Aalto University
School of Science

CS-E4540 Answer Set Programming

Lecture 1: Basic Concepts

Aalto University
School of Science
Department of Computer Science

Autumn 2016

Lecture 1: Basic Concepts

Additional (historical) reference:
J. Lloyd: “Foundations of Logic Programming”, 1987.

1. BACKGROUND FOR RULES

- Answer set programming adopts a rule-based syntax previously used in PROLOG, deductive databases, and expert systems.
- Horn clauses provide rule-based reasoning with a solid theoretical foundation:
  1. Rules can be interpreted as Horn clauses.
  2. Classical models determine the set of logical consequences $\mathrm{Cn}(R)$ associated with each set of rules R.
- Horn clauses lend themselves for efficient implementation which makes them important from the computational point of view.
- Typically, applications require a more expressive language but for now we concentrate on rules corresponding to Horn clauses.
- Rule-based reasoning is still an active research area (see, e.g., `http://www.dbai.tuwien.ac.at/datalog2.0`).

Literals and Clauses

Definitions

1. A literal is either an atom a (a positive literal) or the negation of an atom $\neg a$ (a negative literal).
2. A clause is a disjunction $l_1 \vee \ldots \vee l_n$ of literals $l_1, \ldots, l_n$.
3. A Horn clause is a clause with at most one positive literal.
4. A program clause, or a rule for short, is a disjunction of literals $a \vee \neg b_1 \vee \ldots \vee \neg b_n$ with exactly one positive literal.

Example

The clauses $\neg p \vee \neg q \vee \neg r$ and $\neg p \vee q \vee \neg r$ are Horn clauses but $p \vee \neg q \vee r$ is not. Only the second is a program clause.

2. RULES AND PROGRAMS

Certain notational conventions are adopted for Horn clauses:

- A rule $a \vee \neg b_1 \vee \ldots \vee \neg b_n$ is written $a \leftarrow b_1, \ldots, b_n$ where a and $b_1, \ldots, b_n$ form the head and the body of the rule, respectively.
- A constraint $\neg b_1 \vee \ldots \vee \neg b_n$ is written $\leftarrow b_1, \ldots, b_n$ and it can be viewed as a rule with an empty head (i.e., empty disjunction).
- A fact a ($n = 0$) is a rule written without "$\leftarrow$".
- Full stops "." are also used to separate rules in sets of rules.

Definition

A positive program P is a set of rules as defined above.

Remark

Here the word "positive" refers to the fact that rule bodies are negation-free. Forms of negation will be introduced later.

Satisfaction and Entailment

Definitions

Assume rules and constraints based on a set of atoms $\mathcal{P}$.

1. A rule $a \leftarrow b_1, \ldots, b_n$ is satisfied in an interpretation $I \subseteq \mathcal{P}$, denoted $I \models a \leftarrow b_1, \ldots, b_n$, iff $\{b_1, \ldots, b_n\} \subseteq I$ implies $a \in I$.
2. A constraint $\leftarrow b_1, \ldots, b_n$ is satisfied in an interpretation $I \subseteq \mathcal{P}$, denoted $I \models \leftarrow b_1, \ldots, b_n$, iff $\{b_1, \ldots, b_n\} \not\subseteq I$.
3. An interpretation $M \subseteq \mathcal{P}$ is a model of a set of rules and constraints $P \cup C$, denoted $M \models P \cup C$, iff $M \models r$ for each $r \in P \cup C$.
4. An atom a is a logical consequence of $P \cup C$ iff $a \in M$ for every interpretation $M \subseteq \mathcal{P}$ such that $M \models P \cup C$.

Proposition

Each positive program P is satisfiable ($M = \mathrm{Hb}(P)$ is a trivial model).

Translating Constraints into Rules

- A set of Horn clauses S, effectively a union $P \cup C$ of a positive program P and a set of constraints C, is not satisfiable in general.
- E.g., $\{p,\ \neg p\}$ corresponding to $\{p\} \cup \{\leftarrow p\}$ has no models.
- Any set of constraints C can be translated into a positive program
$$\mathrm{Tr}_{\mathrm{RULE}}(C) = \{\bot \leftarrow b_1, \ldots, b_n \mid \leftarrow b_1, \ldots, b_n \in C\}$$
where $\bot$ is a new atom not appearing in $\mathrm{Hb}(P)$ nor $\mathrm{Hb}(C)$.

Proposition

A set of Horn clauses S, viewed as a union $P \cup C$ in the way explained above, is satisfiable $\iff P \cup \mathrm{Tr}_{\mathrm{RULE}}(C) \not\models \bot$.

Example

The unsatisfiability of $\{p,\ \neg p\}$ can be determined using the translation given above: $\{p\} \cup \mathrm{Tr}_{\mathrm{RULE}}(\{\leftarrow p\}) = \{p.\ \ \bot \leftarrow p.\ \ \} \models \bot$.

3. SUBSET MINIMAL MODELS

- Answer set programming is based on the closed world assumption (CWA) which means that atoms are false by default.
- When models are represented as subsets of $\mathcal{P}$, this idea corresponds to having subset minimal models:
  - E.g., the rule $a \leftarrow b$ has tree models $\emptyset$, $\{a\}$, and $\{a,b\}$.
  - Only the first one is minimal.
- From the knowledge representation perspective, the roles of the two truth values (true and false) become asymmetric:
  - every atom is false by default and
  - we can concentrate on specifying when atoms should be true.
- This contrasts with classical logic where the two truth values have completely symmetric roles.
- Positive rules of the form $a \leftarrow b_1,\ldots,b_n$ offer a basic primitive for specifying the condition(s) on which an atom a should be true.

Formal Definition of Minimal Models

Definition

1. An interpretation $M \subseteq \mathcal{P}$, represented as the set of atoms true in M, is smaller than another interpretation $N \subseteq \mathcal{P}$ iff $M \subset N$.
2. An interpretation $M \subseteq \mathrm{Hb}(P)$ is a minimal model of a (positive) program P iff $M \models P$ and there is no smaller model $N \models P$.

Example

Consider the following positive program:

$$P = \{q \leftarrow r.\ \ r \leftarrow p, q.\ \ \}.$$

- The interpretation $M = \{q, r\}$ is a model of P.
- However, M is not minimal because $N = \emptyset$ is also a model of P.
- But, in contrast, N is a minimal model of P.

Properties of Minimal Models (I)

Theorem

If $M_i \subseteq \mathrm{Hb}(P)$ (where $i \in I$) is a collection of models for a positive program P, then $M = \bigcap\{M_i \mid i \in I\}$ is also a model of P.

Proof

Suppose that $M \not\models P$.

$$
\begin{aligned}
\Longrightarrow \quad & \exists\, a \leftarrow b_1,\ldots,b_n \in P \text{ such that } \{b_1,\ldots,b_n\} \subseteq M \text{ but } a \notin M \\
\Longrightarrow \quad & \{b_1,\ldots,b_n\} \subseteq M_i \text{ for all } i \in I \\
\Longrightarrow \quad & a \in M_i \text{ for all } i \in I \text{ because } M_i \models P, \\
& a \leftarrow b_1,\ldots,b_n \in P \text{ and } M_i \models a \leftarrow b_1,\ldots,b_n \\
\Longrightarrow \quad & a \in M = \bigcap\{M_i \mid i \in I\}, \text{ a contradiction.}
\end{aligned}
$$

Thus $M \models P$ is necessarily the case. □

Properties of Minimal Models (II)

Theorem

A positive program P has at least one minimal model.

Proof

We will cover the case when $|\mathrm{Hb}(P)| = n$ is finite
(a generalization for the infinite case requires transfinite induction).

Since P is a positive program, we know that $M_0 = \mathrm{Hb}(P) \models P$. Then define a decreasing sequence $M_0 \supseteq \ldots \supseteq M_i \supseteq \ldots$ of models for P:

- If M_i is a minimal model of P, let $M_{i+1} = M_i$.
- If M_i is not a minimal model of P, it has a model $N \subset M_i$.
  Let $M_{i+1} = N$.

Assuming that $M_i \models P$ is never minimal implies that the sequence is properly decreasing for all $i \geq 0$. A contradiction when $i > n$. □

Example

The idea of the preceding proof can be demonstrated using

$$P_n = \{p_0 \leftarrow p_0.\ \ p_1 \leftarrow p_1.\ \ p_2 \leftarrow p_2.\ \ \ \ldots\ p_n \leftarrow p_n.\ \ \}$$

which is a positive program with a finite number, i.e., $n+1$, of rules.

- The interpretation $M_0 = \mathrm{Hb}(P_n) = \{p_0, \ldots, p_n\}$ is a model of P but not minimal because $M_1 = \{p_1, \ldots, p_n\} \models P$.
- A generalization for $i > 0$: the interpretation $M_i = \{p_i, \ldots, p_n\}$ is a model of P_n but not minimal because $M_{i+1} = \{p_{i+1}, \ldots, p_n\} \models P_n$.
- When i equals to $\left|\mathrm{Hb}(P_n)\right| = n+1$, we have a minimal model

$$M = \bigcap_{i=0}^{n+1} M_i = \emptyset.$$

Properties of Minimal Models (III)

Theorem

Every positive program P has a unique minimal model, the least model $\mathrm{LM}(P)$ of P, which is the intersection of its all models.

Proof

Since P has at least one minimal model (shown above), let us assume that P had two minimal models, say M_1 and M_2.

$$
\begin{aligned}
\Longrightarrow \quad & M_1 \cap M_2 \models P,\ M_1 \cap M_2 \subseteq M_1, \text{ and } M_1 \cap M_2 \subseteq M_2 \\
\Longrightarrow \quad & M_1 \cap M_2 = M_1 \text{ and } M_1 \cap M_2 = M_2 \qquad (M_1 \text{ and } M_2 \text{ are minimal}) \\
\Longrightarrow \quad & M_1 = M_2.
\end{aligned}
$$

Thus $\mathrm{LM}(P) \subseteq M$ holds for every $M \models P$ because $\mathrm{LM}(P)$ is unique. Since $\mathrm{LM}(P) \models P$, we obtain $\bigcap\{M \subseteq \mathrm{Hb}(P) \mid M \models P\} = \mathrm{LM}(P)$. □

Answer Sets

Corollary

For any positive program P, $\mathrm{LM}(P) = \{a \in \mathrm{Hb}(P) \mid P \models a\}$.

This result has interesting consequence:

- The least model of a positive program P provides means to answer queries about atoms in $\mathrm{Hb}(P)$.
- The set $\mathrm{LM}(P)$ forms the unique answer set of P.

Example

For $P = \{a \leftarrow b, c.\ \ b \leftarrow a, c.\ \ c \leftarrow a, b.\ \}$,

1. $\mathrm{LM}(P \cup \{a.\ \}) = \{a\}$ and
2. $\mathrm{LM}(P \cup \{a.\ \ b.\ \}) = \{a, b, c\}$.

Thus $P \cup \{a.\ \} \not\models c$ but $P \cup \{a.\ \ b.\ \} \models c$.

4. CONSTRUCTING THE LEAST MODEL

- Positive programs lend themselves to specifying when atoms are necessarily/immediately true:
  - Suppose there is a rule $a \leftarrow b$ in the program.
  - If b is necessarily true (by some other rules), so is a.
- The recursive application of this idea leads to a polytime (quadratic) algorithm for computing the consequences of a positive program.
- The algorithm can be formalized using an operator T_P that infers which atoms of a positive program P are necessarily true.
- A linear-time algorithm is feasible using data structures.

W. Dowling and J. Gallier: "Linear time algorithms for testing the satisfiability of propositional Horn formulae", JLP 1(3), 267–284, 1984.

The T_P Operator

Definition

Let P be a positive logic program. Then define an operator $\mathrm{T}_P : \mathbf{2}^{\mathrm{Hb}(P)} \to \mathbf{2}^{\mathrm{Hb}(P)}$ on interpretations $I \subseteq \mathrm{Hb}(P)$ as follows:

$$\mathrm{T}_P(I) = \{a \in \mathrm{Hb}(P) \mid a \leftarrow b_1, \ldots, b_n \in P \text{ and } \{b_1, \ldots, b_n\} \subseteq I\}.$$

An interpretation I is a fixpoint of the operator T_P iff $\mathrm{T}_P(I) = I$.
A fixpoint I is the least fixpoint of T_P iff $I \subseteq I'$ for every $I' = \mathrm{T}_P(I')$.

Example

Let us analyze $P = \{a \leftarrow a.\ \ b.\ \ c \leftarrow b.\ \ d \leftarrow a, b.\ \ \}$.

1. Now $\mathrm{T}_P(\{a\}) = \{a, b\}$ and $\mathrm{T}_P(\{a, b\}) = \{a, b, c, d\}$,
2. the interpretation $M_1 = \{a, b, c, d\}$ is a fixpoint of T_P since $\mathrm{T}_P(M_1) = \{a, b, c, d\} = M_1$, and
3. the interpretation $M_2 = \{b, c\}$ is the least fixpoint of T_P.

Properties of the T_P Operator (I)

Proposition

An interpretation $M \subseteq \mathrm{Hb}(P)$ is a model of a positive program P iff $\mathrm{T}_P(M) \subseteq M$.

Proof

For any interpretation $M \subseteq \mathrm{Hb}(P)$, $M \not\models P$

$\iff$ $\exists\, a \leftarrow b_1, \ldots, b_n \in P$ such that $\{b_1, \ldots, b_n\} \subseteq M$ but $a \notin M$

$\iff$ $\exists a \in \mathrm{T}_P(M)$ such that $a \notin M$

$\iff$ $\mathrm{T}_P(M) \not\subseteq M$. □

Properties of the T_P Operator (II)

Proposition

(Monotonicity) For a positive program P,

$$M \subseteq N \subseteq \mathrm{Hb}(P) \text{ implies } \mathrm{T}_P(M) \subseteq \mathrm{T}_P(N).$$

Proof

For any atom $a \in \mathrm{Hb}(P)$, we have that $a \in \mathrm{T}_P(M)$
$\Longrightarrow$ $\exists\, a \leftarrow b_1, \ldots, b_n \in P$ such that $\{b_1, \ldots, b_n\} \subseteq M$
$\Longrightarrow$ $\exists\, a \leftarrow b_1, \ldots, b_n \in P$ such that $\{b_1, \ldots, b_n\} \subseteq N$ $(M \subseteq N)$
$\Longrightarrow$ $a \in \mathrm{T}_P(N)$. $\square$

Properties of the Least Fixpoint (I)

Proposition

For a positive program P, the operator T_P has the least fixpoint $\mathrm{lfp}(\mathrm{T}_P) = \bigcap\{M \subseteq \mathrm{Hb}(P) \mid M = \mathrm{T}_P(M)\}$.

Proof

1. Every monotonic operator has a least fixpoint (Knaster-Tarski) which is unique. For T_P, we denote this fixpoint by $\mathrm{lfp}(\mathrm{T}_P)$.
2. For the intersection property, it is sufficient to note that by definition $\mathrm{lfp}(\mathrm{T}_P) \subseteq M$ for any fixpoint $M = \mathrm{T}_P(M)$, and for the least fixpoint $M = \mathrm{lfp}(\mathrm{T}_P)$ in particular. □

Iterative Construction

The unique fixpoint $\mathrm{lfp}(\mathrm{T}_P)$ has an inductive definition:

Definition

For a positive program P, define an increasing sequence

$$\mathrm{T}_P \uparrow 0, \mathrm{T}_P \uparrow 1, \mathrm{T}_P \uparrow 2, \ldots$$

of interpretations by setting

1. $\mathrm{T}_P \uparrow 0 = \emptyset$,
2. $\mathrm{T}_P \uparrow i+1 = \mathrm{T}_P(\mathrm{T}_P \uparrow i)$ for $i > 0$, and
3. $\mathrm{T}_P \uparrow \infty = \bigcup_{i=0}^{\infty} \mathrm{T}_P \uparrow i$ for the limit of the sequence.

Properties of the Least Fixpoint (II)

Theorem

For a positive program P, $\mathrm{lfp}(\mathrm{T}_P) = \mathrm{T}_P \uparrow \infty = \mathrm{LM}(P)$.

Proof

We will prove the claim $\mathrm{lfp}(\mathrm{T}_P) = \mathrm{T}_P \uparrow \infty$ when P is finite; the infinite case uses transfinite induction on the iteration sequence of T_P.

- ($\subseteq$) The monotonicity of T_P guarantees that the sequence of interpretations $\mathrm{T}_P \uparrow i$ is increasing. Hence $\mathrm{T}_P(\mathrm{T}_P \uparrow i) = \mathrm{T}_P \uparrow i$ for some $i \geq 0$. Thus $\mathrm{lfp}(\mathrm{T}_P) \subseteq \mathrm{T}_P \uparrow i \subseteq \mathrm{T}_P \uparrow \infty$.
- ($\supseteq$) It follows by induction on i that $\mathrm{T}_P \uparrow i \subseteq \mathrm{lfp}(\mathrm{T}_P)$ for every $i \geq 0$.

For $\mathrm{lfp}(\mathrm{T}_P) = \mathrm{LM}(P)$, we note the following:

- ($\subseteq$) It follows by induction that $\mathrm{T}_P \uparrow i \subseteq \mathrm{LM}(P)$ for every $i \geq 0$.
- ($\supseteq$) Any fixpoint $M = \mathrm{T}_P(M)$ is also a model of P. Thus the intersection of models, i.e., $\mathrm{LM}(P)$, is contained in M. □

Example

Reconsider the program $P = \{a \leftarrow a.\ \ b.\ \ c \leftarrow b.\ \ d \leftarrow a, b.\ \ \}$:

$$
\begin{aligned}
&\mathrm{T}_P \uparrow 0 = \emptyset, \\
&\mathrm{T}_P \uparrow 1 = \mathrm{T}_P(\mathrm{T}_P \uparrow 0) = \mathrm{T}_P(\emptyset) = \{b\}, \\
&\mathrm{T}_P \uparrow 2 = \mathrm{T}_P(\mathrm{T}_P \uparrow 1) = \mathrm{T}_P(\{b\}) = \{b, c\}, \\
&\mathrm{T}_P \uparrow 3 = \mathrm{T}_P(\mathrm{T}_P \uparrow 2) = \mathrm{T}_P(\{b, c\}) = \{b, c\}, \ldots \\
&\mathrm{T}_P \uparrow i+1 = \mathrm{T}_P(\mathrm{T}_P \uparrow i) = \mathrm{T}_P(\{b, c\}) = \{b, c\}, \ldots
\end{aligned}
$$

$$
\Longrightarrow \mathrm{T}_P \uparrow \infty = \bigcup_{i=0}^{\infty} \mathrm{T}_P \uparrow i = \{b, c\} = \mathrm{lfp}(\mathrm{T}_P) = \mathrm{LM}(P).
$$

Remarks

1. If P is a finite positive program (as above), then $\mathrm{lfp}(\mathrm{T}_P)$ is always reached with a finite number of steps.
2. For each $a \in \mathrm{lfp}(\mathrm{T}_P)$, there is a finite $i \geq 0$ such that $a \in \mathrm{T}_P \uparrow i$, even if P is infinite!

5. PROGRAMS WITH VARIABLES

- Disregarding any non-logical features, logic programs and deductive databases can be viewed as sets of rules of the form

$$P(\vec{t}) \leftarrow P_1(\vec{t_1}), \ldots, P_n(\vec{t_n})$$

  where $P(\vec{t})$ and $P_i(\vec{t_i})$'s are atomic formulas involving lists of terms $\vec{t}$, $\vec{t_1}$, ..., $\vec{t_n}$ as their arguments.
- Variables appearing in rules are universally quantified.
- Each set of rules P, also called a positive program in the sequel, has a Herbrand base $\mathrm{Hb}(P)$ associated with it.
- A rule $P(\vec{t}) \leftarrow P_1(\vec{t_1}), \ldots, P_n(\vec{t_n})$ with variables $x_1, \ldots, x_m$ stands for its all ground instances, each of which is obtained by substituting $x_1, \ldots, x_m$ by some ground terms $s_1, \ldots, s_m$.

Answer Sets

- The semantics of a positive program P involving variables is determined by the respective ground program $\mathrm{Gnd}(P)$.
- It is possible to view $\mathrm{Gnd}(P)$ as a propositional program and it is necessarily infinite if P has function symbols and variables.

Definition

Let P be a positive program—potentially involving variables. The unique answer set of P is $\mathrm{LM}(\mathrm{Gnd}(P))$.

This set gives also the correctness criterion for query evaluation:

Proposition

Suppose that $Q(\vec{t})$ is an atomic query involving variables $x_1, \ldots, x_n$ for a positive program P. Then $P \models \exists x_1 \cdots \exists x_n Q(\vec{t}) \iff$
there is a ground substitution θ, which replaces each variable x_i with a ground term $t_i \in \mathrm{Hu}(P)$, such that $Q(\vec{t})\theta \in \mathrm{LM}(\mathrm{Gnd}(P))$.

Example

Consider a positive program P with the following rules

$$R(a,c). \qquad R(b,c). \qquad Q(x) \leftarrow R(x,y).$$

1. The ground program over $\mathrm{Hu}(P) = \{a,b,c\}$ contains the rules

$$\begin{array}{llll} R(a,c). & Q(a) \leftarrow R(a,a). & Q(a) \leftarrow R(a,b). & Q(a) \leftarrow R(a,c). \\ R(b,c). & Q(b) \leftarrow R(b,a). & Q(b) \leftarrow R(b,b). & Q(b) \leftarrow R(b,c). \\ & Q(c) \leftarrow R(c,a). & Q(c) \leftarrow R(c,b). & Q(c) \leftarrow R(c,c). \end{array}$$

2. The answer set is $\mathrm{LM}(\mathrm{Gnd}(P)) = \{R(a,c), R(b,c), Q(a), Q(b)\}$.
3. As earlier, this set provides answers to queries as intended:

$$P \models R(a,c)\ ? \quad \textbf{yes} \qquad P \models R(a,d)\ ? \quad \textbf{no}$$
$$P \models Q(a)\ ? \quad \textbf{yes} \qquad P \models Q(c)\ ? \quad \textbf{no}$$

6. EXPRESSIVE POWER

- Rules are expressive enough to cover basic operations on relations as available in standard relational algebra (Structured Query Language, SQL):
  1. Union: $\text{EUNational}(x) \leftarrow \text{Finn}(x).$
     $\text{EUNational}(x) \leftarrow \text{Swede}(x).$
  2. Intersection: $\text{Father}(x) \leftarrow \text{Parent}(x), \text{Man}(x).$
  3. Projection: $\text{Parent}(x) \leftarrow \text{Parent}(x, y).$
  4. Selection: $\text{Millionaire}(x) \leftarrow \text{Assets}(x, y), \text{Greater}(y, 999999).$
  5. Composition: $\text{Result}(x, y) \leftarrow \text{Student}(x, i), \text{Grade}(i, y).$
- Unlike SQL, positive programs enable recursive definitions.
- But the conditions used in the form of rules considered so far cannot refer to complements of relations as in SQL.

Contrast with Relational Algebra

Example

E.g., the transitive closure of a relation is expressible:

$$\text{Connection}(x,y) \leftarrow \text{Flight}(x,y).$$
$$\text{Connection}(x,y) \leftarrow \text{Flight}(x,z), \text{Connection}(z,y).$$

Example

However, it is non-trivial to add negation ($\sim$ below) into rules:

$$\text{Man}(a).\ \ \text{Man}(b).\ \ \text{Man}(c).$$
$$\text{Shaves}(c,x) \leftarrow \text{Man}(x), \sim\text{Shaves}(x,x).$$
$$\text{Shaves}(a,a).$$

OBJECTIVES

- You are able to define minimal models and the least model for positive programs, and to prove simple properties about them.
- You know the interconnection between the least model of a positive program and its logical consequences.
- You are able to construct the least model for the given positive program P by calculating the least fixpoint of T_P.
- You have some preliminary ideas how minimal models are exploited in knowledge representation.
- You are aware of the basic similarities and differences of relational algebra and rule-based languages.

TIME TO PONDER

Consider two positive programs P_1 and P_2 and their union $P_1 \cup P_2$.

Which of the following do hold in general?

1. $\mathrm{LM}(P_1 \cup P_2) \subseteq \mathrm{LM}(P_1) \cup \mathrm{LM}(P_2)$.
2. $\mathrm{LM}(P_1) \cup \mathrm{LM}(P_2) \subseteq \mathrm{LM}(P_1 \cup P_2)$.