Aalto University
School of Science

CS-E4540 Answer Set Programming

Lecture 2: Negation and Non-Monotonicity

Aalto University
School of Science
Department of Computer Science

Autumn 2016

Lecture 2: Negation and Non-Monotonicity

Foundational paper:
M. Gelfond and V. Lifschitz:
"The Stable Model Semantics for Logic Programming",
In Proc. of ICLP-88, 1070–1080.

1. NEGATIVE CONDITIONS

- The semantics based on least models provides a logical foundation for rule-based reasoning:

$$P \models a \text{ iff } a \in \mathrm{LM}(P)$$

  for any atom a appearing in P.
- In particular, atoms $a \in \mathrm{Hb}(P)$ that are not logical consequences of P, i.e., $P \not\models a$ holds, are false in $\mathrm{LM}(P)$ by default.
- In many applications, it is convenient/necessary to refer to complements of certain relations using negative conditions.
- The notion of answer sets based on stable models provides a declarative semantics for programs involving negative conditions.

Example

Consider the following definition of a conscript:

$$\text{Conscript}(x) \leftarrow \text{Person}(x), \sim\!\text{Female}(x).$$

Example

Consider the following set of rules involving negative conditions.

$$\text{Conscript}(x) \leftarrow \text{Person}(x), \sim\text{Female}(x).$$
$$\text{Female}(x) \leftarrow \text{Person}(x), \sim\text{Volunteer}(x), \sim\text{Conscript}(x).$$
$$\text{Person}(\text{joe}).$$

What would be the right answer for the query $\text{Conscript}(\text{joe})$?

- The meaning of the rules depends on the order of application:

$$\text{Person}(\text{joe}), \sim\text{Female}(\text{joe}) \implies \text{Conscript}(\text{joe})$$
$$\text{Person}(\text{joe}), \sim\text{Volunteer}(\text{joe}), \sim\text{Conscript}(\text{joe}) \implies \text{Female}(\text{joe})$$

- Thus it seems non-trivial to combine recursive definitions with negation and, in particular, to obtain a declarative semantics.

2. STABLE MODEL SEMANTICS

- In 1988, Gelfond and Lifschitz proposed stable models in order to provide a declarative semantics for negative conditions in rules.
- The rules of normal logic programs are of the form

$$a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m.$$

  where $\sim$ denotes negation by default.
- Stable models are based on the following two ideas:
  1. $M \models \sim c$ holds for a negative condition $\sim c \iff c \notin M$, and
  2. a model M is stable iff it is the least Herbrand model for the rules having their all negative conditions satisfied by M.

Example

Given the program $P = \{b \leftarrow \sim c.\ \ a \leftarrow b, \sim d.\ \ \}$, $M = \{a, b\}$ satisfies these requirements.

Example

Reconsider the program from the preceding example after grounding:

$$\mathrm{Conscript}(\mathrm{joe}) \leftarrow \mathrm{Person}(\mathrm{joe}),\ \sim\!\mathrm{Female}(\mathrm{joe}).$$
$$\mathrm{Female}(\mathrm{joe}) \leftarrow \mathrm{Person}(\mathrm{joe}),\ \sim\!\mathrm{Volunteer}(\mathrm{joe}),\ \sim\!\mathrm{Conscript}(\mathrm{joe}).$$
$$\mathrm{Person}(\mathrm{joe}).$$

- The model $M = \{\mathrm{Person}(\mathrm{joe}), \mathrm{Conscript}(\mathrm{joe})\}$ is stable.
- The negative conditions of the first and the last rule are true in M which is the least Herbrand model of the respective positive rules:

$$\mathrm{Conscript}(\mathrm{joe}) \leftarrow \mathrm{Person}(\mathrm{joe}). \quad \mathrm{Person}(\mathrm{joe}).$$

- But $N = \{\mathrm{Person}(\mathrm{joe}), \mathrm{Female}(\mathrm{joe})\}$ is also stable (which suggests us to specify Joe's gender; or to revise the given rules).

Definition of Stability

Definition

Let P be a normal logic program without variables and $M \subseteq \mathrm{Hb}(P)$ an interpretation. The Gelfond-Lifschitz reduct of P with respect to M, denoted by P^M, is:

$$\{a \leftarrow b_1, \ldots, b_n \mid a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m \in P \\ \text{and } M \models \sim c_1, \ldots, \sim c_m\}.$$

Remark

In the definition of P^M, $M \models \sim c_1, \ldots, \sim c_m$ iff $M \cap \{c_1, \ldots, c_m\} = \emptyset$.

Definition

Let P be a normal logic program without variables.
An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of P iff $M = \mathrm{LM}(P^M)$.

Example

Consider a normal logic program P having the rules listed below:

$$
\begin{aligned}
&a \leftarrow c, \sim b.\\
&b \leftarrow \sim a.\\
&c \leftarrow \sim d.\\
&d \leftarrow \sim a.
\end{aligned}
$$

1. The interpretation $M_1 = \{a, c\}$ is a stable model of P because $P^{M_1} = \{a \leftarrow c.\ \ c.\ \ \}$ and M_1 is the least model of P^{M_1}.
2. But $M_2 = \{a, d\}$ is not stable because $P^{M_2} = \{a \leftarrow c.\ \ \}$ for which the least model is $\emptyset$. Note that $M_2 \models P$ in the classical sense.
3. Finally, $M_3 = \{b, d\}$ is also a stable model of P.

The Γ_P Operator

Definition

Given a normal logic program P, define an operator $\Gamma_P : \mathbf{2}^{\mathrm{Hb}(P)} \to \mathbf{2}^{\mathrm{Hb}(P)}$ by setting

$$\Gamma_P(M) = \{a \mid a \in \mathrm{Hb}(P) \text{ and } P^M \models a\}.$$

Proposition

An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of a normal program P iff $M = \Gamma_P(M)$.

Proof

It is easy to see that for any $M \subseteq \mathrm{Hb}(P)$, $\Gamma_P(M) = \mathrm{LM}(P^M)$. $\square$

Properties of the Γ_P Operator

The operator Γ_P is not monotonic but antimonotonic:

Proposition

For any normal program P and interpretations $M \subseteq N \subseteq \mathrm{Hb}(P)$, $\Gamma_P(N) \subseteq \Gamma_P(M)$.

Proof

It is sufficient to note that $M \subseteq N$ implies $P^N \subseteq P^M$ and $\mathrm{LM}(P^N) \subseteq \mathrm{LM}(P^M)$ by the monotonicity of $\mathrm{LM}(\cdot)$. $\square$

Properties of Stable Models

- Unlike the least model of a positive program, stable models are not necessarily unique as demonstrated by programs below:

  1. $P_0 = \{a \leftarrow \sim a.\ \}$ has no stable models.
  2. $P_1 = \{a \leftarrow \sim b.\ \}$ has one stable model $\{a\}$.
  3. $P_2 = \{a \leftarrow \sim b.\ b \leftarrow \sim a.\ \}$ has two stable models $\{a\}$ and $\{b\}$.

☞ We write $\mathrm{SM}(P)$ for the set of stable models of P.

- Stable models are minimal in the sense that if $M \in \mathrm{SM}(P)$ then there is no other $N \in \mathrm{SM}(P)$ such that $N \subset M$.

☞ For normal programs, the set $\mathrm{SM}(P)$ forms an antichain.

- A stable model $M \in \mathrm{SM}(P)$ is strongly grounded in the rules of P:

$$a \in M \text{ iff } P^M \models a.$$

Answer Set Programming

- A traditional PROLOG system answers a query Q either "yes" (with an answer substitution θ for the variables of Q) or "no".
- Stable models, or answer sets, are based on a novel interpretation of logic programs as sets of constraints on models.
- Typically, answer sets—computed using a special search engine—capture solutions to the problem being solved.

$$
\begin{tikzcd}
\boxed{\text{Problem}} \arrow[d] & \boxed{\text{Solution}} \\
\boxed{\text{Set of rules}} \arrow[r] & \boxed{\text{Answer set}} \arrow[u]
\end{tikzcd}
$$

Remark

Rule-based languages are highly expressive:
Many problems involving constraints can be reformulated as problems of finding a stable model for the respective set of rules.

3. VARIABLES AND DOMAINS

- Let P be a normal logic program—potentially involving variables.
- The respective ground program $\mathrm{Gnd}(P)$ is defined in the same way as for positive programs.

Definition

A Herbrand interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of P iff $M = \Gamma_{\mathrm{Gnd}(P)}(M) = \mathrm{LM}(\mathrm{Gnd}(P)^M)$.

Example

Let us consider $P = \{A(c,d).\ \ B(x) \leftarrow A(x,y), \sim B(y).\ \ \}$. The ground program $\mathrm{Gnd}(P)$ contains the following rules:

$$
\begin{array}{lll}
A(c,d). & B(c) \leftarrow A(c,c), \sim B(c). & B(c) \leftarrow A(c,d), \sim B(d). \\
 & B(d) \leftarrow A(d,c), \sim B(c). & B(d) \leftarrow A(d,d), \sim B(d).
\end{array}
$$

The interpretation $M = \{A(c,d),\ B(c)\}$ is the only stable model of P.

Domain Predicates

- Ground programs $\mathrm{Gnd}(P)$ can become very large and they may contain many useless or redundant rules.
- A way to prune unnecessary rules is to introduce domain predicates, which are relation symbols with a fixed interpretation.
- Even recursive definitions for domain predicates (see $G(\cdot,\cdot)$ below) can be tolerated if recursion does not involve negation.

Example

Consider the following example:

$$
\begin{array}{ll}
D(a). \quad E(b). & F(x) \leftarrow D(x). \quad F(x) \leftarrow E(x). \\
G(x,y) \leftarrow D(x), E(y). & G(y,x) \leftarrow G(x,y), F(x), F(y). \\
R(y,x) \leftarrow G(x,y), \sim S(y,x). & S(y,x) \leftarrow G(x,y), \sim R(y,x).
\end{array}
$$

Here D, E, F, and G are domain predicates but R and S are not.

Example—Cont'd

Some observations about the preceding program, say P, follow:

1. The Herbrand universe $\mathrm{Hu}(P) = \{a, b\}$ is finite and thus the ground program $\mathrm{Gnd}(P)$ remains also finite.
   - The least Herbrand model for the uppermost six rules (say P') is $\mathrm{LM}(\mathrm{Gnd}(P')) = \{D(a), E(b), F(a), F(b), G(a,b), G(b,a)\}$.
   - The model $\mathrm{LM}(\mathrm{Gnd}(P'))$ can be represented as a set of facts.
2. Only two ground instances of the last two rules each are needed:

$$
\begin{array}{ll}
R(b,a) \leftarrow G(a,b), \sim S(b,a). & R(a,b) \leftarrow G(b,a), \sim S(a,b). \\
S(b,a) \leftarrow G(a,b), \sim R(b,a). & S(a,b) \leftarrow G(b,a), \sim R(a,b).
\end{array}
$$

3. An intelligent grounder can simplify these further by dropping conditions $G(a,b)$ and $G(b,a)$ as they are satisfied for sure.

Restricting Domains of Variables

- The idea is to control the size of the resulting ground program by using positive body literals to restrict the domains of variables.

Definition

A normal program P is safe iff for each rule

$$R(\vec{t}) \leftarrow R_1(\vec{t_1}), \ldots, R_n(\vec{t_n}), \sim S_1(\vec{u_1}), \ldots, \sim S_m(\vec{u_m})$$

of P and for each variable x appearing in the rule, x appears in some of the positive conditions $R_i(\vec{t_i})$.

Example

The rule $R(x,y) \leftarrow D(x), D(y), \sim S(y,x)$ is safe,
but the rules $F(x,y) \leftarrow D(x), E(x)$ and $E(x) \leftarrow \sim D(x)$ are not.

4. PROGRAMMING TIPS

The logical connectives of propositional logic are available.

1. The conjunction of conditions $c_1,\ldots,c_n$ is captured by a single (positive) rule $c \leftarrow c_1,\ldots,c_n.$
2. Expressing the disjunction of conditions $d_1,\ldots,d_n$ requires the introduction of n rules $d \leftarrow d_1. \quad \ldots \quad d \leftarrow d_n.$
3. A rule $f \leftarrow b_1,\ldots,b_n,{\sim}f$ expresses a constraint $\leftarrow b_1,\ldots,b_n,{\sim}f$ that formalizes the negation $\neg(b_1 \wedge \ldots \wedge b_n \wedge \neg f)$.
   If all program rules with head f are “self-defeating”, the negation of the body simplifies to $\neg(b_1 \wedge \ldots \wedge b_n)$.

Example

One is supposed to have one or two delicacies out of three:

Some $\leftarrow$ Cake. Some $\leftarrow$ Bun. Some $\leftarrow$ Cookie.

All $\leftarrow$ Cake, Bun, Cookie. F $\leftarrow$ All, $\sim$F. F $\leftarrow$ $\sim$Some, $\sim$F.

Making Choices

- A choice between two atoms a and b can be expressed in terms of two normal rules $a \leftarrow \sim b$ and $b \leftarrow \sim a$.
- Such a choice can be generalized for any number of atoms and conditionalized by adding literals in rule bodies.
- A typical approach in ASP is to express a number of choices and then exclude certain combinations using other rules/constraints.

Example

One is supposed to have coffee or tea—but not both—and also one of three delicacies in case tea is selected:

$$
\begin{array}{ll}
\text{Coffee} \leftarrow \sim\text{Tea}. & \text{Cake} \leftarrow \text{Tea}, \sim\text{Cookie}, \sim\text{Bun}. \\
\text{Tea} \leftarrow \sim\text{Coffee}. & \text{Bun} \leftarrow \text{Tea}, \sim\text{Cookie}, \sim\text{Cake}. \\
 & \text{Cookie} \leftarrow \text{Tea}, \sim\text{Bun}, \sim\text{Cake}.
\end{array}
$$

Rules with Exceptions

- Normal programs enable context-dependent reasoning in which the applicability of rules depends dynamically on the context.
- In common-sense reasoning, it is typical to formalize the normal state of affairs including any exceptions to that.

Example

Birds do normally fly—unless we have an exceptional bird.

$$\text{Flies}(x) \leftarrow \text{Bird}(x), \sim\text{Abnormal}(x).$$
$$\text{Abnormal}(x) \leftarrow \text{Penguin}(x). \quad \text{Abnormal}(x) \leftarrow \text{Oily}(x). \quad \ldots$$

The stable models of this program, say P, behave as follows:

1. $\mathrm{SM}(P \cup \{\text{Bird}(\text{tw}).\ \}) = \{\{\text{Bird}(\text{tw}), \text{Flies}(\text{tw})\}\}$.
2. $\mathrm{SM}(P \cup \{\text{Bird}(\text{tw}).\ \text{Oily}(\text{tw}).\ \}) = \{\{\text{Bird}(\text{tw}), \text{Oily}(\text{tw}), \text{Abnormal}(\text{tw})\}\}$.

5. PROBLEM SOLVING

To illustrate the use of normal rules in a practical setting, we will formalize the following problems:

1. Satisfiability checking
   - The canonical NP-complete problem established by Cook [1971].
2. Graph 3-coloring
3. Hamiltonian cycles in graphs

Satisfiability Checking

A set of clauses S is translated into a normal program P_S as follows:

1. For each atom $a \in \mathrm{Hb}(S)$, we introduce a new atom $\overline{a}$ and two rules $\overline{a} \leftarrow \sim a$ and $a \leftarrow \sim \overline{a}$.
2. Each clause $a_1 \vee \ldots \vee a_n \vee \neg b_1 \vee \ldots \vee \neg b_m$ from S is translated into
$$f \leftarrow \overline{a}_1, \ldots, \overline{a}_n, b_1, \ldots, b_m, \sim f$$
where $f \notin \mathrm{Hb}(S)$ is a new atom.

$\Longrightarrow \mathrm{Hb}(P_S) = \mathrm{Hb}(S) \cup \{\overline{a} \mid a \in \mathrm{Hb}(S)\} \cup \{f\}$.

Proposition

A set of clauses S has a model M, i.e., S is satisfiable, iff
the program P_S has a stable model N such that $M = N \cap \mathrm{Hb}(S)$.

Example

Consider the translation of $S = \{a \vee b,\ a \vee \neg b,\ \neg a \vee \neg b\}$ into a normal program. The translation P_S consists of the following rules:

$$
\begin{array}{llll}
a \leftarrow \sim\overline{a}. & \overline{a} \leftarrow \sim a. & b \leftarrow \sim\overline{b}. & \overline{b} \leftarrow \sim b. \\
f \leftarrow \overline{a}, \overline{b}, \sim f. & f \leftarrow \overline{a}, b, \sim f. & f \leftarrow a, b, \sim f. &
\end{array}
$$

A number of observations can be made:

1. Now, the set of clauses S has a model M iff the program P_S has a stable model N such that $M = N \cap \{a, b\}$.
2. Because $N_1 = \{a, \overline{b}\}$ is a stable model of P_S, we know that $M_1 = \{a\}$ is a model of S.
3. On the other hand, $N_2 = \{\overline{a}, \overline{b}\}$ is not a stable model of P_S.

Graph 3-Coloring

A graph G can be represented by facts of the form "Edge(x, y)."
where x and y stand for nodes. The following normal program P_G^{3c} is a uniform encoding for the problem of coloring the nodes of G with three colors so that the endpoints of each edge have different colors.

$$
\begin{array}{lll}
\text{Node}(x) \leftarrow \text{Edge}(x,y). & \text{Node}(y) \leftarrow \text{Edge}(x,y). & \text{(projection)} \\
\text{Black}(x) \leftarrow \text{Node}(x), \sim\text{White}(x), \sim\text{Grey}(x). & & \text{(choices)} \\
\text{White}(x) \leftarrow \text{Node}(x), \sim\text{Black}(x), \sim\text{Grey}(x). & & \\
\text{Grey}(x) \leftarrow \text{Node}(x), \sim\text{White}(x), \sim\text{Black}(x). & & \\
\text{F} \leftarrow \text{Edge}(x,y), \text{Black}(x), \text{Black}(y), \sim\text{F}. & & \text{(constraints)} \\
\text{F} \leftarrow \text{Edge}(x,y), \text{White}(x), \text{White}(y), \sim\text{F}. & & \\
\text{F} \leftarrow \text{Edge}(x,y), \text{Grey}(x), \text{Grey}(y), \sim\text{F}. & &
\end{array}
$$

Proposition

The graph G has a 3-coloring iff P_G^{3c} has a stable model.

Hamiltonian Cycles in Graphs

The problem is to check whether a given graph has a Hamiltonian cycle that visits all nodes of the graph exactly once. In addition to the edge relation, the following rules are introduced in program P_G^{H}.

1. The nodes of the graph are extracted from the edge relation:

$$\mathsf{Node}(x) \leftarrow \mathsf{Edge}(x,y). \quad \mathsf{Node}(y) \leftarrow \mathsf{Edge}(x,y).$$
$$\mathsf{Same}(x,x) \leftarrow \mathsf{Node}(x).$$

2. Any cycle starts from a particular node chosen here.

$\mathsf{Start}(x) \leftarrow \mathsf{Node}(x), \sim\mathsf{Other}(x).$
$\mathsf{Other}(x) \leftarrow \mathsf{Node}(x), \sim\mathsf{Start}(x).$
$\mathsf{F} \leftarrow \mathsf{Start}(x), \mathsf{Start}(y), \sim\mathsf{Same}(x,y), \mathsf{Node}(x), \mathsf{Node}(y), \sim\mathsf{F}.$
$\mathsf{HasStart} \leftarrow \mathsf{Start}(x), \mathsf{Node}(x).$
$\mathsf{F} \leftarrow \sim\mathsf{HasStart}, \sim\mathsf{F}.$

Hamiltonian Cycles—Cont'd

3. Next the edges which are on the cycle are chosen.

$$\begin{aligned}
&\mathsf{In}(x1,x2) \leftarrow \mathsf{Edge}(x1,x2),\ \sim\mathsf{Out}(x1,x2).\\
&\mathsf{Out}(x1,x3) \leftarrow \mathsf{In}(x1,x2),\ \sim\mathsf{Same}(x2,x3),\ \mathsf{Edge}(x1,x2),\ \mathsf{Edge}(x1,x3).\\
&\mathsf{Out}(x3,x2) \leftarrow \mathsf{In}(x1,x2),\ \sim\mathsf{Same}(x1,x3),\ \mathsf{Edge}(x1,x2),\ \mathsf{Edge}(x3,x2).
\end{aligned}$$

4. All nodes of the graph must be reachable via the cycle.

$$\begin{aligned}
&\mathsf{Reached}(x) \leftarrow \mathsf{In}(y,x),\ \mathsf{Start}(y),\ \mathsf{Edge}(y,x).\\
&\mathsf{Reached}(x) \leftarrow \mathsf{In}(y,x),\ \mathsf{Reached}(y),\ \mathsf{Edge}(y,x).\\
&\mathsf{F} \leftarrow \mathsf{Node}(x),\ \sim\mathsf{Reached}(x),\ \sim\mathsf{F}.
\end{aligned}$$

Proposition

The program P_G^{H}—together with facts that describe the edge relation—has a stable model $\iff$ G has a Hamiltonian cycle.

OBJECTIVES

- You know what kind of problems arise when negative conditions are incorporated into recursive definitions.
- You are able to reproduce the definition of stable models and to prove simple properties about them.
- You can calculate stable models for simple normal logic programs (at least by exhaustive generation of model candidates).
- You are able to formalize simple constraint programming problems by describing their solutions in terms of rules.

TIME TO PONDER

As demonstrated above, a normal logic program can have several stable models, a unique stable model, or no stable models at all.

Problem

Design a propositional normal program P_n that has exactly $n \geq 0$ stable models.

How does the length of P_n, measured in the number of atoms and connectives, change as the function of n?