Aalto University
School of Science

Stable-Unstable Semantics: Beyond NP with Normal Logic Programs

Bart Bogaerts[1,2], Tomi Janhunen[1], Shahab Tasharrofi[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]:

$$h_1 \vee \cdots \vee h_l \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m.$$

- The main decision problems of DLPs are either Σ_2^P- or Π_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]
  - CLASPD [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 $u \vee h_1 \vee \cdots \vee h_l \leftarrow a_1 \wedge \cdots \wedge a_n$ 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]:

$$r: \; h_1 \vee \cdots \vee h_l \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m.$$

$$\longmapsto \quad \left\{ \begin{array}{lll} \mathrm{head}(r, h_1). & \ldots & \mathrm{head}(r, h_l). \\ \mathrm{pbody}(r, a_1). & \ldots & \mathrm{pbody}(r, a_n). \\ \mathrm{npody}(r, b_1). & \ldots & \mathrm{nbody}(r, b_m). \end{array} \right.$$

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

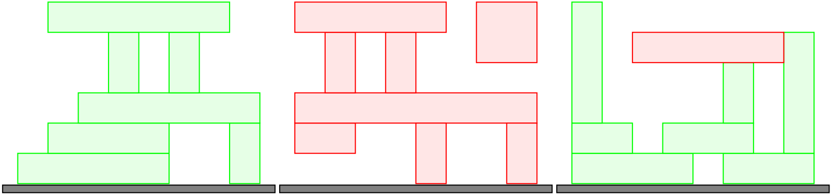
$$\begin{array}{rcl} \mathrm{in}(H) & \leftarrow & \mathrm{head}(R, H) \wedge \\ & & \mathrm{in}(P) : \mathrm{pbody}(R, P) \wedge \\ & & \neg\mathrm{in}(N) : \mathrm{nbody}(R, N) \wedge \\ & & \neg\mathrm{in}(OH) : \mathrm{head}(R, OH) : OH \neq H. \end{array}$$

- 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 $\mathcal{P}$ over a signature σ may have a set of parameters $\tau \subseteq \sigma$ not occurring in the heads of rules.
- An interpretation $M \subseteq \sigma$ of $\mathcal{P}$ is
  1. a stable model of $\mathcal{P}$, iff M is a $\subseteq$-minimal model of the Gelfond-Lifschitz reduct $\mathcal{P}^M$, and
  2. a parameterized stable model of $\mathcal{P}$, iff M is a stable model of the program $\mathcal{P} \cup \{a \leftarrow \mid a \in \tau \cap M\}$.

Example

Consider the following program $\mathcal{P}$ 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 τ.

Combination

- A combined logic program is pair $(\mathcal{P}_g, \mathcal{P}_t)$ of normal logic programs $\mathcal{P}_g$ and $\mathcal{P}_t$ with vocabularies σ_g and σ_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 $(\mathcal{P}_g, \mathcal{P}_t)$:

$$\begin{array}{|c|}\hline \{y_1, n_1, y_2, n_2\} \\ \hline y_1 \leftarrow \neg x_1. \\ n_1 \leftarrow \neg p_1. \\ y_2 \leftarrow \neg x_2. \\ n_2 \leftarrow \neg p_2. \\ \hline \{x_1, p_1, x_2, p_2\} \\ \hline \end{array}$$

$$\begin{array}{|lll|}\hline \multicolumn{3}{|c|}{\{t_x, f_x, t_y, f_y, f_1, f_2, f\}} \\ \hline f_1 \leftarrow \neg y_1 \wedge n_1 \wedge t_x. & \multicolumn{2}{l|}{f_2 \leftarrow \neg y_2 \wedge n_2 \wedge t_x.} \\ f_1 \leftarrow \neg y_1 \wedge \neg n_1 \wedge f_x. & \multicolumn{2}{l|}{f_2 \leftarrow \neg y_2 \wedge \neg n_2 \wedge f_x.} \\ f_1 \leftarrow y_1 \wedge n_1 \wedge t_y. & \multicolumn{2}{l|}{f_2 \leftarrow y_2 \wedge n_2 \wedge t_y.} \\ f_1 \leftarrow y_1 \wedge \neg n_1 \wedge f_y. & \multicolumn{2}{l|}{f_2 \leftarrow y_2 \wedge \neg n_2 \wedge f_y.} \\ f \leftarrow f_1 \wedge f_2. & t_x \leftarrow \neg f_x. & t_y \leftarrow \neg f_y. \\ f \leftarrow \neg f. & f_x \leftarrow \neg t_x. & f_y \leftarrow \neg t_y. \\ \hline \multicolumn{3}{|c|}{\{y_1, n_1, y_2, n_2\}} \\ \hline \end{array}$$

Stable-Unstable Semantics

- Let $(\mathcal{P}_g, \mathcal{P}_t)$ be a combined logic program with vocabularies σ_g and σ_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 τ_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

$$\mathcal{P}_g\colon \quad a \leftarrow \neg b.\ b \leftarrow \neg a. \qquad \mathcal{P}_t\colon \quad c \leftarrow a, \neg c.$$

the only stable-unstable model is $M = \{a\}$.

Example

$\{y_1, n_1, y_2, n_2\}$
$y_1 \leftarrow \neg x_1.$
$n_1 \leftarrow \neg p_1.$
$y_2 \leftarrow \neg x_2.$
$n_2 \leftarrow \neg p_2.$
$\{x_1, p_1, x_2, p_2\}$

$\{t_x, f_x, t_y, f_y, f_1, f_2, f\}$		
$f_1 \leftarrow \neg y_1 \wedge n_1 \wedge t_x.$	$f_2 \leftarrow \neg y_2 \wedge n_2 \wedge t_x.$	
$f_1 \leftarrow \neg y_1 \wedge \neg n_1 \wedge f_x.$	$f_2 \leftarrow \neg y_2 \wedge \neg n_2 \wedge f_x.$	
$f_1 \leftarrow y_1 \wedge n_1 \wedge t_y.$	$f_2 \leftarrow y_2 \wedge n_2 \wedge t_y.$	
$f_1 \leftarrow y_1 \wedge \neg n_1 \wedge f_y.$	$f_2 \leftarrow y_2 \wedge \neg n_2 \wedge f_y.$	
$f \leftarrow f_1 \wedge f_2.$	$t_x \leftarrow \neg f_x.$	$t_y \leftarrow \neg f_y.$
$f \leftarrow \neg f.$	$f_x \leftarrow \neg t_x.$	$f_y \leftarrow \neg t_y.$
$\{y_1, n_1, y_2, n_2\}$		

Clause	M_i	Stable models given M_i
$x \vee x$	$\{x_1, p_1, x_2, p_2\}$	$\{f_x, f_y, f_1, f_2, f\}$, $\{f_x, t_y, f_1, f_2, f\}$
$x \vee \overline{x}$	$\{x_1, p_1, x_2, n_2\}$	—
$x \vee y$	$\{x_1, p_1, y_2, p_2\}$	$\{f_x, f_y, f_1, f_2, f\}$
$x \vee \overline{y}$	$\{x_1, p_1, y_2, n_2\}$	$\{f_x, t_y, f_1, f_2, f\}$
...	...	...

☞ $\{x_1, p_1, x_2, n_2\}$, $\{x_1, n_1, x_2, p_2\}$, $\{y_1, p_1, y_2, n_2\}$, $\{y_1, n_1, y_2, p_2\}$.

Results

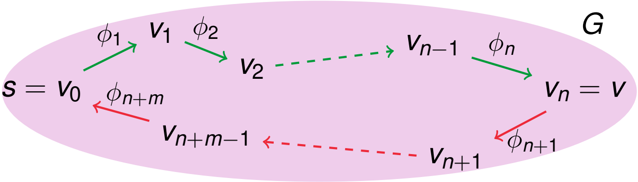
- Any disjunctive program $\mathcal{P}$ can be rewritten as a combined logic program $(\mathcal{P}_g, \mathcal{P}_t)$ as done by GNT [J. et al., 2006].
- We call a combined logic program $(\mathcal{P}_g, \mathcal{P}_t)$ independent, if $\sigma_g \cap \sigma_t = \emptyset$, i.e., $\mathcal{P}_g$ and $\mathcal{P}_t$ 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. Σ_2^P-complete in general, and
  2. D^P-complete for independent combined programs.

Encodings

- Winning strategies for parity games
  - Correspond to model checking problems in μ-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 ϕ.
- 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 Σ_2^P-complete decision problem to verify if a given vertex $v \in V$ is a point of no return.

Encoding: Generating Program $\mathcal{P}_g$

$$
\begin{array}{l}
0 \leq \#\{\mathrm{pick}_g(X, Y)\} \leq 1 \leftarrow \mathrm{arc}(X, Y, L). \\
\leftarrow \mathrm{pick}_g(X, Y) \wedge \mathrm{pick}_g(X', Y') \\
\quad \wedge\, \mathrm{arc}(X, Y, \mathrm{pos}(A)) \wedge \mathrm{arc}(X', Y', \mathrm{neg}(A)). \\
\mathrm{r}_g(X) \leftarrow \mathrm{init}(X). \\
\mathrm{r}_g(Y) \leftarrow \mathrm{r}_g(X) \wedge \mathrm{pick}_g(X, Y). \\
\leftarrow \neg \mathrm{r}_g(X) \wedge \mathrm{pick}_g(X, Y). \\
\leftarrow \mathrm{ponr}(X) \wedge \neg \mathrm{r}_g(X). \\
\leftarrow \mathrm{ponr}(X) \wedge \mathrm{pick}_g(X, Y). \\
\leftarrow \mathrm{pick}_g(X, Y) \wedge \mathrm{pick}_g(X, Z) \wedge Y \neq Z. \\
\leftarrow \mathrm{pick}_g(X, Y) \wedge \mathrm{pick}_g(Z, Y) \wedge X \neq Z.
\end{array}
$$

Encoding: Testing Program $\mathcal{P}_t$

$$
\begin{array}{l}
0 \leq \#\{\mathrm{pick}_t(X, Y)\} \leq 1 \leftarrow \mathrm{arc}(X, Y, L). \\
\mathrm{pick}(X, Y) \leftarrow \mathrm{pick}_t(X, Y). \\
\mathrm{pick}(X, Y) \leftarrow \mathrm{pick}_g(X, Y). \\
\leftarrow \mathrm{pick}(X, Y) \wedge \mathrm{pick}(X', Y') \wedge \\
\quad \mathrm{arc}(X, Y, \mathrm{pos}(A)) \wedge \mathrm{arc}(X', Y', \mathrm{neg}(A)). \\
\mathrm{r}_t(X) \leftarrow \mathrm{ponr}(X). \\
\mathrm{r}_t(Y) \leftarrow \mathrm{r}_t(X) \wedge \mathrm{pick}_t(X, Y). \\
\leftarrow \neg \mathrm{r}_t(X) \wedge \mathrm{pick}_t(X, Y). \\
\leftarrow \mathrm{init}(X) \wedge \neg \mathrm{r}_t(X). \\
\leftarrow \mathrm{init}(X) \wedge \mathrm{pick}_t(X, Y). \\
\leftarrow \mathrm{pick}_t(X, Y) \wedge \mathrm{pick}_t(X, Z) \wedge Y \neq Z. \\
\leftarrow \mathrm{pick}_t(X, Y) \wedge \mathrm{pick}_t(Z, Y) \wedge X \neq Z.
\end{array}
$$

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
$$\exists \vec{x}(\phi \wedge \neg \exists \vec{y} \psi).$$
- 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].

$$
\begin{aligned}
T_{SM}:\quad & \forall A : \mathrm{i}(A) \Rightarrow \mathrm{a}(A).\\
& \forall R : \mathrm{r}(R) \Rightarrow \big((\forall A : \mathrm{pb}(R,A) \Rightarrow \mathrm{i}(A)) \wedge (\forall B : \mathrm{nb}(R,B) \Rightarrow \neg \mathrm{i}(B)) \Rightarrow\\
& \qquad\qquad \exists H : \mathrm{h}(R,H) \wedge \mathrm{i}(H)\big).\\
& \neg \exists \mathrm{i}' :\\
& \quad (\forall A : \mathrm{i}'(A) \Rightarrow \mathrm{i}(A)) \wedge (\exists A : \mathrm{i}(A) \wedge \neg \mathrm{i}'(A)) \wedge\\
& \quad \forall R : \mathrm{r}(R) \Rightarrow \big((\forall A : \mathrm{pb}(R,A) \Rightarrow \mathrm{i}'(A)) \wedge\\
& \qquad\qquad (\forall B : \mathrm{nb}(R,B) \Rightarrow \neg \mathrm{i}(B)) \Rightarrow \exists H : \mathrm{h}(R,H) \wedge \mathrm{i}'(H)\big).
\end{aligned}
$$

Proof-of-Concept Implementation

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

$$T_{SM}[\mathrm{r}/\mathrm{r}_g, \mathrm{a}/\mathrm{a}_g, \mathrm{h}/\mathrm{h}_g, \mathrm{pb}/\mathrm{pb}_g, \mathrm{nb}/\mathrm{nb}_g].$$
$$\neg\exists \mathrm{i}_t : T_{SM}[\mathrm{r}/\mathrm{r}_t, \mathrm{a}/\mathrm{a}_t, \mathrm{h}/\mathrm{h}_t, \mathrm{pb}/\mathrm{pb}_t, \mathrm{nb}/\mathrm{nb}_t, \mathrm{i}/\mathrm{i}_t]$$
$$\wedge (\forall A : \mathrm{a}_g(A) \wedge \mathrm{a}_t(A) \Rightarrow (\mathrm{i}(A) \Leftrightarrow \mathrm{i}_t(A))).$$

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

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

- The implementation is available under `http://research.ics.aalto.fi/software/sat/sat-to-sat/`

Beyond Second Level with Normal Logic Programs

- 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 Σ_k^P-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/