2016
Background: Disjunctive Logic Programs (DLPs)

- An extension of normal logic programs in terms of proper disjunctive rules [Gelfond and Lifschitz, 1991]:
  \[ h_1 \lor \cdots \lor h_l \leftarrow a_1 \land \cdots \land a_n \land \neg b_1 \land \cdots \land \neg b_m. \]

- The main decision problems of DLPs are either \( \Sigma_2^P \)- or \( \Pi_2^P \)-complete [Eiter and Gottlob, 1995].

- A number of native answer set solvers that implement the search for answer sets in the disjunctive case:
  - DLV [Leone et al., 1998/2006]
  - GNT [J. et al., 2000/2006]
  - CMODELS [Giunchiglia et al., 2006]
  - CLASP D [Drescher et al., 2008]

- The underlying (co)NP-oracle can only be accessed in an indirect way, e.g., using saturation or meta programming.
Background: Saturation

- A positive disjunctive program $\mathcal{P}$ can be embedded in a DLP as an oracle by including
  - the rule $u \leftarrow \neg u$ for a new atom $u$ not occurring in $\mathcal{P}$,
  - the rule $u \lor h_1 \lor \cdots \lor h_l \leftarrow a_1 \land \cdots \land a_n$ for each rule of $\mathcal{P}$, and
  - the rule $a \leftarrow u$ for each atom of $\mathcal{P}$.

- The atoms in $\mathcal{P}$ and $u$ form a single strongly connected component (SCC) that cannot be shifted.

- It is impossible to exploit default negation in the oracle as pointed out by [Eiter and Polleres, 2006].

- It is also quite difficult to detect and maintain oracles of the form above in existing encodings.
Background: Meta Interpretation

▶ Meta interpretation renders disjunctive rules as data
  [Eiter and Polleres, 2006; Gebser et al. 2011]:

\[
\begin{align*}
r : \ h_1 \lor \cdots \lor h_l & \leftarrow a_1 \land \cdots \land a_n \land \neg b_1 \land \cdots \land \neg b_m. \\
\rightarrow & \begin{cases}
\text{head}(r, h_1). & \ldots \text{head}(r, h_l). \\
\text{pbody}(r, a_1). & \ldots \text{pbody}(r, a_n). \\
\text{nbody}(r, b_1). & \ldots \text{nbody}(r, b_m).
\end{cases}
\end{align*}
\]

▶ The semantics of rules can be tailored using meta rules:

\[
\begin{align*}
\text{in}(H) & \leftarrow \text{head}(R, H) \land \\
\text{in}(P) : \text{pbody}(R, P) \land \\
\neg \text{in}(N) : \text{nbody}(R, N) \land \\
\neg \text{in}(OH) : \text{head}(R, OH) : OH \neq H.
\end{align*}
\]

▶ Second-order features can be expressed via saturation.
Our Approach

- A new way of combining (normal) logic programs so that
  - the interface for oracles is made explicit and
  - the semantics is defined in terms of stable-unstable models.

Distinguished features:
- All variables are quantified implicitly (no prenex form)!
- A proof-of-concept implementation is readily obtained in the SAT-TO-SAT framework [J. et al., 2016].
- The entire PH can be covered using the idea recursively.
Outline

Combined Logic Programs

Stable-Unstable Semantics

Applications

Implementation

Beyond Second Level with Normal Logic Programs

Conclusion
Logic Programs: Syntax and Semantics

- A (normal) logic program $P$ over a signature $\sigma$ may have a set of parameters $\tau \subseteq \sigma$ not occurring in the heads of rules.

- An interpretation $M \subseteq \sigma$ of $P$ is
  1. a stable model of $P$, iff $M$ is a $\subseteq$-minimal model of the Gelfond-Lifschitz reduct $P^M$, and
  2. a parameterized stable model of $P$, iff $M$ is a stable model of the program $P \cup \{a \leftarrow | a \in \tau \cap M\}$.

Example
Consider the following program $P$ parameterized by $\tau = \{c\}$:

$$a \leftarrow b \land c. \quad b \leftarrow c. \quad b \leftarrow a \land \neg c. \quad a \leftarrow \neg c.$$ 

Then $M_1 = \{a, b, c\}$ and $M_2 = \{a, b\}$ are stable given $\tau$. 
A combined logic program is a pair \((\mathcal{P}_g, \mathcal{P}_t)\) of normal logic programs \(\mathcal{P}_g\) and \(\mathcal{P}_t\) with vocabularies \(\sigma_g\) and \(\sigma_t\) such that

1. the generating program \(\mathcal{P}_g\) is parameterized by \(\tau_g \subseteq \sigma_g\) and
2. the testing program \(\mathcal{P}_t\) is parameterized by \(\sigma_g \cap \sigma_t\).

**Example**

Consider the following combined logic program \((\mathcal{P}_g, \mathcal{P}_t)\):

\[
\begin{align*}
\{y_1, n_1, y_2, n_2\} & \\
y_1 & \leftarrow \neg x_1. \\
n_1 & \leftarrow \neg p_1. \\
y_2 & \leftarrow \neg x_2. \\
n_2 & \leftarrow \neg p_2. \\
\{x_1, p_1, x_2, p_2\} & \\
\end{align*}
\]

\[
\begin{align*}
f_1 & \leftarrow \neg y_1 \land n_1 \land t_x. \\
f_2 & \leftarrow \neg y_2 \land n_2 \land t_x. \\
f_1 & \leftarrow \neg y_1 \land \neg n_1 \land f_x. \\
f_2 & \leftarrow \neg y_2 \land \neg n_2 \land f_x. \\
f_1 & \leftarrow y_1 \land n_1 \land t_y. \\
f_2 & \leftarrow y_2 \land n_2 \land t_y. \\
f_1 & \leftarrow y_1 \land \neg n_1 \land f_y. \\
f_2 & \leftarrow y_2 \land \neg n_2 \land f_y. \\
f & \leftarrow f_1 \land f_2. \\
t_x & \leftarrow \neg f_x. \\
t_y & \leftarrow \neg f_y. \\
f & \leftarrow \neg f. \\
f_x & \leftarrow \neg t_x. \\
f_y & \leftarrow \neg t_y. \\
\{y_1, n_1, y_2, n_2\} & \\
\end{align*}
\]
Stable-Unstable Semantics

Let \((\mathcal{P}_g, \mathcal{P}_t)\) be a combined logic program with vocabularies \(\sigma_g\) and \(\sigma_t\).

A interpretation \(I \subseteq \sigma_g\) is a stable-unstable model of \((\mathcal{P}_g, \mathcal{P}_t)\) iff the following two conditions hold:

1. \(I\) is a parameterized stable model of \(\mathcal{P}_g\) with respect to \(\tau_g\) (the parameters of \(\mathcal{P}_g\)) and
2. there is no parameterized stable model \(J\) of \(\mathcal{P}_t\) that coincides with \(I\) on \(\sigma_t \cap \sigma_g\) (i.e., such that \(I \cap \sigma_t = J \cap \sigma_g\)).

Example

For the combined program

\[
\mathcal{P}_g: \quad a \leftarrow \neg b. \ b \leftarrow \neg a. \\
\mathcal{P}_t: \quad c \leftarrow a, \neg c.
\]

the only stable-unstable model is \(M = \{a\}\).
Example

\[
\begin{align*}
\{y_1, n_1, y_2, n_2\} \\
y_1 &\leftarrow \neg x_1. \\
n_1 &\leftarrow \neg p_1. \\
y_2 &\leftarrow \neg x_2. \\
n_2 &\leftarrow \neg p_2. \\
\{x_1, p_1, x_2, p_2\}
\end{align*}
\]

\[
\begin{align*}
\{t_x, f_x, t_y, f_y, f_1, f_2, f\} \\
f_1 &\leftarrow \neg y_1 \land n_1 \land t_x. \\
f_2 &\leftarrow \neg y_2 \land n_2 \land t_x. \\
f_1 &\leftarrow \neg y_1 \land \neg n_1 \land f_x. \\
f_2 &\leftarrow \neg y_2 \land \neg n_2 \land f_x. \\
f_1 &\leftarrow y_1 \land n_1 \land t_y. \\
f_2 &\leftarrow y_2 \land n_2 \land t_y. \\
f_1 &\leftarrow y_1 \land \neg n_1 \land f_y. \\
f_2 &\leftarrow y_2 \land \neg n_2 \land f_y. \\
f &\leftarrow f_1 \land f_2. \\
t_x &\leftarrow \neg f_x. \\
t_y &\leftarrow \neg f_y. \\
f &\leftarrow \neg f. \\
f_x &\leftarrow \neg t_x. \\
f_y &\leftarrow \neg t_y.
\end{align*}
\]

\[
\begin{align*}
\{y_1, n_1, y_2, n_2\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Clause</th>
<th>(M_i)</th>
<th>Stable models given (M_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x \lor x)</td>
<td>({x_1, p_1, x_2, p_2})</td>
<td>({f_x, f_y, f_1, f_2, f}, {f_x, t_y, f_1, f_2, f})</td>
</tr>
<tr>
<td>(x \lor \overline{x})</td>
<td>({x_1, p_1, x_2, n_2})</td>
<td>–</td>
</tr>
<tr>
<td>(x \lor y)</td>
<td>({x_1, p_1, y_2, p_2})</td>
<td>({f_x, f_y, f_1, f_2, f})</td>
</tr>
<tr>
<td>(x \lor \overline{y})</td>
<td>({x_1, p_1, y_2, n_2})</td>
<td>({f_x, t_y, f_1, f_2, f})</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\[
\{x_1, p_1, x_2, n_2\}, \{x_1, n_1, x_2, p_2\}, \{y_1, p_1, y_2, n_2\}, \{y_1, n_1, y_2, p_2\}.
\]
Results

- Any disjunctive program $\mathcal{P}$ can be rewritten as a combined logic program $(\mathcal{P}_g, \mathcal{P}_t)$ as done by GNT [J. et al., 2006].

- We call a combined logic program $(\mathcal{P}_g, \mathcal{P}_t)$ independent, if $\sigma_g \cap \sigma_t = \emptyset$, i.e., $\mathcal{P}_g$ and $\mathcal{P}_t$ cannot interact with each other.

- Deciding the existence of a stable-unstable model for a finite combined program $(\mathcal{P}_g, \mathcal{P}_t)$ is
  1. $\Sigma^P_2$-complete in general, and
  2. $D^P$-complete for independent combined programs.
Encodings

- **Winning strategies for parity games**
  - Correspond to model checking problems in $\mu$-calculus.
  - Plays are infinite paths in a graph.
  - Existing encodings in difference logic [Heljanko et al., 2012] can be improved to be linear.

- **Conformant planning**
  - Certain facts about the initial state and/or the actions’ effects are unknown.
  - The native ASP encoding of [Leone et al., 2001] can now be expressed without saturation.

- **Points of no return in formula-labeled graphs**
  - New prototypical problem that combines graphs and logic.
Points of No Return

Based on a directed multigraph $G = (V, A, s)$:
- $V$ is a set of vertices,
- $s \in V$ is an initial vertex, and
- $A$ is a set of arcs $u \xrightarrow{\phi} v$ labeled by Boolean formulas $\phi$.

The criteria for a point of no return:

$\phi_1 \wedge \cdots \wedge \phi_n \in \text{SAT but } \phi_1 \wedge \cdots \wedge \phi_{n+m} \in \text{UNSAT (always)}$.

In general, it is a $\Sigma_2^P$-complete decision problem to verify if a given vertex $v \in V$ is a point of no return.
Encoding: Generating Program $\mathcal{P}_g$

\[0 \leq \#\{\text{pick}_g(X, Y)\} \leq 1 \leftarrow \text{arc}(X, Y, L).\]
\[\leftarrow \text{pick}_g(X, Y) \wedge \text{pick}_g(X', Y') \wedge \text{arc}(X, Y, \text{pos}(A)) \wedge \text{arc}(X', Y', \text{neg}(A)).\]
\[r_g(X) \leftarrow \text{init}(X).\]
\[r_g(Y) \leftarrow r_g(X) \wedge \text{pick}_g(X, Y).\]
\[\leftarrow \neg r_g(X) \wedge \text{pick}_g(X, Y).\]
\[\leftarrow \text{ponr}(X) \wedge \neg r_g(X).\]
\[\leftarrow \text{ponr}(X) \wedge \text{pick}_g(X, Y).\]
\[\leftarrow \text{pick}_g(X, Y) \wedge \text{pick}_g(X, Z) \wedge Y \neq Z.\]
\[\leftarrow \text{pick}_g(X, Y) \wedge \text{pick}_g(Z, Y) \wedge X \neq Z.\]
Encoding: Testing Program $\mathcal{P}_t$

$0 \leq \#\{\text{pick}_t(X, Y)\} \leq 1 \iff \text{arc}(X, Y, L)$.

$\text{pick}(X, Y) \iff \text{pick}_t(X, Y)$.

$\text{pick}(X, Y) \iff \text{pick}_g(X, Y)$.

$\iff \text{pick}(X, Y) \land \text{pick}(X', Y') \land$

$\text{arc}(X, Y, \text{pos}(A)) \land \text{arc}(X', Y', \text{neg}(A))$.

$r_t(X) \iff \text{ponr}(X)$.

$r_t(Y) \iff r_t(X) \land \text{pick}_t(X, Y)$.

$\iff \neg r_t(X) \land \text{pick}_t(X, Y)$.

$\iff \text{init}(X) \land \neg r_t(X)$.

$\iff \text{init}(X) \land \text{pick}_t(X, Y)$.

$\iff \text{pick}_t(X, Y) \land \text{pick}_t(X, Z) \land Y \neq Z$.

$\iff \text{pick}_t(X, Y) \land \text{pick}_t(Z, Y) \land X \neq Z$. 
The SAT-to-SAT Architecture

- The core SAT-to-SAT solver [J. et al., 2016] consists of two CDCL SAT solvers essentially solving a formula
  \[ \exists \vec{x}(\phi \land \neg \exists \vec{y}\psi). \]

- Using a recursive SAT-to-SAT architecture, quantified Boolean formulas (QBFs) can be solved [B. et al., 2016b].

- It is possible to translate second-order specifications into SAT-to-SAT instances [B. et al., 2016a].

\[
T_{SM} : \quad \forall A : i(A) \Rightarrow a(A).
\]
\[
\forall R : r(R) \Rightarrow ((\forall A : \text{pb}(R, A) \Rightarrow i(A)) \land (\forall B : \text{nb}(R, B) \Rightarrow \neg i(B)) \Rightarrow \exists H : h(R, H) \land i(H)).
\]
\[
\neg \exists \vec{i}' : \quad (\forall A : i'(A) \Rightarrow i(A)) \land (\exists A : i(A) \land \neg i'(A)) \land
\forall R : r(R) \Rightarrow ((\forall A : \text{pb}(R, A) \Rightarrow i'(A)) \land
(\forall B : \text{nb}(R, B) \Rightarrow \neg i(B)) \Rightarrow \exists H : h(R, H) \land i'(H)).
\]
Proof-of-Concept Implementation

- The stable-unstable semantics can specified using a second-order theory $T_{SU}$:

  $$T_{SM}[r/r_g, a/a_g, h/h_g, pb/pb_g, nb/nb_g].$$

  $$\neg \exists i_t : T_{SM}[r/r_t, a/a_t, h/h_t, pb/pb_t, nb/nb_t, i/i_t]$$

  $$\wedge (\forall A : a_g(A) \wedge a_t(A) \Rightarrow (i(A) \iff i_t(A))).$$

- For a second-order interpretation $I$ that captures the structure of a combined logic program $(\mathcal{P}_g, \mathcal{P}_t)$,

  $$I \models T_{SU} \iff i^I$$ is a stable-unstable model of $(\mathcal{P}_g, \mathcal{P}_t)$.

- The implementation is available under

  http://research.ics.aalto.fi/software/sat/sat-to-sat/
Combined programs can be generalized using a parameter $k$ that determines the depth of combination:

- any normal logic program $\mathcal{P}$ is 1-combined,
- any combined logic program $(\mathcal{P}_g, \mathcal{P}_t)$ is 2-combined, and
- for $k > 2$, a $k$-combined program is a pair $(\mathcal{P}, \mathcal{C})$ where $\mathcal{P}$ is a normal program and $\mathcal{C}$ is a $(k - 1)$-combined program.

The stable-unstable semantics is analogously defined for $k$-combined programs with the depth of combination $k > 2$.

In general, it is $\Sigma^P_k$-complete to decide if a finite $k$-combined program has a stable-unstable model.
Conclusion

- Combined logic programs under stable-unstable models enable programming on the second level of the PH.
- The new methodology surpasses the need for previous saturation and meta-interpretation techniques.
- A proof-of-concept implementation is obtained by combining CDCL SAT solvers in an appropriate way.
- By recursive application of the idea, we obtain a gateway to programming on any level \( k \) of the PH.
- There are interesting avenues for future work:
  - Building a native solver for combined programs
  - The theory of stable-unstable semantics as such
See You at LPNMR’17 in Finland

14th International Conference on Logic Programming and Nonmonotonic Reasoning, July 3–6, 2017

http://lpnmr2017.aalto.fi/