Answer Set Programming and Extensions
Unit 1 – Basic Concepts and Properties

Thomas Eiter

Institut für Informationsysteme, TU Wien

VTSA Summer School 2016, Liège, August 29-September 2, 2016

Austrian Science Fund (FWF) grants P24090, P26471, P27730
Unit Outline

1. Background
2. Answer Set Semantics
3. Some Properties
4. ASP Paradigm
5. Computational Complexity
6. Language Extensions
7. Relation to Other Solving Formalisms
Introduction

- Answer Set Programming (ASP): recent problem solving approach
- Term coined by Lifschitz [1999,2002]
- Proposed by other people at about the same time, e.g. [Marek and Truszczyński, 1999], [Niemelä, 1999]
- It has roots in KR, logic programming, and nonmonotonic reasoning
- At an abstract level, relates to Satisfiability (SAT) solving and Constraint Programming (CP)
- Books: [Baral, 2002], [Gebser et al., 2012]
- Compact survey: [Brewka et al., 2011]
- Forthcoming: special issue of AI Magazine on ASP
French Phrases, Italian Soda (*Dell Logic Puzzles*)

- Six people sit at a round table
- Each drinks a different kind of soda
- Each plans to visit a different French-speaking country
- The person who is planning a trip to Quebec, who drank either blueberry or lemon soda, didn’t sit in seat number one.
- Jeanne didn’t sit next to the person who enjoyed the kiwi soda.
- The person who has a plane ticket to Belgium, who sat in seat four or seat five, didn’t order the tangelo soda.
- ...

**Question:**

What is each of them drinking, and where is each of them going?
Wanted!

A general-purpose approach to model and solve such and other problems

Issues:

- Diverse domains
- Constraints
- Incomplete information
- Preferences and priority
- Spatial and temporal reasoning
- ...

Option:

Answer Set Programming (ASP) paradigm!
Logic Programming – Prolog Revisited

1960s/70s: Logic as a programming language (??)

- Breakthrough in Computational Logic by Robinson’s discovery of the Resolution Principle (1965)

Kowalski (1979):

\[
\text{ALGORITHM} = \text{LOGIC} + \text{CONTROL}
\]

- Knowledge for problem solving (LOGIC)
- “Processing” of the knowledge (CONTROL)
Prolog

Prolog = “Programming in Logic”

- Basic data structures: terms
- Programs: rules and facts
- Computing: Queries (goals)
  - Proofs provide answers
  - SLD-resolution
  - unification - basic mechanism to manipulate data structures

Example 1

\[
\text{man(dilbert).}
\]
\[
\text{person(X) } \leftarrow \text{ man(X).}
\]

Query  \( ?= \text{person(X)} \)

Answer  \( X = \text{dilbert} \)

- Extensive use of recursion
The key: techniques to search for proofs

- Understanding of the resolution mechanism is important
- It may make a difference which logically equivalent form is used (e.g., termination).

Example 2

\[
\text{reverse}([X|Y], Z) \leftarrow \text{append}(U, [X], Z), \text{reverse}(Y, U). \quad (1)
\]
\[
\text{vs}
\text{reverse}([X|Y], Z) \leftarrow \text{reverse}(Y, U), \text{append}(U, [X], Z). \quad (2)
\]

Query:  \(? – \text{reverse}([a|X], [b, c, d, b])\)

- (1) yields answer “no”, (2) does not terminate

Is this truly declarative programming?
LP Desiderata

Relieve the programmer from several concerns.

It is desirable that

- the order of subgoals in a rule does not matter;
- the order of program rules does not matter;
- termination is not subject to such order.

“Pure” declarative programming

- Prolog does not satisfy these desiderata
- Satisfied e.g. by the answer set semantics of logic programs
**Horn Logic Programs**

### Definition 3 ((Horn) Logic Program)

A (Horn) logic program $P$ is a finite set of clauses (rules) $r$ of the form

$$ a \leftarrow b_1, \ldots, b_m, \quad (1) $$

where $a, b_1, \ldots, b_m$ are atoms of a first-order language $L$.

- $a$ is the *head* of the rule
- $b_1, \ldots, b_m$ is the *body* of the rule.

- Roughly, (1) can be seen as material implication $(\forall) b_1 \land \cdots \land b_m \rightarrow a$.
- Semantics: use *Herbrand interpretations*, which are sets of ground atoms that are true (terms are interpreted by themselves, no other elements),
- *Models* are such interpretations where $a$ is true whenever $b_1, \ldots, b_m$ are.
- We can reduce $P$ to its *grounding* $\text{grnd}(P) = \bigcup_{r \in P} \text{grnd}(r)$, where $\text{grnd}(r)$ is the set of all ground instances of $r$ (via ground substitutions).
Herbrand Semantics

Definition 4

- Given a logic program $P$, the **Herbrand universe** of $P$, $HU(P)$, is the set of all terms which can be formed from constants and functions symbols in $P$ (resp., the vocabulary, if explicitly known).

- The **Herbrand base** of $P$, $HB(P)$, is the set of all ground atoms which can be formed from predicates and terms $t \in HU(P)$.

- A (Herbrand) **interpretation** is a first-order interpretation $I = (D, \cdot^I)$ of the vocabulary with domain $D = HU(P)$ where each term $t \in HU(P)$ is interpreted by itself, i.e., $t^I = t$.

- $I$ is identified with the set $\{ p(t_1, \ldots, t_n) \in HB(P) \mid \langle t_1^I, \ldots, t_n^I \rangle \in p^I \}$.

Informally, a (Herbrand) interpretation can be seen as a set denoting which ground atoms are true in a given scenario.
Herbrand Semantics /2

Example 5

Program $P$:

\[
\begin{align*}
p(X, Y, Z) &\leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \\
h(X, Z') &\leftarrow p(X, Y, Z'), h(X, Y), t(Z, Z', r). \\
&\quad p(0, 0, b). \quad h(0, 0). \quad t(a, b, r).
\end{align*}
\]

■ Constant symbols: 0, a, b, r.

■ $HU(P)$: \{0, a, b, r\}

■ $HB(P)$:
\[
\begin{align*}
\{ & p(0, 0, 0), p(0, 0, a), \ldots, p(r, r, r), \\
h(0, 0), h(0, a), \ldots, h(r, r, r), \\
t(0, 0, 0), t(0, 0, a), \ldots, t(r, r, r) \}
\end{align*}
\]

■ Some Herbrand interpretations:
\[
\begin{align*}
I_1 &= \emptyset; \quad I_2 = HB(P); \quad I_3 = \{h(0, 0), t(a, b, r), p(0, 0, b)\}.
\end{align*}
\]
Minimal Model Semantics

- A logic program has multiple models in general.
- Select one of these models as the canonical model.
- Commonly accepted: truth of an atom in model $I$ should be “founded” by clauses.

**Example 6**

Given

$$P_1 = \{ a \leftarrow b. \quad b \leftarrow c. \quad c \},$$

truth of $a$ in the model $I = \{a, b, c\}$ is “founded.”

Given

$$P_2 = \{ a \leftarrow b. \quad b \leftarrow a. \quad c \},$$

truth of $a$ in the model $I = \{a, b, c\}$ is not founded.
Minimal Model Semantics (cont’d)

Semantics: Prefer models with true-part as small as possible.

**Definition 7**

A model \( I \) of an LP \( P \) is *minimal*, if no model \( J \subset I \) of \( P \) exists.

**Theorem 8**

*Every* Horn LP \( P \) *has a least (single minimal) model, denoted* \( LM(P) \).

**Example 9**

- For \( P_1 = \{ a \leftarrow b. \ b \leftarrow c. \ c \} \), we have \( LM(P_1) = \{ a, b, c \} \).
- For \( P_2 = \{ a \leftarrow b. \ b \leftarrow a. \ c \} \), we have \( LM(P_2) = \{ c \} \).

- \( LM(P) \) is the canonical model of \( P \) (rule/subgoal order immaterial)
- \( LM(P) \) is the *least fixpoint* of a monotone (even continuous) 1-step consequence operator \( T_P \) on Herbrand interpretations
Fixpoint Characterization

The minimal model can be computed via fixed-point iteration.

**Definition 10 (T_P Operator)**

Let \( T_P : 2^{HB(P)} \rightarrow 2^{HB(P)} \) be defined as

\[
T_P(I) = \left\{ a \mid \text{there exists some } a \leftarrow b_1, \ldots, b_m \text{ in } \text{grnd}(P) \text{ such that } \{b_1, \ldots, b_m\} \subseteq I \right\}.
\]

Notation: \( T_P^0 = \emptyset \), and \( T_P^{i+1} = T_P(T_P^i) \), for \( i \geq 0 \).

**Fundamental result:**

**Theorem 11**

\( T_P \) has a least fixed point, \( \text{lfp}(T_P) \), and the sequence \( \langle T_P^i \rangle \), \( i \geq 0 \), converges to \( \text{lfp}(T_P) \).

Proof: Use the fixed-point theorems of Knaster-Tarski and Kleene.
Fixpoint Characterization /2

Example 12

For \( P_3 = \{ a \leftarrow b. \ b \leftarrow c. \ c \} \), we have

\[
T_{P_3}^0 = \emptyset, \ T_{P_3}^1 = \{c\}, \ T_{P_3}^2 = \{c, b\}, \ T_{P_3}^3 = \{c, b, a\}, \ T_{P_3}^4 = T_{P_3}^3.
\]

Hence \( \text{lfp}(T_{P_3}) = \{c, b, a\} \).

For \( P_4 = \{ a \leftarrow b. \ b \leftarrow a. \ c \} \), we have

\[
T_{P_4}^0 = \emptyset, \ T_{P_4}^1 = \{c\}, \ T_{P_4}^2 = T_{P_4}^1.
\]

Hence \( \text{lfp}(T_{P_4}) = \{c\} \).
Negation in Logic Programs

Why negation?

- Natural linguistic concept
- Facilitates convenient, declarative descriptions (definitions)

E.g., "Men who are not husbands are singles."

Definition 13 (normal logic program)

A normal logic program is a set of rules of the form

\[ a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \quad (n, m \geq 0) \]

where \( a \) and all \( b_i, c_j \) are atoms in a first-order language \( L \).

\text{not} is called "negation as failure", "default negation", or "weak negation"

Things get more complex!
Programs with Negation

Prolog: “not \( \langle X \rangle \)” means “Negation as Failure (to prove to \( \langle X \rangle \))”

Different from negation in classical logic!

Example 14 (Dilbert Program)

\[
\begin{align*}
\text{man}(dilbert). \\
\text{single}(X) \leftarrow \text{man}(X), \text{not} \text{ husband}(X). \\
\text{husband}(X) \leftarrow \text{fail}. \quad \% \text{ fail = "false" in Prolog}
\end{align*}
\]

Query:

\(? - \text{single}(X).\)

Answer:

\(X = dilbert\).
Example 15 (cont’d)

Modifying the last rule of the Dilbert program, we get:

\[
\begin{align*}
\text{man}(\text{dilbert}). \\
\text{single}(X) & \leftarrow \text{man}(X), \text{not husband}(X). \\
\text{husband}(X) & \leftarrow \text{man}(X), \text{not single}(X).
\end{align*}
\]

Query: 

\[
? - \text{single}(X).
\]

Result in Prolog ????

**Problem**: not a single intuitive model!

Two intuitive models:

\[
M_1 = \{\text{man}(\text{dilbert}), \text{single}(\text{dilbert})\}, \quad \text{and} \\
M_2 = \{\text{man}(\text{dilbert}), \text{husband}(\text{dilbert})\}.
\]

Which one to choose?
Semantics of Logic Programs With Negation

- “War of Semantics” in LP (1980/90ies):
  Meaning of programs like the Dilbert example above

- Great Schism: Single model vs. multiple model semantics

- To date:
  - Well-Founded Semantics [van Gelder et al., 1991]
    Partial model: $\text{man}(\text{dilbert})$ is true,
    $\text{single}(\text{dilbert}), \text{husband}(\text{dilbert})$ are unknown
    Alternative models: $M_1 = \{\text{man}(\text{dilbert}), \text{single}(\text{dilbert})\}$,
    $M_2 = \{\text{man}(\text{dilbert}), \text{husband}(\text{dilbert})\}$.

- Agreement for so-called “stratified programs”
  Different selection principles for non-stratified programs
Answer Set Programs

Definition 16 ((disjunctive) answer set program)

A (disjunctive) answer set program $P$ is a set of rules of the form

$$a_1 \lor \cdots \lor a_l \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$$

where all $a_i, b_j, c_k$ are first-order atoms over a first-order vocabulary.

- Notation: $\text{Body}(r) = \{b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n\}$ and $\text{Head}(r) = \{a_1, \ldots, a_k\}$
- In practice, function symbols are restricted or disregarded;
- Generalizes normal programs (single literal head, i.e. $l = 1$)
- Notation: $\text{HB}_P$ is the Herbrand base of $P$, i.e., the set of all ground (variable-free) atoms $p$ with predicates and ground terms constructible from $P$. 
Answer Sets

- Answer Sets are based on Herbrand interpretations.
- They are also called *stable models* and reflect a *closed world perspective*.

**Definition 17 (Satisfaction)**

Consider interpretation $M \subseteq HB_P$: $M$ satisfies

- $a$ (resp. $\neg a$) if $a \in M$ ($a \notin M$) for a ground atom $a$;
- a ground (variable-free) rule $r$,

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

if either (i) $M$ does not satisfy some literal $b_i$ or $\neg c_j$ in $\text{Body}(r)$, or (ii) $M$ satisfies some $a_i \in \text{Head}(r)$;
- a ground program $P$, if $M$ satisfies each $r \in P$;
- a rule $r$ (resp. program $P$), if $M$ satisfies each $r' \in grnd(r)$ (resp. $grnd(P)$).
Answer Sets /2

- For *not* -free ("positive") programs, an intuitive semantics are *minimal models*:

**Definition 18 (Minimal Model)**

An interpretation $M \subseteq HB_P$ is minimal model of $P$, if (i) $M$ satisfies $P$ and (ii) no $N \subset M$ satisfies $P$.

- **Key idea for arbitrary programs:** elimination of *not*

**Definition 19 (Gelfond-Lifschitz (GL) reduct $P^M$)**

Given program $P$, remove from $grnd(P)$

1. every rule $a_1 \lor \cdots \lor a_k \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$ where some $c_i$ is in $M$, and
2. all literals *not* $c_j$ from the remaining rules.

Use $M$ as an *assumption* on how negation finally evaluates
**Definition 20 (Answer Set)**

$M$ is an *answer set* of a program $P$, if $M$ is a minimal model of $P^M$. $\text{AS}(P)$ denotes the set of all answer sets of program $P$.

- $M$ satisfies all rules of $P$
- Moreover, $P$ can “reproduce” $M$ with an *assumption* on how negation finally evaluates (*stability*)
- Note: for disjunction-free $P$, “a minimal” = “the least”
  \[\Rightarrow\] fixpoint construction via operator $T_{P^M}$
- For positive $P$, $P^M = P$, so $\text{AS}(P)$ are the minimal models
- Answer sets have equivalent definitions [Lifschitz, 2008]
- Logical reconstruction: equilibrium logic [Pearce, 2006], based on the *logic of here-and-there* [Heyting, 1930] (= 3-valued Gödel logic)
Example 21 (Dilbert cont’d)

\[ \text{man}(\text{dilbert}) \]  \hspace{1cm} (f_1)
\[ \text{single}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}), \text{not husband}(\text{dilbert}) \]  \hspace{1cm} (r_1)
\[ \text{husband}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}), \text{not single}(\text{dilbert}) \]  \hspace{1cm} (r_2)

Candidate interpretations:

- \( M_1 = \{ \text{man}(\text{dilbert}), \text{single}(\text{dilbert}) \} \),
- \( M_2 = \{ \text{man}(\text{dilbert}), \text{husband}(\text{dilbert}) \} \),
- \( M_3 = \{ \text{man}(\text{dilbert}), \text{single}(\text{dilbert}), \text{husband}(\text{dilbert}) \} \)
- \( M_4 = \{ \text{man}(\text{dilbert}) \} \),

\( M_1 \) and \( M_2 \) are answer sets.

We can equivalently replace \((r_1), (r_2)\) here by

\[ \text{husband}(\text{dilbert}) \lor \text{single}(\text{dilbert}) \leftarrow \text{man}(\text{dilbert}) \]

(more natural reading).
Constraints

 Programs with \( not \) might lack answer sets.

Example 22

\[
P = \{ \ p \leftarrow \text{not} \ p. \ \}\]

NO answer set is possible ("derive \( p \) if it is not derivable")

Is this bad?? **Russell’s Barber Paradox:**

\[
\text{shaves}(X, Y) \leftarrow \text{barber}(X), \text{man}(Y), \text{not shaves}(Y, Y).
\]
\[
\text{man}(bertrand). \ \text{man}(dilbert).
\]
\[
\text{barber}(bertrand).
\]

- Adding \( p \leftarrow q_1, \ldots, q_m, \text{not} \ r_1, \ldots, \text{not} \ r_n, \text{not} \ p. \) to \( P \), where \( p \) is fresh, “kills” all answer sets of \( P \) that:
  1. contain \( q_1, \ldots, q_m \), and
  2. do not contain \( r_1, \ldots, r_n \).

- Short: \( \leftarrow q_1, \ldots, q_m, \text{not} \ r_1, \ldots, \text{not} \ r_n. \) (constraint)
Some Properties of Answer Sets

Proposition 1 (Minimality, Non-monotonicity)

Every answer set \( M \) of \( P \) is a minimal model of \( P \).

E.g. \( P = \{ a \leftarrow \text{not } b \} \): \( M = \{ a \} \) \quad \text{\( P \cup \{ b \leftarrow \} : M = \{ b \} \)}

Proposition 2 (Supportedness)

Given an answer \( M \) of \( P \), for every literal \( a \in M \) there is some rule \( r \) from \( \text{grnd}(P) \) that fires and yields \( a \), i.e. \( M \models \text{Body}(r) \) and \( M \cap \text{Head}(r) = \{ a \} \).

But: stable \( \neq \) minimal + supported!

E.g. \( P = \{ a \leftarrow a; a \leftarrow \text{not } a \} \)
Some Properties of Answer Sets /2

**Proposition 3 (Failure of Cumulativity)**

*From \( a \in M \), for each answer set \( M \) of \( P \), it does not follow that \( P \) and \( P \cup \{ a \leftarrow \} \) have the same answer sets (even if \( P \) has answer sets).*

E.g.

\[
P = \{ \begin{align*} & b \leftarrow \text{not } c; \quad a \leftarrow b; \quad c \leftarrow \text{not } b; \quad a \leftarrow \text{not } a \end{align*} \}
\]

**Proposition 4 (Generalization of Stratified Semantics)**

*If in a normal \( P \) “\( \text{not} \) ” is layered (“\( P \) is stratified”), then \( P \) has a unique answer set \( M^*(P) \), which coincides with the perfect model.*

\( M^*(P) \) is characterized by an iterative fixpoint construction

E.g.

\[
P = \left\{ \begin{align*} & \text{man(dilbert)}. \quad \text{husband}(X) \leftarrow \text{man}(X), \text{married}(X). \quad \text{single}(X) \leftarrow \text{man}(X), \text{not husband}(X). \end{align*} \right\}
\]
ASP Paradigm

General idea: answer sets are solutions!
Reduce solving a problem instance $I$ to computing answer sets of an LP

Problem Instance $I$ \begin{array}{c} \rightarrow \text{Encoding:} \quad \text{Program } P \end{array} \begin{array}{c} \rightarrow \text{ASP Solver} \quad \text{Model(s)} \end{array} \begin{array}{c} \rightarrow \text{Solution(s)} \end{array}

- **Method:**
  1. *encode* $I$ as a (non-monotonic) logic program $P$, such that solutions of $I$ are represented by models of $P$
  2. *compute* some model $M$ of $P$, using an ASP solver
  3. *extract* a solution for $I$ from $M$.

  variant: compute multiple/all models (for multiple/all solutions)

- Often: decompose $I$ into *problem specification* and *data*

- Approach: *guess and check* (alias *generate and test*) plus *auxiliary definitions*
Example: 3-Coloring

Problem specification \( PS \): 3-coloring condition

Problem specification \( P_{PS} \)

\[
\begin{align*}
g(X) \lor r(X) \lor b(X) & \leftarrow \text{node}(X) \quad \{ \text{Guess} \} \\
& \leftarrow b(X), b(Y), \text{edge}(X, Y) \\
& \leftarrow r(X), r(Y), \text{edge}(X, Y) \\
& \leftarrow g(X), g(Y), \text{edge}(X, Y) \\
\end{align*}
\]

Data \( P_D \): Graph \( G = (V, E) \)

\[
P_D = \{ \text{node}(v) \mid v \in V \} \cup \{ \text{edge}(v, w) \mid (v, w) \in E \}.
\]

Correspondence 3-colorings \( \iff \) models:

\( v \in V \) is colored with \( c \in \{ r, g, b \} \) iff \( c(v) \) is in the model of \( P_{PS} \cup P_D \).
Example: 3-Coloring (ctd)

\[ PD = \{ \text{node}(a), \text{node}(b), \text{node}(c), \text{edge}(a, b), \text{edge}(b, c), \text{edge}(a, c) \} \]
Example: Hamiltonian Path/Cycle

**Input:** A directed graph represented by \textit{node}(\_\_) and \textit{edge}(\_,\_\_) and a starting node \textit{start}(\_\_).

**Problem:** Find a path/cycle beginning at the starting node which contains all nodes of the graph.

\[
\begin{align*}
\text{inPath}(X, Y) \lor \text{outPath}(X, Y) & \leftarrow \text{edge}(X, Y). \quad \text{\{Guess\}} \\
\leftarrow \text{inPath}(X, Y), \text{inPath}(X, Y_1), Y \neq Y_1. \\
\leftarrow \text{inPath}(X, Y), \text{inPath}(X_1, Y), X \neq X_1. \\
\leftarrow \text{node}(X), \text{not reached}(X). \\
\text{reached}(X) & \leftarrow \text{start}(X). \\
\text{reached}(X) & \leftarrow \text{reached}(Y), \text{inPath}(Y, X). \quad \text{\{Auxiliary Def.\}} 
\end{align*}
\]
Example: Hamiltonian Path/Cycle

Input: A directed graph represented by node(\_\_) and edge(_, _, _) and a starting node start(\_\_).

Problem: Find a path/cycle beginning at the starting node which contains all nodes of the graph.

\[
\text{inpPath}(X, Y) \lor \text{outPath}(X, Y) \leftarrow \text{edge}(X, Y). \quad \{\text{Guess}\}
\]

\[
\leftarrow \text{inpPath}(X, Y), \text{inpPath}(X, Y_1), Y \neq Y_1.
\leftarrow \text{inpPath}(X, Y), \text{inpPath}(X_1, Y), X \neq X_1.
\leftarrow \text{node}(X), \text{not reached}(X).
\leftarrow \text{not start_reached}.
\]

\[
\text{reached}(X) \leftarrow \text{start}(X).
\text{reached}(X) \leftarrow \text{reached}(Y), \text{inpPath}(Y, X).
\text{start_reached} \leftarrow \text{start}(Y), \text{inpPath}(X, Y).
\]

\[\{\text{Check}\}\]

\[\{\text{Auxiliary Def.}\}\]
Example: Hamiltonian Path/Cycle (ctd)

\[ P_D = \{ \text{node}(a), \text{node}(b), \text{node}(c), \text{node}(d), \text{edge}(a, b), \text{edge}(a, c), \text{edge}(c, b), \text{edge}(b, c), \text{edge}(b, d), \text{edge}(d, c), \text{edge}(d, a), \text{edge}(a, d), \text{start}(a) \} \]
Example: Spanning Tree

Input: A directed graph represented by node(\_) and edge(_, _, _) and a starting node start(\_).

Problem: Find a tree with root at the starting node that spans the graph

\[
inTree(X, Y) \lor outTree(X, Y) \leftarrow \text{edge}(X, Y).
\]

\[
\leftarrow inTree(X, Y), \text{start}(Y).
\]
\[
\leftarrow inTree(X, Y), \text{inTree}(X_1, Y), X \neq X_1.
\]
\[
\leftarrow \text{node}(X), \text{not reached}(X).
\]

\[
\text{reached}(X) \leftarrow \text{start}(X).
\]
\[
\text{reached}(X) \leftarrow \text{reached}(Y), \text{inTree}(Y, X).
\]

\}

Guess

Check

Auxiliary Def.

Change only one constraint in Ham-Path Encoding (and inPath to inTree)
Computational Complexity: Normal Programs

Theorem 23

Deciding whether a normal LP $P$ has some answer set is

- **NP-complete in the propositional case**;
- **NEXPTIME-complete in the datalog (function-free) case**;
- **$\Sigma^1_1$-complete in the general first-order case (Herbrand models)**.

- 3-Colorability is well-known NP-complete problem
- Datalog case: succinct 3-Colorability (graph is represented by Boolean Circuit) is NEXPTIME-complete $\Rightarrow$ circuit emulation
- can express all NP search problems
Complexity of Disjunctive Logic Programs

- Disjunctive Logic properties can express properties outside NP.
- Example: graph 3-uncolorability (co-NP-complete).

\[
\begin{align*}
  b(X) & \lor r(X) \lor g(X) \leftarrow node(X). \quad \text{Guess} \\
  non\_col & \leftarrow r(X), r(Y), edge(X,Y). \\
  non\_col & \leftarrow g(X), g(Y), edge(X,Y). \\
  non\_col & \leftarrow b(X), b(Y), edge(X,Y). \\
  & \leftarrow \text{not} \ non\_col. \\
  r(X) & \leftarrow non\_col, node(X). \\
  g(X) & \leftarrow non\_col, node(X). \\
  b(X) & \leftarrow non\_col, node(X). \\
\end{align*}
\]

- atom \textit{non\_col} flags spoiled guesses
- “saturation rules” permit one answer set \( M_{sat} \) if \( G \) is 3-uncolorable
- no answer set exists if \( G \) is 3-colorable (must derive \textit{non\_col})

- This pattern is known as “saturation technique”
Complexity of Disjunctive Logic Programs /2

- Using saturation, problems beyond NP and co-NP are expressible.
- Example: Subgraph Strong 3-Colorability (SS3COL)

**Instance:** a graph $G$ as above, and a subset $\text{node}_1$ of its nodes.

**Problem:** find a 3-coloring of the subgraph $G_1$ of $G$ on $\text{node}_1$ that can not be extended to a 3-coloring of $G$.

\[
\begin{align*}
&b(X) \lor r(X) \lor g(X) \leftarrow \text{node}(X). \quad \text{Guess} \\
&\leftarrow C(X), C(Y), \text{edge}(X, Y), \text{node}_1(X), \text{node}_1(Y). \\
&\text{non}_{\text{col}} \leftarrow C(X), C(Y), \text{edge}(X, Y), \neg \text{node}_1(X). \\
&\text{non}_{\text{col}} \leftarrow C(X), C(Y), \text{edge}(X, Y), \neg \text{node}_1(Y). \\
&\leftarrow \neg \text{non}_{\text{col}}. \\
&C(X) \leftarrow \text{non}_{\text{col}}, \text{node}(X), \neg \text{node}_1(X). \quad \text{Saturate}
\end{align*}
\]

for $C \in \{r, g, b\}$

- Note: SS3COL is complete for $\Sigma_2^p = \text{NP}^{\text{NP}}$.
Complexity of Disjunctive Logic Programs /3

**Theorem**

Deciding whether a disjunctive logic program $P$ has some answer set is

- $\Sigma^p_2$-complete in the propositional case;
- $\text{NExpTime}^{\text{NP}}$-complete in the datalog case;
- $\Sigma^1_1$-complete in the general first-order case.

- **Note:** no complexity increase in the full FO case (!)
- **Disjunction can be compiled away:**
  - Express answer set existence as $\exists M \forall M' \psi(M, M')$, where $\psi$ is a Boolean combination of existential sentences.
  - rewrite to $\exists M \forall x \forall M' \exists y \psi'(M, x, M', y)$, where $\psi'$ is quantifier-free.
  - Emulate Arithmetic to express $\forall M' \exists y \psi'$ in FOL.
- **Higher complexity with other constructs** (e.g., weak constraints)
- **Survey:** [Dantsin et al., 2001]
ASP Language Extensions

- Many extensions were proposed, some motivated by applications
- Some are syntactic sugar, other strictly add expressiveness
  - strong negation
  - optimization: weak constraints, weight constraints, minimize, to compute optimal / feasible answer sets
  - aggregates
  - preferences: e.g., PLP
  - functions, lists, sets
  - templates (for macros), external functions (dlvhex)
  - ...

- Standard syntax: ASP-Core-2 [Calimeri et al., 2012]
  - e.g. disjunction $\lor$ is written as $|$ 
  - choice construct $m \leq \{a_1; a_2; \ldots; a_k\} \leq n$ to select $n$ to $m$ elements

- Combinations/Interfaces with/to other formalisms
- Front-ends to specific applications (e.g., diagnosis, planning, inheritance reasoning)
Strong Negation

- Strong negation “¬” (also written as “–”) is provided as possibility to express that something is provably false.
- This is different from negation as failure.

Example 24

“At a railroad crossing, cross the rails if no train approaches.”

walk ← at(L), crossing(L), not train_approaches(L).

walk ← at(L), crossing(L), ¬train_approaches(L).

“¬” is syntactic sugar, easily compiled away: for each predicate \( p \),

- replace \( \neg p \) by a fresh predicate \( p_{\neg} \).
- add the constraint \( \leftarrow p(X), p_{\neg}(X) \).
Weak Constraints

- Allow to formalize *optimization problems* in an easy and natural way.
- Integrity constraints vs. weak constraints:
  - integrity constraints “kill” unwanted models;
  - weak constraints express desiderata to satisfy if possible.
- Syntax (DLV):

  \[
  :\sim b_1,\ldots,b_k, \text{not } b_{k+1},\ldots, \text{not } b_m. \quad \text{[Weight: Level]}
  \]

  where
  - all \( b_i \) are atoms (resp. “classical” literals)
  - \( \text{Weight}, \text{Level} \) are numbers (or variables occurring in some \( b_i, i \leq k \), that instantiate to numbers)

- The answer sets of a program \( P \) plus weak constraints \( WC \) are the answer sets \( M \) of \( P \) with least violation of \( WC \), called *best models*.
- ASP-Core-2: variation of syntax
Weak Constraints: Semantics for \((P, WC)\)

Semantics via aggregated violation cost \((WC = \{wc_1, \ldots, wc_n\})\):

\[ wc : \sim b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m. \quad [\text{Weight : Level}] \]

- as usual, consider the grounding \(\text{grnd}(wc)\) of \(wc\)
- Interpretation \(I\) violates a ground \(wc (I \not\models wc)\), if \(\{b_1, \ldots, b_k\} \subseteq I\) and \(I \cap \{b_{k+1}, \ldots, b_m\} = \emptyset\)
- The cost of \(I\) at level \(\ell\) is

\[
c(I, \ell) = \sum_{i=1}^{n} \sum_{(\theta, w) \in \mathcal{V}_i(I, \ell)} w,
\]

where \(\mathcal{V}_i(I, \ell) = \{(\theta, w) \mid wc_i\theta = :\sim B. [w, \ell] \in \text{grnd}(wc_i), I \not\models wc_i\theta\}\)

- \(I\) is safe, if each \(c(I, \ell)\) is well-defined (all \(w\)'s are numbers)

- a safe \(M \in \text{AS}(P)\) dominates a safe \(M' \in \text{AS}(P)\), if \(c(M, \ell) < c(M', \ell)\) for some \(\ell\) and \(c(M, \ell') = c(M', \ell')\) for all \(\ell' > \ell\)

- a safe \(M \in \text{AS}(P)\) is \textbf{best (optimal)}, if no \(M' \in \text{AS}(P)\) dominates \(M\)
Weak Constraints: Examples

Example 25 (Default values for weights and levels)

\[ a \lor b. \quad c : - b. \]
\[ : \sim a. \]
\[ : \sim b. \]
\[ : \sim c. \]

Best model: a
Cost ([Weight:Level]): \<[1:1]\>

Answer set \{b, c\} is discarded because it violates two weak constraints!
Weak Constraints: Examples /2

Example 26 (Weights vs. Levels)

Weights:

\[ a \lor b. \]

\[ \neg a. [1:] \]

\[ \neg a. [1:] \]

\[ \neg b. [2:] \]

Best model: b
Cost ([Weight:Level]): <[2:1]>

Levels:

\[ a \lor b_1 \lor b_2. \]

\[ \neg a. [:1] \]

\[ \neg b_1. [:2] \]

\[ \neg b_2. [:2] \]

Best model: a
Cost ([Weight:Level]): <[1:1],[0:2]>

Best model: a
Cost ([Weight:Level]): <[2:1]>
Weak Constraints with Levels

Levels express the relative importance of the requirements.

Example 27 (Divide employees in two project groups $p_1$ and $p_2$)

1. Skills of group members should be different
2. Persons in the same group should not be married to each other
3. Members of a group should possibly know each other

Requirement (3) is less important than (1) and (2)

assign(X,p1) v assign(X,p2) :- employee(X).

\[\sim assign(X,P), assign(Y,P), X!=Y, same\_skill(X,Y). \ [2]\]
\[\sim assign(X,P), assign(Y,P), X!=Y, married(X,Y). \ [2]\]
\[\sim assign(X,P), assign(Y,P), X=Y, not\_know(X,Y). \ [1]\]
Example: Traveling Salesperson (TSP)

Input: a directed graph represented by $node(_)$, straight connections $edge(_, _, _) \text{ and a starting node } start(_)$.

Problem: find a cheapest roundtrip beginning at the starting node

\[
\begin{aligned}
\text{inPath}(X, Y, C) \lor \text{outPath}(X, Y, C) & \leftarrow \text{edge}(X, Y, C). \quad \text{Guess} \\
\leftarrow \text{inPath}(X, Y, C), \text{inPath}(X, Y_1, C_1), Y \neq Y_1. \\
\leftarrow \text{inPath}(X, Y, C), \text{inPath}(X_1, Y, C_1), X \neq X_1. \\
\leftarrow \text{node}(X), \text{not reached}(X). \\
\leftarrow \text{not start_reached} \text{.} \% \text{for cycle}
\end{aligned}
\]

\[
\begin{aligned}
\text{reached}(X) & \leftarrow \text{start}(X). \\
\text{reached}(X) & \leftarrow \text{reached}(Y), \text{inPath}(Y, X, C). \\
\text{start_reached} & \leftarrow \text{start}(Y), \text{inPath}(X, Y, C).
\end{aligned}
\]

\[
\begin{aligned}
\neg \text{inPath}(X, Y, C).[C:1] & \quad \text{Optimize}
\end{aligned}
\]
Example: Minimum Spanning Tree

Input: A directed graph represented by \( \text{node}(\_), \) weighted edges \( \text{edge}(\_, \_, \_ \), and a starting node \( \text{start}(\_). \)

Problem: Find a minimum spanning tree with root at the starting node

\[
\text{inTree}(X, Y, C) \lor \text{outTree}(X, Y, C) \leftarrow \text{edge}(X, Y, C). \quad \text{Guess}
\]

\[
\leftarrow \text{inTree}(X, Y, C), \ \text{start}(Y).
\]

\[
\leftarrow \text{inTree}(X, Y, C), \ \text{inTree}(X_1, Y, C), \ X \neq X_1.
\]

\[
\leftarrow \text{node}(X), \ \text{not reached}(X).
\]

\[
\text{reached}(X) \leftarrow \text{start}(X).
\]

\[
\text{reached}(X) \leftarrow \text{reached}(Y), \ \text{inTree}(Y, X, C).
\]

\[
\sim \ \text{inPath}(X, Y, C).[C:1] \quad \text{Optimize}
\]
Example: Minimum Spanning Tree (ctd.)

\[ P_D = \{ \text{node}(a), \text{node}(b), \]
\[ \text{node}(c), \text{node}(d), \]
\[ \text{edge}(a, b, 1), \text{edge}(a, c, 1) \]
\[ \text{edge}(c, b, 2), \text{edge}(b, c, 1) \]
\[ \text{edge}(b, d, 1), \text{edge}(c, d, 1) \]
\[ \text{start}(a) \} \]
Aggregates

- Allow arithmetic operations over a set of elements, as e.g. in SQL:

\[
\text{select count(\ast) from empl;}
\]

- ASP provides aggregation functions \#count, \#sum, \#min, \#max

\[
\#\text{count}\{\text{Emp, Dept, Job : empl(Emp, Dept, Job)}\}
\]

- These aggregate functions occur in aggregate atoms in rule bodies

\[
\text{small_dept}(D) \leftarrow \#\text{count}\{E, D : \text{empl}(E, D, J)\} < 10, \text{dept}(D)
\]

- Satisfaction of aggregate atom \#aggr\{e_1; \ldots ; e_n\} \prec u, \ e_i = t_1, \ldots , t_m : l_1, \ldots l_k \text{ in interpretation } I:
  - instantiate the global variables (occurring outside the } e_i) (:= \theta)
  - collect for all full groundings } \theta' \supseteq \theta \text{ s.t. } I \models l_1 \theta', \ldots l_k \theta' \text{ the tuples } (t_1 \theta', \ldots , t_m \theta') (=: T(\theta, I))
  - } I \models \#\text{aggr}\{e_1; \ldots ; e_n\} \prec u \text{ for } \theta \text{ if } \#\text{aggr}(T(\theta, I)) \prec u \theta \text{ holds (for } \#\text{sum}, \text{ use the multiset of all } t_1)
Semantics of Programs with Aggregates

- Programs $P$ are reduced to their groundings $\text{grnd}(P)$

**Aggregate Answer Sets**

Given an interpretation $I$ and a ground program $P$, the *Faber-Leone-Pfeifer (FLP) reduct* $fP^I$ is as follows:

$$fP^M = \{ r \in P \mid r = H \leftarrow B, I \models B \};$$

that is, keep the rules $r$ whose bodies are satisfied.

An interpretation $I$ is an answer of $P$, if $I$ is a minimal model of $fP^I$.

- For ordinary programs $P$, the Gelfond-Lifschitz (GL) reduct $P^I$ and the FLP reduct $fP^I$ are equivalent.
- For programs $P$ with aggregates, every answer set wrt. $fP^I$ is one wrt. the naturally extended $P^I$, but not vice versa.
- FLP-answer sets are minimal models of $P$, GL-answer sets not necessarily.
Example 28 (Minimum Spanning Tree)

with aggregates and weak constraints

% Guess the edges that are part of the tree.
inTree(X,Y,C) v outTree(X,Y,C) :- edge(X,Y,C).

% Check that we are really dealing with a tree!
:- start(R), not #count{X : inTree(X,R,C)} = 0.
:- edge(_,Y,_), not start(Y),
    not #count{X : inTree(X,Y,C)} = 1.
% Note: ensures also that each node
% in the graph is reached.

% Nothing in life is free.
~ inTree(X,Y,C). [C:1]
Example: Seating Problem

**Problem:** Given some tables of a given number of chairs each, generate a sitting arrangement for a number of given guests, such that:

- people liking each other should sit at the same table, and
- people disliking each other should not sit at the same table.

\[
\begin{align*}
\text{at}(P, T) \lor \text{not at}(P, T) & :\text{ person}(P), \text{ table}(T). \\
\text{:- table}(T), \text{nchairs}(C), \text{not count}\{P : \text{at}(P, T)\} & \leq C. \\
\text{:- person}(P), \text{not count}\{T : \text{at}(P, T)\} & = 1. \\
\text{:- like}(P_1, P_2), \text{at}(P_1, T), \text{not at}(P_2, T). \\
\text{:- dislike}(P_1, P_2), \text{at}(P_1, T), \text{at}(P_2, T). 
\end{align*}
\]
Example: Seating Problem (ctd.)

\[ P_D = \{ \text{person}(p1), \text{person}(p2), \text{person}(p3), \text{person}(p4), \text{table}(t1), \text{table}(t2), \text{nchairs}(4), \text{like}(p1, p2), \text{dislike}(p1, p3) \} \]
ASP vs. SAT Solving

Benefits of ASP compared to SAT Solving:
- uniform encodings (data + generic part)
- structured representation
- elaboration tolerance
- expressiveness (transitive closure, aggregates, negation as failure etc)
- variety of reasoning tasks

Comparisons of ASP/SAT Efficiency
- early work e.g. [Arieli et al., 2004]
- ASP solvers (clasp) in some contexts competitive
ASP vs. Constraint Programming (CP)

- **Benefits of ASP compared to CP:**
  - disjunctive constraints
  - recursive constraints
  - similar wrt. SAT, nonmonotonic features

- **Disadvantages of ASP compared to CP:**
  - grounding of constraints (no symbolic treatment)

  ⇒ Constraint Answer Set Programming: Integrate ASP + CP

- **Early comparisons of ASP / CP (modeling, efficiency):**
  - Cadoli *et al.* [2006], ECAI
  - Dovier *et al.* [2007], AAAI
  - Coban *et al.* [2008], LaSh Workshop
References I


Elvin Coban, Esra Erdem, and Ferhan Türe. 

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. 
Complexity and Expressive Power of Logic Programming. 

Agostino Dovier, Andrea Formisano, and Enrico Pontelli. 

Thomas Eiter, Georg Gottlob, and Heikki Mannila. 
Disjunctive Datalog. 

Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. 
Answer set programming: A primer. 


References IV


Vladimir Lifschitz, David Pearce, and Agustín Valverde.
A characterization of strong equivalence for logic programs with variables.

Vladimir Lifschitz.
Answer Set Planning.

Vladimir Lifschitz.
Answer Set Programming and Plan Generation.

Vladimir Lifschitz.
Twelve definitions of a stable model.

Victor W. Marek and Miroslaw Truszczyński.
Stable Models and an Alternative Logic Programming Paradigm.
Ilkka Niemelä.
Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.

David Pearce.
Equilibrium logic.

J.S. Schlipf.
Complexity and Undecidability Results in Logic Programming.

A. van Gelder, K.A. Ross, and J.S. Schlipf.
The Well-Founded Semantics for General Logic Programs.

Stefan Woltran.
A common view on strong, uniform, and other notions of equivalence in answer-set programming.
Assignments

Optimal Golomb Ruler (OGR)

- **Problem:** Place a given number of marks on a ruler, such that no two pairs of marks measure the same distance, and the length of the ruler is minimal.

- **Applications:** antenna design, mobile communication technology

% Example input for an OGR of size 4

```prolog
position(0..10).
mark(1..4).
```

**Bonus:** Ensure only *perfect optimal* Golomb Rulers are calculated, i.e., all distances 1, 2, 3 ..., \( n \) are represented
Assignments

Evacuation route planning

Input: A directed graph $G$ represented by $node(\_)$ and $edge(\_, \_)$ and exit nodes $exit(\_)$.

Problem: Find for each node a path to some exit node

- The exit nodes are sinks in $G$, i.e., have no outgoing edges
- Different optimization criteria might apply:
  1. each node can reach some exit within $k$ steps;
  2. edges are weighted, as least cost path is desired;
  3. like before, but the capacity of the exits is bounded (each can be chosen for at most $c$ nodes);
  4. . .
Equilibrium Logic

Basis for semantic characterization of answer sets:

logic of here-and-there (HT) [Heyting, 1930]

- 2-worlds intuitionistic logic (an intermediate logic between intuitionistic and classical logic)
- HT logic coincides with 3-valued Gödel logic
- Restrict attention to program-like theories $T$ (sets of formulas)

$$b_1 \land \ldots \land b_m \land \neg b_{m+1} \land \ldots \land \neg b_n \rightarrow a_1 \lor \ldots \lor a_l,$$

- Semantically characterized by Kripke models of two worlds, $h(\text{here}) \leq t(\text{here})$

⇒ Prop. HT-interpretations are pairs $(X, Y)$ of sets of atoms, s.t. $X \subseteq Y$

Intuitively, $Y$ says what is false, and $X$ what is true based on assumption of $Y$; thus $Y$ deals with GL-reduct $T^Y$
Satisfaction

Satisfaction at the world $w$ of an HT model $M = (X, Y)$:

Notation: $(X, Y) \,(X, Y) | h = X, \,(X, Y) | t = Y$

1. $M, w \models a$ if $a \in M|w$, for any atom $a$,
2. $M, w \not\models \bot$,
3. $M, w \models \phi \land \psi$ if $M, w \models \phi$ and $M, w \models \psi$,
4. $M, w \models \phi \lor \psi$ if $M, w \models \phi$ or $M, w \models \psi$,
5. $M, w \models \phi \rightarrow \psi$ if for every $w' \geq w$: $M|w' \not\models \phi$ or $M|w' \models \psi$,

where $\neg \phi$ is $\phi \rightarrow \bot$

$M \models \phi$ iff $M, w \models \phi$ for all $w \in \{h, t\}$

As $M, w \models \phi$ is monotonic in $w$, $M \models \phi$ iff $M, h \models \phi$

Thus $M \models \neg \phi$ iff $M|t \not\models \phi$. 
Satisfaction (cont’d)

- Alternatively define \( M \models \phi, M = (X, Y) \), straight:
  1. \((X, Y) \models a\) if \(a \in X\), for any atom \(a\),
  2. \((X, Y) \not\models \bot\),
  3. \((X, Y) \models \phi \land \psi\) if \((X, Y) \models \phi\) and \((X, Y) \models \psi\),
  4. \((X, Y) \models \phi \lor \psi\) if \((X, Y) \models \phi\) or \((X, Y) \models \psi\),
  5. \((X, Y) \models \phi \rightarrow \psi\) if (i) \((X, Y) \not\models \phi\) or \((X, Y) \models \psi\), and
     (ii) \(Y \models \phi \rightarrow \psi\).

where \(\neg \phi\) is \(\phi \rightarrow \bot\); thus \((X, Y) \models \neg \phi\) iff \(Y \not\models \phi\).

- An HT-interpretation \((X, Y)\) satisfies a theory \(T\), iff it satisfies all formulas

- Axiomatic proof system: e.g. [Lifschitz et al., 2007]
Equilibrium Models

Definition 29

An HT-interpretation \((Y, Y)\) is an *equilibrium model* of a theory \(T\), iff \((Y, Y) \models T\) and for all \(X \subset Y\), \((X, Y) \not\models T\). (*h-minimality*)

Example 30

- \(P = \{a \leftarrow \text{not } b; \ b \leftarrow \text{not } a\}\) has the eq. models \((a, a)\) and \((b, b)\)
- \(P = \{a \leftarrow \text{not } a\}\) has no equilibrium model

Theorem 31 ([Pearce, 2006])

\(M\) is an answer set of \(P\) iff \((M, M)\) is an equilibrium model of \(P\).

The use of HT/Equilibrium Logic has many attractive features:

- easy generalization to richer syntax (strong negation, nesting etc).
- lifts to non-ground programs (under suitable provisos)
- fruitful base for *paracoherent* semantics (deal with inconsistency)
Application: Equivalence in ASP

Problem: Replacement property (substitution) does not hold

If $P \equiv P'$ then $Q \cup P \equiv Q \cup P'$

Example 32 (Counterexample)

$P = \{ a \leftarrow \neg b \}$, $P' = \{ a \}$, $Q = \{ b \leftarrow \neg a \}$.

- $P$ and $P'$ have the same answer set $\{a\}$
- $Q \cup P'$ has the answer set $\{a\}$, but $Q \cup P$ the answer sets $\{a\}$, $\{b\}$

Definition 33 (Strong equivalence [Lifschitz et al., 2001])

$P \equiv_s P'$ iff $Q \cup P \equiv Q \cup P'$ for all programs $Q$

- That is, $P$ and $P'$ have the same answer set in each context
Application: Equivalence in ASP (ctd)

Theorem 34

$HT$ captures strong equivalence: $P \equiv_s Q$ iff $\models_{HT} P \leftrightarrow Q$

- Thus, $HT$ logic may be viewed as *monotonic core* of ASP

- Refined notions (survey [Woltran, 2008]):
  
  *uniform equivalence* (restrict $Q$ to sets of facts), *modular equivalence*, *projection equivalence*, *relativized hyperequivalence*, ...

- computational complexity may be high, undecidable for some notions
Complexity: Propositional Case

Claim: For normal propositional $P$, answer set existence is NP-complete.

Proof (Sketch):

- **Membership:** Guess an answer set $M$ of $P$.
  Computing $P^M$ and testing whether $M$ is the least model of $P^M$ is polynomial ($P^M$ is Horn).

- **Hardness:** Reduce SAT instance $E = c_1 \land \cdots \land c_m$, where $c_i = l_{i,1} \lor \cdots \lor l_{i,k_i}$, and each $l_{i,j}$ is a literal over $X = \{x_1, \ldots, x_n\}$.
  Construct the following program $P$:

  \[
  \begin{align*}
  x_i & \leftarrow \text{not } \neg x_i & \text{for each } x_i \in X \\
  \neg x_i & \leftarrow \text{not } x_i \\
  c_i & \leftarrow l_{i,j} & \text{for all } l_{i,j} \\
  sat & \leftarrow c_1, \ldots, c_m \\
  sat & \leftarrow \text{not } sat
  \end{align*}
  \]

  where the $c_i$ and $sat$ are new atoms.

  Then $P$ has some answer set iff $E$ is satisfiable.
Complexity: Datalog Case

- By simple grounding, we obtain that for datalog, deciding answer set existence and brave reasoning are in \( \text{NEXPTime} \).

- The problems are also \( \text{NEXPTime} \)-hard.

- Proof: e.g. by encodings of exponential-time NDTMs, cf. [Dantsin et al., 2001]

- Alternative proof technique: Complexity “upgrading” (Kolaitis and Papadimitriou [1991], Gottlob et al. [1997, 1999]):
  - Problem input \( I \) (e.g., a SAT instance) is not a string, but a Boolean circuit describing the bits of the string \( I \)
  - This representation can be exponentially smaller (sometimes)
  - Simulate circuit using a datalog program to “explode” succinct representation
**Complexity: FO Logic Programs**

**Claim:** Deciding whether a normal logic program $P$ with function symbols has some answer set is $\Sigma_1^1$-complete.

**Note:** $\Sigma_1^1$ is in the *Analytic Hierarchy*, and more expressive than the classes $\Sigma_n^0$, $n \geq 0$, in the *Arithmetic Hierarchy*.

Cautious reasoning from positive normal programs is r.e. ($\Sigma_1^0$)-complete.

**Proof** (rough sketch; for more, see [Schlipf, 1995]):

- answer set existence is expressible by a second-order sentence of form $\exists T \forall x \exists y \phi(T, x, y)$, where $\phi(T, x, y)$ is quantifier-free.

- Every $\Sigma_1^1$ sentence is convertible to a sentence $\Phi$ of this form (by second-order Skolemization)
  $\Phi$ is expressible (over the Herbrand universe) by answer set existence.
Suppose wlog $\phi_i(Tx, y) = \bigvee_{i=1}^{n} \phi_i(T, x, y)$ is in DNF.

Then, the program

\[
\begin{align*}
T(x) & \leftarrow \text{not } T'(x) \quad \text{for each } T \in T \\
T'(x) & \leftarrow \text{not } T(x) \\
sat(x) & \leftarrow \phi^*_i(T, x, y) \quad \text{for each } i = 1, \ldots, n \\
sat(x) & \leftarrow \text{not } sat(x) \\
eq(x, x) & \leftarrow
\end{align*}
\]

where $\phi^*(T, x, y)$ is $\phi(T, x, y)$ but

- "\neg" is replaced by "not", and
- "=" is replaced by "eq" (if present),

has some answer set iff $\Phi$ is true on its Herbrand universe.
Disjunctive Programs: Propositional Case

**Claim**: For disjunctive propositional $P$, answer set existence is $\Sigma^p_2$-complete.

**Proof (Sketch):**

- **Membership**: Guess an answer set $M$ of $P$
  
  Computing $P^M$ and testing whether $M$ is a minimal model of $P^M$ is polynomial with an NP oracle:
  
  Ask the oracle whether some $N \subset M$ satisfies $P^M$ (this is in NP)
  
  This yields an $\text{NP}^{\text{NP}} = \Sigma^p_2$ algorithm.

- **Hardness**: Reduce 2-QBFSAT, where instances are of quantified Boolean formulas of the form

  $$\Phi = \exists x_1 \cdots \exists x_n \forall y_1 \cdots \forall y_l F$$

  and $F = d_1 \lor \cdots \lor d_m$ is a DNF, where $d_i = l_{i,1} \land \cdots \land l_{i,k_i}$, and each $l_{i,j}$ is a literal over the atoms $x_i$ and $y_j$ (without loss of generality, $k_i = 3$, denoted 2-QBF3SAT).
Disjunctive Programs: Propositional Case

Construct the following program $P$ from $\Phi$

\[
\begin{align*}
  x_i \vee \neg x_i & \leftarrow \quad \text{for each } x_i \\
  y_i \vee y'_i & \leftarrow \quad \text{for each } y_j \\
  & \quad \text{sat} \leftarrow \sigma(l_{i,1}), \ldots, \sigma(l_{i,k_i}) \quad \text{for all } i \\
  & \quad y_i \leftarrow \text{sat} \quad \text{for each } y_j \\
  & \quad y'_i \leftarrow \text{sat} \\
  & \quad \text{sat} \leftarrow \text{not sat}
\end{align*}
\]

where $sat$, all $y'_i$ are new atoms, and $\sigma(l_{i,j})$ replaces $l_{i,j}$ of form $\neg y_j$ with $y'_j$.

Then $P$ has some answer set iff $\Phi$ evaluates to true.

- $sat$ must be contained in every answer set
- Informally, the disjunctive rules create an assignment to $x_i$ resp. $y_j$.
- Upon an assignment to all $x_i$, $sat$ must be derivable no matter what assignment to the $y_j$ is given
- By the replacement $\sigma(\cdot)$, the search space of such assignments are the subsets of $\{y_j, y'_j \mid 1 \leq j \leq m\}$ satisfying the rules $y_j \vee y'_j \leftarrow$. 
Aggregate Atoms

Definition 35

An aggregate atom has the form

$$
\#aggr\{e_1; \ldots; e_n\} \prec u
$$

where

- $\#aggr \in \{\#\text{count}, \#\text{sum}, \#\text{min}, \#\text{max}\}$;
- each $e_i$ is an expression $t_1, \ldots, t_m : l_1, \ldots, l_k$, $m, k \geq 0$ where each $t_j$ is a term and each $l_h$ a not-literal;
- $\prec \in \{<, \leq, =\neq, >, \geq\}$

- $u \prec \#aggr\{E\}$ and $u_1 \prec \#aggr\{E\} \prec u_2$ available as shortcuts
- aggregate atoms may occur under not,
- DLV: $n = 1$, also $\#\text{times}$ (product)
Aggregate Atoms /2

Example 36

- \#\text{count}\{\text{Emp}, \text{Dept} : \text{empl}(\text{Emp}, \text{Dept}, \text{Job})\} < 10
- \neg \#\text{count}\{\text{Emp}, \text{Dept} : \text{empl}(\text{Emp}, \text{Dept}, \text{Job})\} < 10
- 20 \leq \#\text{sum}\{\text{Sal}, \text{Emp} : \text{sal}(\text{Emp}, \text{Sal})\} \leq 50

Aggregate atoms may occur in rule bodies:

Example 37

\text{small\_dept}(D) \leftarrow \#\text{count}\{E, D : \text{empl}(E, D, J)\} < 10, \text{dept}(D)

- Informally, singles out the small departments as those having less than 10 employees
Aggregate Atom Semantics

- Value of a ground aggregate atom \( a = \#aggr\{E\} < u \), where \( E = e_1; \ldots; e_n \), in an interpretation \( I \):
  - \( eval(E, I) = \{(t_1, \ldots, t_m) \mid t_1, \ldots, t_m : l_1, \ldots l_k \text{ occurs in } E, I \models = l_1, \ldots l_k\} \)
  - \( I \) satisfies \( a(I \models = a) \), if \( \#aggr(eval(E, I)) < u \) is true wrt. \( I \), where
    - \( \#count(T) = |T| \),
    - \( \#sum(T) = \sum_{(t_1, \ldots, t_m) \in T, t_i \text{ is integer } t_i} t_1 \),
    - \( \#min(T) = \min \{t_1 \mid (t_1, \ldots, t_m) \in T\} \)
    - \( \#max(T) = \max \{t_1 \mid (t_1, \ldots, t_m) \in T\} \)
- for non-ground \( a \), instantiation \( inst(a, V) \) relative to a set of global variables \( V \), is as usual but no \( X \in V \) is replaced
- for a ground substitution \( \theta \) for \( V \), \( I \models = a\theta \) if
  - \( I \models = \#aggr\{inst(E)\theta\} < u\theta \)
- within a rule \( r \), the global vars of \( a \) are those occurring outside \( E \).