Answer Set Programming (ASP)

Jef Wijsen

UMONS

February 27, 2024

Table of Contents

Definitions I

- We assume a set σ of symbols, also called atoms.
- A positive literal is an atom; a negative literal is the negation of an atom.
- A rule r is an expression

$$h \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m$$

  where $h, a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms.
- If $m = n = 0$, we simply write h.
- We will mostly treat the body of such a rule as a set of literals $B = \{a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m\}$, and define $B^+ := \{a_1, \ldots a_n\}$ and $B^- := \{b_1, \ldots, b_m\}$.
- An interpretation I is a set of atoms. We write $I \models B$ if $B^+ \subseteq I$ and $B^- \cap I = \emptyset$.
- A program is a set of rules.

Definitions II

- An interpretation M is a model of a program P if for every rule $h \leftarrow B$ of P, if $M \models B$, then $h \in M$.
- A model is minimal if no proper subset of it is a model.
- A model M is supported if for every $h \in M$, there is a rule $h \leftarrow B$ such that $M \models B$.

Example

Let $P_1 = \{b \leftarrow \neg a\}$.

- $\{\}$ is not a model;
- $\{b\}$ is a model that is minimal and supported;
- $\{a\}$ is a model that is minimal but not supported; and
- $\{a, b\}$ is a model that is neither minimal nor supported.

The precedence graph:

$$\begin{tikzcd} b \arrow[r, bend left, "-"] & a \end{tikzcd}$$

Negation is stratified.

Example

Let $P_2 = \left\{ \begin{array}{l} b \leftarrow a \\ a \leftarrow b \end{array} \right.$

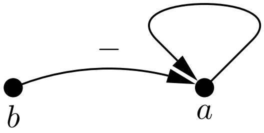
- $\{\}$ is a model that is minimal and supported;
- $\{a\}$ is not a model;
- $\{b\}$ is not a model; and
- $\{a, b\}$ is a model that is supported but not minimal.

Example

Let $P_3 = \left\{ \begin{array}{l} b \leftarrow \neg a \\ a \leftarrow a \end{array} \right.$

- $\{\}$ is not a model;
- $\{a\}$ is a model that is minimal and supported;
- $\{b\}$ is a model that is minimal and supported; and
- $\{a, b\}$ is a model that is neither minimal nor supported.

But intuitively, $\{a\}$ is self-supporting, (i.e., supported through a cyclic derivation).



Negation is stratified.

Stable model (without using B^+ or B^-)

Definition 1

The Gelfond-Lifschitz transform (or reduct) of a program P with respect to an interpretation M, denoted P^M, is a negation-free program constructed as follows. For every rule of P, of the form

$$h \leftarrow a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m,$$

do the following:

- if $m \geq 1$ **and** M contains some b_i $(1 \leq i \leq m)$, then remove the rule;
- otherwise add $h \leftarrow a_1, \ldots, a_n$ to P^M.

An interpretation M is a stable model of P if it is the unique minimal model of P^M.

- Note: if P is negation-free, then $P^M = P$.

Stable model: Some intuition

$$h \leftarrow a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m,$$

do the following:

- if $m \geq 1$ **and** M contains some b_i $(1 \leq i \leq m)$, then remove the rule;

For M to be stable, the reduct P^M must contain a rule that allows us to derive b_i. Such a rule must be of the form:

$$b_i \leftarrow c_1, \ldots, c_\ell.$$

Note: If $\ell = 0$, then b_i is given as a fact.

Recall that P^M contains no negation.

Stable model (using B^+ and B^-)

Definition 2

The Gelfond-Lifschitz transform (or reduct) of a program P with respect to an interpretation M, denoted P^M, is a negation-free program constructed as follows: for every rule $h \leftarrow B$ of P,

- if $B^- \cap M \neq \emptyset$, then remove the entire rule; and
- if $B^- \cap M = \emptyset$, then remove the negative literals from the rule.

An interpretation M is a stable model of P if it is the unique minimal model of P^M.

- Informally, the Gelfond-Lifschitz transform first removes every rule $h \leftarrow B$ such that $M \not\models B^-$ (whence $M \models \{h \leftarrow B\}$), and then removes all negative literals from the remaining rules.
- Why does P^M have a unique minimal model? See *Bases de Données II*.
- But is the unique minimal model of P^M necessarily also a model of P? We will see shortly that this is indeed the case.

Example

Let $P_3 = \left\{ \begin{array}{l} b \leftarrow \neg a \\ a \leftarrow a \end{array} \right.$

We have $P_3^{\{a\}} = \left\{ \begin{array}{l} \cancel{b \leftarrow \neg a} \\ a \leftarrow a \end{array} \right. = \left\{ \begin{array}{l} \\ a \leftarrow a \end{array} \right.$

We have $P_3^{\{b\}} = \left\{ \begin{array}{l} b \leftarrow \cancel{\neg a} \\ a \leftarrow a \end{array} \right. = \left\{ \begin{array}{l} b. \\ a \leftarrow a \end{array} \right.$

- $\{a\}$ is a model that is minimal and supported, but not stable;
- $\{b\}$ is a stable model.

Informally, atoms in a stable model must be **derivably** true[a] once we accept (without derivation!) that atoms not in the model are false.

[a] and therefore must occur in rule heads

- $\{a\}$ is not stable, because a cannot be derived from $\neg b$;
- $\{b\}$ is a stable, because b can be derived from $\neg a$.

See [Tru18, Theorem 2.10]:

Theorem 3

If M is a stable model of a variable-free program P, then M is a minimal model of P that, moreover, is supported.

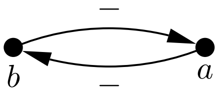
The converse does not hold (see previous example).

Two Stable Models

Let $P_4 = \left\{ \begin{array}{l} b \leftarrow \neg a \\ a \leftarrow \neg b \end{array} \right.$

We have $P_4^{\{a\}} = \left\{ \begin{array}{l} \cancel{b \leftarrow \neg a} \\ a \leftarrow \cancel{\neg b} \end{array} \right. = \left\{ \begin{array}{l} \\ a. \end{array} \right.$

- $\{a\}$ is a stable model; and
- by symmetry, $\{b\}$ is a stable model.


$$\begin{tikzcd} b \arrow[r, bend left, "-"] & a \arrow[l, bend left, "-"] \end{tikzcd}$$

Negation is not stratified.

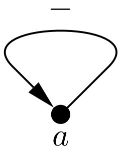
Informally, a is derivably true once we accept that b is false (and vice versa).

No Stable Model

Let $P_5 = \{a \leftarrow \neg a\}$.

- $\{\}$ is not a model;
- $\{a\}$ is a model that is minimal but not supported (and therefore not stable by Theorem 3).

By Definition 2, the model $\{a\}$ is not stable, because it is not the unique minimal model of $P_5^{\{a\}} = \{\}$.



Negation is not stratified.

The Empty Program

Let $P_0 = \{\}$, the empty program. Show the following:

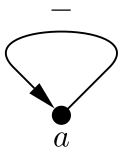
- every interpretation is a model of P_0; and
- the empty model is the only stable model of P_0.

Example

Let $P_6 = \{a \leftarrow a \land \neg a\}$.

The model $\{\}$ is stable because it is the unique minimal model of $P_6^{\{\}} = \{a \leftarrow a\}$.

The precedence graph:



Negation is not stratified.

Caveat

clingo ASP allows negative literals in rule heads (and double negations). For example,

```
not cold :- sunny.
cold :- not not cold.
sunny :- not cold.
```

We will only consider negation-free rule heads.

Different but Equivalent Characterization of Stable Models

Definition 4

The Faber-Pfeifer-Leone transform of a program P with respect to an interpretation M, denoted $fpl(P, M)$, is the program that contains all (and only) the rules $h \leftarrow B$ of P for which $M \models B$ holds true.

Theorem 5 (Theorem 3.6 in [FPL11])

M is a stable model of P $\iff$ M is a minimal (w.r.t. $\subseteq$) model of $fpl(P, M)$.

Comparison

Let r be the following rule:

$$h \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m$$

For any program P containing r,

- if $b_1 \notin M$ and $\cdots$ and $b_m \notin M$, then
  - if $a_1, \ldots, a_n \in M$, then $fpl(P, M)$ contains r, and P^M contains $h \leftarrow a_1 \wedge \cdots \wedge a_n$;
  - if $a_i \notin M$ for some i, then $fpl(P, M)$ does not contain r, but P^M still contains $h \leftarrow a_1 \wedge \cdots \wedge a_n$;
- if $b_j \in M$ for some j, then P^M and $fpl(P, M)$ contain no rules corresponding to r.

Example

Let $P_3 = \left\{ \begin{array}{l} b \leftarrow \neg a \\ a \leftarrow a \end{array} \right.$

We have $fpl(P_3, \{a\}) = \left\{ \begin{array}{l} \cancel{b \leftarrow \neg a} \\ a \leftarrow a \end{array} \right. = \left\{ \begin{array}{l} \\ a \leftarrow a \end{array} \right.$

We have $fpl(P_3, \{b\}) = \left\{ \begin{array}{l} b \leftarrow \neg a \\ \cancel{a \leftarrow a} \end{array} \right. = \left\{ \begin{array}{l} b \leftarrow \neg a \\ \\ \end{array} \right.$

- $\{a\}$ is not a minimal model of $fpl(P_3, \{a\})$;
- $\{b\}$ is a minimal model of $fpl(P_3, \{b\})$.

Rules with Empty Head = Constraints

A rule

$$\leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m \tag{1}$$

can be regarded as shorthand for

$$h \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m \wedge \neg h \tag{2}$$

where h is a fresh atom not occurring elsewhere.

Rationale: Let M be an interpretation for a program containing (2). Assume that $\{a_1, \ldots, a_n\} \subseteq M$ and $\{b_1, \ldots, b_m\} \cap M = \emptyset$. Two cases can occur:

Case that $h \notin M$. Then the Gelfond-Lifschitz transform contains $h \leftarrow a_1 \wedge \cdots \wedge a_n$. Since $h \notin M$, M is not stable.

Case that $h \in M$. Then the Gelfond-Lifschitz transform contains no rule with head h. Since h is not supported, M is not stable.

To conclude, there is no stable model that satsfies the body of rule (1).

Indexing

$$\begin{cases} b \\ a1 \leftarrow b \\ a2 \leftarrow b \\ \quad\vdots \\ a99 \leftarrow b \end{cases}$$

In clingo ASP, one can write:

```
b.
a(I) :- b, I=1..99.
```

Choice

In clingo ASP, one can write:

```
b.
3 { a(I) : I=1..99 } 7 :- b.
```

Every stable model contains b, and contains at least 3 and at most 7 atoms of $\{a_1, \ldots, a_{99}\}$. For example, $\{b, a_{11}, a_{23}, a_{37}, a_{41}\}$ is a stable model.

In principle, such a rule can be written as a normal variable-free logic program as previously defined. For example,

```
b.
1 { a(I) : I=1..2 } 1 :- b.
```

has the same stable models as:

$$\begin{cases} b \\ a1 \leftarrow b, \neg a2 \\ a2 \leftarrow b, \neg a1 \end{cases}$$

Sudoku

Assume a classical Sudoku. Number the nine 3×3-squares as follows:

1	2	3
4	5	6
7	8	9

Number the nine cells within each 3×3-square in the same way. The jth cell within the ith 3×3-square has index (i, j). Each cell (i, j) can store one number $k \in \{1, \ldots, 9\}$.
We define $9 \times 9 \times 9$ propositional variables. The variable a_{ijk} is *true* if cell (i, j) stores k, and is *false* otherwise.

A Sudoku Program

```
% Every cell stores at least one number.
1 { a(I,J,K) : K=1..9 } :- I=1..9, J=1..9.
% Within each 3x3-square, every number occurs at least once.
1 { a(I,J,K) : J=1..9 } :- I=1..9, K=1..9.
% Indices belonging to a Same Column.
isc(1,4). isc(4,7). isc(1,7).
isc(2,5). isc(5,8). isc(2,8). isc(3,6). isc(6,9). isc(3,9).
isc(I,J) :- isc(J,I).
isc(I,I) :- I=1..9.
% Indices belonging to a Same Row.
isr(1,2). isr(2,3). isr(1,3).
isr(4,5). isr(5,6). isr(4,6). isr(7,8). isr(8,9). isr(7,9).
isr(I,J) :- isr(J,I).
isr(I,I) :- I=1..9.
% Distinct cells in same row/column cannot store the same number.
eq(I,J,I,J) :- I=1..9, J=1..9.
:- a(I,J,K), a(II,JJ,K), isc(I,II), isc(J,JJ), not eq(I,J,II,JJ).
:- a(I,J,K), a(II,JJ,K), isr(I,II), isr(J,JJ), not eq(I,J,II,JJ).
```

The Prefilled Cells

```
a(1,2,6). a(1,8,9). a(1,9,3).
a(2,4,4). a(2,6,3). a(2,8,8). a(2,9,5).
a(4,2,3). a(4,3,6). a(4,5,5). a(4,6,1). a(4,7,4).
a(5,3,4). a(5,7,2).
a(6,3,5). a(6,4,4). a(6,5,8). a(6,7,3). a(6,8,9).
a(8,1,5). a(8,2,6). a(8,4,1). a(8,6,2).
a(9,1,7). a(9,2,3). a(9,8,5).
```

	6							
			4		3			
	9	3		8	5			
	3	6			4			5
	5	1				4	8	
4			2			3	9	
			5	6		7	3	
			1		2			
							5	

Wrap Up

- For every program P and interpretation M, the following are equivalent:
  1. M is a stable model of P.
  2. M is the unique minimal model of P^M.
  3. M is a minimal model of $\mathit{fpl}(P, M)$.
- **Property.** Every atom in a stable model is derivably *true* when atoms not in the model are fixed to *false*. For example, the atom a is derivably *true* in $\{a \leftarrow \neg b,\ b \leftarrow \neg a\}$ when we fix b to *false*.
- **Property.** If $h \leftarrow B \wedge \neg h$ is the only rule with head h in a given program P, then no stable model of P satisfies B.
  A syntax shorthand for this rule is: $\leftarrow B$.
- **Corollary.** If a program P with rule $h \leftarrow \neg h$ has a stable model, then P must contain at least one other rule with head h.

ASP vs. Stratified Datalog

See [EIK09, Theorem 5] for the following result:

Theorem 6

Stable model semantics and stratified Datalog semantics coincide on Datalog programs with stratified negation.

Thus, stable model semantics extends stratified Datalog semantics to programs in which negation is not stratified.

Table of Contents

Models

```
person(donald).
happy(X) :- person(X), not happy(X).
```

- This program is not in stratified Datalog.
- {`person(donald)`, `happy(donald)`} is a model, but `happy(donald)` is not supported within this model.
- We use the Closed World Assumption (CWA): facts not in the model are false.
- {`person(donald)`} violates the second rule, and therefore is not a model.
- {`person(donald)`, `happy(donald)`, `person(melania)`, `happy(melania)`} is also a model, but only `person(donald)` is supported within this model.

Supported Models

```
person(donald).
man(X) :- person(X), not female(X).
female(X) :- person(X), not man(X).
```

- This program is not in stratified Datalog.
- {`person(donald), man(donald)`} is a model in which `man(donald)` is supported (by means of the middle rule). This will be a necessary (but not a sufficient) property for a model to be stable.
- {`person(donald), female(donald)`} is another stable model.
- {`person(donald), man(donald), female(donald)`} is a model, but not a stable one: neither `man(donald)` nor `female(donald)` is supported within this model.

Definitions

- An atom (or fact) is an expression $R(c_1, \dots, c_n)$ where R is an n-ary relation name and $c_1, \dots, c_n$ are constants.
- A positive literal is a fact; a negative literal is the negation of a fact.
- If B is a set of literals, then B^+ is the set of positive literals in B, and $B^- := \{A \mid \neg A \in B\}$.
- Rules and programs are as in Datalog (but negation need not be stratified).
- A ground instance of a program rule is any variable-free rule that can be obtained by substituting constants for variables in the rule.
- The grounding of a program P, denoted $\mathit{ground}(P)$, is the program consisting of all ground instances of rules in P.
- An interpretation is a set of facts.
- An interpretation M is a model of a program P if it is a model of $\mathit{ground}(P)$. Notice that $\mathit{ground}(P)$ is a variable-free logic program for which models have been previously defined.

Stable Models

Let P be a program, where we assume that all database facts are listed in the program.

Definition 7

The Gelfond-Lifschitz transform (or reduct) of a program P with respect to a set M of facts, denoted P^M, is a negation-free and variable-free program constructed as follows: for every rule $h \leftarrow B$ of $\mathit{ground}(P)$,

- if $B^- \cap M \neq \emptyset$, then remove the entire rule; and
- if $B^- \cap M = \emptyset$, then remove the negative literals from the rule.

A set M of facts is a stable model of P if it is the unique minimal model of P^M.

Supported Models

A model M of a program P is supported if for every fact $h \in M$, there exists a rule $h \leftarrow B$ in $\mathit{ground}(P)$ such that $M \models B$.

It does *not* immediately follow from Definition 7 that every stable model M of P is also a model of P (because $P \neq P^M$ in general). It can be shown that Theorem 3 also holds for programs with variables:

Theorem 8

If M is a stable model of a program P, then M is a model of P that, moreover, is minimal and supported.

Example: No Stable Model

Lat P be the following program:

```
person(donald).
happy(X) :- person(X), not happy(X).
```

Let $M = \{\texttt{person(donald)},\ \texttt{happy(donald)}\}$. Then, the reduct P^M is the following program:

```
person(donald).
~~happy(donald) :- person(donald), not happy(donald).~~
```

The unique minimal model of P^M is $\{\texttt{person(donald)}\}$. Since M is distinct from the unique minimal model of P^M, it follows that M is not a stable model.

Example: Two Stable Models

Let P be the following program:

```
person(donald).
man(X) :- person(X), not female(X).
female(X) :- person(X), not man(X).
```

Let $M_1 = \{$`person(donald), man(donald)`$\}$. Then, the reduct P^{M_1} is the following program:

```
person(donald).
man(donald) :- person(donald), ~~not female(donald)~~.
~~female(donald) :- person(donald), not man(donald).~~
```

The unique minimal model of P^{M_1} is $\{$`person(donald), man(donald)`$\}$. Since M_1 is equal to the unique minimal model of P^{M_1}, it follows that M_1 is a stable model.

Exercise: Show that $M_2 = \{$`person(donald), female(donald)`$\}$ is another stable model, but that $M_1 \cup M_2$ is not a stable model.

Graph 3-Colorability

Assume a finite set of edge-facts, for example:

```
edge(1,2). edge(1,3). edge(2,3).
```

The following program finds all 3-colorings:

```
adjacent(X,Y) :- edge(X,Y).
adjacent(X,Y) :- edge(Y,X).
vertex(X) :- adjacent(X,Y).

green(X) :- vertex(X), not red(X), not blue(X).
blue(X)  :- vertex(X), not red(X), not green(X).
red(X)   :- vertex(X), not blue(X), not green(X).

:- adjacent(X,Y), red(X), red(Y).
:- adjacent(X,Y), blue(X), blue(Y).
:- adjacent(X,Y), green(X), green(Y).
```

Rules with Empty Head=Integrity Constraints

```
:- adjacent(X,Y), red(X), red(Y).
```

is tantamount to:

```
happy(donald) :- adjacent(X,Y), red(X), red(Y),
                 not happy(donald).
```

Assume, toward a contradiction, the existence of a stable model that contains `adjacent(i,j), red(i), red(j)`. Such a model must contain `happy(donald)` in order to satisfy the latter rule. But `happy(donald)` will not be supported within that model, contradicting that the model is stable.

Correctness proof

Claim 1

If the input is a 3-colorable graph, then the program has a stable model.

Proof (sketch).

Let γ be a valid 3-coloring of the input graph with co-domain $\{r, b, g\}$. Let M be a set of facts such that for every vertex i,

- if $\gamma(i) = r$, then M contains `red(i)`;
- if $\gamma(i) = b$, then M contains `blue(i)`; and
- if $\gamma(i) = g$, then M contains `green(i)`.

Assume that M contains `red(i)`. Then M contains neither `blue(i)` nor `green(i)`. Then `red(i)` is supported within M because of the program rule

```
red(X) :- vertex(X), not blue(X), not green(X).
```

□

Correctness proof (other direction)

Claim 2

If the program has a stable model, then the input graph is 3-colorable.

Proof (sketch).

Let M be a stable model. Whenever M contains `vertex(i)`, it must also contain `green(i)`, `blue(i)`, or `red(i)` (because of the three rules "in the middle"). Assume, toward a contradiction, that M contains two such facts, say `green(i)` and `blue(i)`. Then `green(i)` is not supported within M, contradicting that M is stable. □

Data Complexity

$$\text{ASP} \simeq \begin{array}{c}\text{Datalog with (possibly unstratified) negation}\\ + \\ \text{stable model semantics}\end{array}$$

Claim 3

ASP has NP*-hard data complexity.*

Proof.

Deciding whether a graph is 3-colorable is known to be NP-complete. The desired result then follows from Claims 1 and 2. □

Clingo

The database file `graphK3.lp` is:

```
edge(1,2). edge(1,3). edge(2,3).
```

Execution of the command

```
clingo my3coloring.lp graphK3.lp 0
```

yields:

```
Answer: 1
green(1) red(2) blue(3)
```

[. . .]

```
Answer: 6
blue(1) red(2) green(3)
SATISFIABLE

Models       : 6
```

Discussion

- ASP programming is different from database querying: a database query maps a database to a uniquely defined set of tuples (called query answer), whereas an ASP program yields all stable models (also called answer sets).
- Stable model semantics is a conservative extension of stratified Datalog semantics (which you studied in *Bases de données II*). Thus, stratified Datalog programs return unique stable models.

Choice

The following program has $8 = 2^3$ stable models:

```
person(donald). person(melania). person(jeb).
happy(X) :- person(X), not unhappy(X).
unhappy(X) :- person(X), not happy(X).
```

In clingo ASP, choice can be more succinctly expressed by using curly braces $\{\ldots\}$:

```
person(donald). person(melania). person(jeb).
{ happy(X) : person(X) }.
unhappy(X) :- person(X), not happy(X).
```

Cardinality Constraints: At Least Two Happy Persons

The following program has 4 stable models:

```
person(donald). person(melania). person(jeb).
equal(X,X) :- person(X).
happy(X) :- person(X), not unhappy(X).
unhappy(X) :- person(X), not happy(X).
goodModel :- happy(X), happy(Y), not equal(X,Y).
:- not goodModel.
```

In clingo ASP, cardinality constraints can be more succinctly expressed by using $\ell\{\dots\}u$:

```
person(donald). person(melania). person(jeb).
2 { happy(X) : person(X) }.
unhappy(X) :- person(X), not happy(X).
```

Cardinality Rules

```
adjacent(X,Y) :- edge(X,Y).
adjacent(X,Y) :- edge(Y,X).
vertex(X) :- adjacent(X,Y).
1 { red(X); green(X); blue(X) } 1 :- vertex(X).
:-adjacent(X,Y), red(X), red(Y).
:-adjacent(X,Y), blue(X), blue(Y).
:-adjacent(X,Y), green(X), green(Y).
```

Caveat: This is fundamentally different from disjunction in rule heads (see later).

Syntactic Sugar?!

The following two ground programs have the same stable models:

```
vertex(42).
1 { red(42); green(42); blue(42) } 1 :- vertex(42).
```

and

```
vertex(42).
red(42)   :- vertex(42), not green(42), not blue(42).
green(42) :- vertex(42), not red(42), not blue(42).
blue(42)  :- vertex(42), not red(42), not green(42).
```

Data versus program

Variables in ASP range over finite domains that come from the input data. ASP constructs may be more easily understood when programs with variables are viewed as compact representations of variable-free programs.

Cardinality Rules in Clingo (Example)

```
r(1). r(2). r(3). r(4).

%*
The stable models of the following program have the
same r-facts as the program with the single rule:
               1 { select(X) : r(X) }.
*%

select(X) :- r(X), not reject(X).
reject(X) :- r(X), not select(X).
goodModel :- r(X), select(X).
goodModel :- not goodModel.

#show select/1.
```

Cardinality Rules in Clingo (Example)

```
r(1,a). r(2,a).
r(2,b). r(3,b).
r(3,c). r(4,c).
s(a). s(b).

1 { select(X) : r(X,Y) } 1 :- s(Y).

#show select/1.
```

There are two stable models:

```
Answer: 1
select(1) select(3)
Answer: 2
select(2)
SATISFIABLE
```

Data Complexity

- In database theory, we distinguish between a database of facts, and a program that is mostly fact-free. In logic programming, this distinction is usually blurred.
- For every program P, define $eval(P)$ as the following decision problem:

  Given a database $\mathbf{db}$*, does* $\mathbf{db} \cup P$ *have a stable model?*

- Note that if P_1, P_2 are distinct programs, then $eval(P_1)$ and $eval(P_2)$ are different problems. These problems are also different from the following decision problem:

  Given a program P *and a database* $\mathbf{db}$*, does* $\mathbf{db} \cup P$ *have a stable model?*

Data Complexity of (Disjunction-Free) Programs

Claim 4

For every disjunction-free program P, eval(P) is in NP.

Proof (sketch).

We can guess a stable model M of $\mathbf{db} \cup P$, and test in polynomial time whether it is indeed a stable model. The test uses Definition 7 (i.e., test whether M is the minimal model of P^M). This test does not take more than polynomial time, because *ground*(P) is of polynomial size in the length of P.
To see why the latter claim is true, let ℓ be the length of P. The number of variables in P cannot be greater than ℓ. Let n be the number of constants that occur in $\mathbf{db}$ or P. With at most ℓ variables and n constants, we can construct at most n^ℓ distinct valuations. Note that although n^ℓ can be huge, it is nevertheless polynomial, because ℓ is fixed (i.e., P is not part of the input). □

Table of Contents

Definitions

- A rule is an expression

$$h_1 \vee \cdots \vee h_\ell \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m$$

  where $h_1, \ldots, h_\ell, a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_m$ are atoms. Possibly $\ell = 0$.
- We can treat the head of such a rule as a set $H = \{h_1, \ldots, h_\ell\}$.
- An interpretation I is a model of a program P if for every rule $H \leftarrow B$ in P, if $I \models B$, then $H \cap I \neq \emptyset$.
- The Gelfond-Lifschitz transform of a disjunctive program is defined in the same way as for normal programs.
- An interpretation M is a stable model of P if it is <u>a minimal</u> model of P^M.

Example

P

```
a|b|c :- not c.
```

- $P^{\{a\}} = P^{\{b\}} = \{a \vee b \vee c \leftarrow \emptyset\}$. Therefore, since both $\{a\}$ and $\{b\}$ are minimal models of $\{a \vee b \vee c \leftarrow \emptyset\}$, they are stable models of P.
- $P^{\{c\}} = \{\}$. Since $\{c\}$ is not a minimal model of the empty program, it is not a stable model.

Note that clingo uses the symbol `|` (vertical bar) rather than $\vee$. Furthermore, `|` can be replaced with `;` (semicolon).

Faber-Pfeifer-Leone Transform

Same content as slide 18. Replace “program” with “disjunctive program.”

Disjunctive Heads Are a True Extension

P

```
a|b.
b :- a.
a :- b.
```

$\{a, b\}$ is a stable model of P, because $\{a, b\}$ is a minimal model of $P^{\{a,b\}} = P$.

Q

```
b :- not a.
a :- not b.
b :- a.
a :- b.
```

$Q^{\{a,b\}}$

```
b :- a.
a :- b.
```

$\{a, b\}$ is not a stable model of Q, because the unique minimal model of $Q^{\{a,b\}}$ is $\{\}$.

Thus, P and Q are not equivalent.

Caveat: Disjunction ≠ Choice

The program

```
    a ; b. % which can also be written as:   a | b.
 1 {c ; d}.
```

has six models:

```
Answer: 1
a d
Answer: 2
a c
Answer: 3
a c d
Answer: 4
b d
Answer: 5
b c
Answer: 6
b c d
SATISFIABLE
```

Table of Contents

Disjunctive Datalog

- In disjunctive Datalog, the head of a rule can be a disjunction of atoms.
- See also [EGM97, Example 2]. The following negation-free disjunctive Datalog program has a minimal model containing `colored` if and only if the input graph is 3-colorable:

  ```
  adjacent(X,Y) :- edge(X,Y).
  adjacent(X,Y) :- edge(Y,X).
  vertex(X) :- adjacent(X,Y).
  green(X) | blue(X) | red(X) :- vertex(X).
  notColored :- adjacent(X,Y), red(X), red(Y).
  notColored :- adjacent(X,Y), blue(X), blue(Y).
  notColored :- adjacent(X,Y), green(X), green(Y).
  notColored | colored.
  ```

  The fourth rule assigns a color to each vertex. If `notColored` is not derivable from the assignment, a minimal model is obtained by including `colored`.

Disjunctive Rule Heads in clingo ASP

- A program is called disjunctive if some rule heads are disjunctions of two or more atoms.
- The Potassco User Guide says:

  In general, the use of disjunction may increase computational complexity. We thus suggest to use "choice constructs" instead of disjunction, unless the latter is required for complexity reasons.

- It is known that disjunctive programs can solve problems that are hard for $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^{\mathrm{P}}$. If $\mathrm{NP}^{\mathrm{NP}} \neq \mathrm{NP}$, then these problems are not in NP.
- Consequently, the proof of Claim 4 must fail for disjunctive programs. Can you see where it fails?

Choice vs. Disjunctive Rule Heads

- The construct of choice in clingo ASP is strictly less powerful than disjunction in rule heads:
  - choice rules can always be equivalently rewritten in rules with atomic rule heads; but
  - disjunction in rule heads cannot be eliminated from disjunctive programs that solve a $\mathrm{NP}^{\mathrm{NP}}$-hard problem (unless $\mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}$).

A Problem Obviously in $\mathrm{NP}^{\mathrm{coNP}}$: $\exists\forall$3-Coloring

$\exists\forall$3-coloring:

Input: An undirected graph $G = (V, E)$ and a partition of its vertices in two subsets X, A (i.e., $X \cup A = V$ and $X \cap A = \emptyset$).

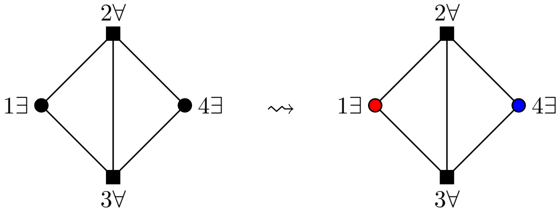
Question: Is there a 3-coloring of the subgraph induced by X that cannot be extended to a 3-coloring of G?[1]

It may be helpful to think of the vertices in X and A as, respectively, eXists-vertices and forAll-vertices.

[1]The subgraph induced by X is the graph whose vertex-set is X and whose edge-set is $\{\{u, v\} \in E \mid u, v \in X\}$.

For example, we claim that the following is a "yes"-instance of ∃∀3-coloring:

```
edge(1,2). edge(1,3). edge(2,4). edge(3,4). edge(2,3).
x(1). x(4). a(2). a(3).
```



To prove our claim, note that the subgraph induced by $X = \{1, 4\}$ has an empty edge-set. If we color $1 \mapsto$ red and $4 \mapsto$ blue, then we must have $2 \not\mapsto$ red, $3 \not\mapsto$ red, $2 \not\mapsto$ blue, $3 \not\mapsto$ blue. However, 2 and 3 cannot both be green, because they are adjacent.

Program P for ∃∀3-coloring

```
adjacent(X,Y) :- edge(X,Y).     adjacent(X,Y) :- edge(Y,X).

red(X) | blue(X) | green(X) :- x(X).
red(Y) | blue(Y) | green(Y) :- a(Y).

w :- adjacent(X,Y), red(X), red(Y).     % To support w (cf. below)
w :- adjacent(X,Y), blue(X), blue(Y).   % and satisfy constraints,
w :- adjacent(X,Y), green(X), green(Y).  % some a-vertex has the
                                         % same color as one of its neighbors.
red(Y)   :- w, a(Y). % Since w is in each stable model (cf. below),
blue(Y)  :- w, a(Y). % each stable model assigns 3 colors to
green(Y) :- w, a(Y). % each a-vertex.

w :- not w. % w is in each stable model, supported by another rule.

:- x(X), x(Y), adjacent(X,Y), red(X), red(Y).
:- x(X), x(Y), adjacent(X,Y), blue(X), blue(Y).
:- x(X), x(Y), adjacent(X,Y), green(X), green(Y).
```

Intuition I

Here is an informal description of the functioning of the program.

- Because of "`w :- not w`," every stable model must contain `w`. Then the Gelfond-Lifschitz transform will contain "`red(Y):-w,a(Y)`," "`blue(Y):-w,a(Y)`," and "`green(Y):-w,a(Y)`" (modulo replacements of `Y` by constants). Consequently, every stable model assigns three colors to each `a`-vertex.
- A model M containing `w` is not supported (and hence not stable) unless there is a rule "`w :-` B" such that $M \models B$ (and therefore "`not w`" $\notin B$). There are three such rules in the program, and the bodies of those rules each require that two adjacent vertices have the same color. At least one of these adjacent vertices must be an `a`-vertex, because the last three rules stipulate that adjacent `x`-vertices have distinct colors.

Intuition II

- A model M is not stable unless it is a minimal model of its Gelfond-Lifschitz transform. The crux now is that if the a-vertices are 3-colorable (given a 3-coloring of the `x`-vertices), then there is a smaller model of the Gelfond-Lifschitz transform: one that does not contain `w` and assigns only one color (instead of three) to each a-vertex.

  Indeed, since no two adjacent vertices have the same color in such a 3-coloring, the body of the rule "`w :- adjacent(X,Y), red(X), red(Y)`" is false, and does not insert `w` (likewise for `blue` and `green`).

Correctness Proof I

Let M be a stable model. We observe the following:

- M must contain `w` (because of the rule `w :- not w`). Thus P^M consists of all rules of $ground(P)$ except `w :- not w`. Since M is a minimal model of P^M, we have that M contains <u>each of</u> `red(Y)`, `blue(Y)`, and `green(Y)`, for every a-vertex `Y`.
- It is also clear that M contains <u>at least one of</u> `red(X)`, `blue(X)`, or `green(X)`, for every `x`-vertex `X`. Assume toward a contradiction that M contains both `red(X)` and `blue(X)` for some `x`-vertex `X`. It can be seen that $M \setminus \{$`blue(X)`$\}$ is also a model, contradicting that M is minimal. We conclude by contradiction (and symmetry) that M contains <u>exactly one of</u> `red(X)`, `blue(X)`, or `green(X)`, for every `x`-vertex `X`.

Correctness Proof II

We now show:

P has a stable model $\iff$ the answer to $\exists\forall$3-coloring is "yes"

$\boxed{\implies}$ Let M be a stable model of P. Let γ be a coloring of the `x`-vertices such that $\gamma(X)$ equals r, b, or g depending on whether M contains, respectively, `red(X)`, `blue(X)`, or `green(X)`, for every `x`-vertex X. Assume toward a contradiction that γ can be extended to a valid 3-coloring of G, denoted γ^+. Let I be the interpretation that contains `red(V)`, `blue(V)`, or `green(V)` depending on whether $\gamma^+(V)$ equals, respectively, r, b, or g, for every vertex V. We let I not contain `w`. It can now be seen that I is a model of P^M such that $I \subsetneq M$, contradicting that M is a stable model of P. We conclude by contradiction that γ cannot be extended to a valid 3-coloring of G.

$\boxed{\impliedby}$ Let γ be a valid 3-coloring of the subgraph induced by X that cannot be extended to a 3-coloring of G. Let I be the interpretation that contains exactly one of `red(X)`, `blue(X)`, or `green(X)` depending on

Correctness Proof III

whether $\gamma(X)$ equals, respectively, r, b, or g, for every `x`-vertex X. Furthermore, I contains `w` and contains each of `red(Y)`, `blue(Y)`, and `green(Y)` for every a-vertex Y. Notice that P^I consists of all rules of *ground*(P) except `w :- not w`, and I is a model of P^I. We show that I is a minimal model of P^I, which implies that I is a stable model of P. Assume toward a contradiction that there is a model M of P^I such that $M \subsetneq I$. It can be seen that it must be the case that M does not contain `w` and that M contains exactly one of `red(Y)`, `blue(Y)`, or `green(Y)` for every a-vertex Y. Let γ^+ be the extension of γ such that $\gamma^+(Y)$ equals r, b, or g depending on whether I contains, respectively, `red(Y)`, `blue(Y)`, or `green(Y)`, for every a-vertex Y. Since I is a model of P^M not containing `w`, it follows that I falsifies the body of each rule with head `w`. Therefore, γ^+ is a valid 3-coloring of G that extends γ, which contradicts our initial hypothesis that no such extension exists. $\square$

Pitfall

Since $\exists\forall$3-Coloring is $\mathrm{NP}^{\mathrm{NP}}$-hard, it cannot be solved by a disjunction-free program (unless $\mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}$).

Therefore, in our program for $\exists\forall$3-Coloring, it must be wrong to write

```
1 { red(X) ; blue(X) ; green(X) } :- x(X).
1 { red(Y) ; blue(Y) ; green(Y) } :- a(Y).
```

instead of

```
red(X) | blue(X) | green(X) :- x(X).
red(Y) | blue(Y) | green(Y) :- a(Y).
```

P_{wrong}

Let P_{wrong} be the program obtained from P by replacing the rule

```
red(Y) | blue(Y) | green(Y) :- a(Y).
```

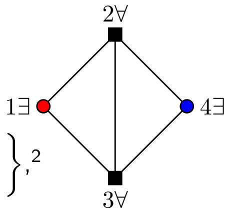
with the following three rules:

```
red(Y)   :- a(Y), not blue(Y), not green(Y).
blue(Y)  :- a(Y), not red(Y), not green(Y).
green(Y) :- a(Y), not red(Y), not blue(Y).
```

P_{wrong} does not solve ∃∀3-Coloring!!!

What Goes Wrong in P_{wrong} (Compared to P)? I

Take the following “yes”-instance of ∃∀3-Coloring:



Let $M = \left\{ \begin{array}{r} \texttt{w}, \texttt{red}(1), \texttt{blue}(4), \texttt{red}(2), \texttt{blue}(2), \texttt{green}(2), \\ \texttt{red}(3), \texttt{blue}(3), \texttt{green}(3) \end{array} \right\}^2,$

which is a stable model of our correct program P.

Recall how $\boxed{P^M_{wrong}}$ is obtained:

```
~~red(2)   :- a(2), not blue(2), not green(2).~~
                        ⋮
~~green(3) :- a(3), not red(3), not blue(3).~~
```

Thus, P^M_{wrong} does not require that vertices 2 and 3 be colored. It can then be checked that $I = \{\texttt{red}(1), \texttt{blue}(4)\}$ is a smaller model of P^M_{wrong}, and therefore M is not a stable model of P_{wrong}.

What Goes Wrong in P_{wrong} (Compared to P)? II

We have that $\boxed{P^M}$ properly extends P^M_{wrong} with the following rules:

```
red(2) | blue(2) | green(2) :- a(2).
red(3) | blue(3) | green(3) :- a(3).
```

Thus, P^M does require that vertices 2 and 3 be colored.

[2] For the sake of simplicity, we have omitted from M and I all `edge`-facts, `x`-facts, `a`-facts, and `adjacent`-facts.

Exercise

Let $P_{correct?}$ be the program obtained from P by replacing the rule

```
red(X) | blue(X) | green(X) :- x(X).
```

with the following three rules:

```
red(X)   :- x(X), not blue(X), not green(X).
blue(X)  :- x(X), not red(X), not green(X).
green(X) :- x(X), not red(X), not blue(X).
```

Does $P_{correct?}$ solve $\exists\forall$3-Coloring???

Exercise: Complement of 3-Colorability

Let $P_{not\_3\_colorable}$ be the following program:

```
adjacent(X,Y) :- edge(X,Y).     adjacent(X,Y) :- edge(Y,X).
vertex(X) :- adjacent(X,Y).

red(Y) | blue(Y) | green(Y) :- vertex(Y).

w :- adjacent(X,Y), red(X), red(Y).
w :- adjacent(X,Y), blue(X), blue(Y).
w :- adjacent(X,Y), green(X), green(Y).

red(Y)   :- w, vertex(Y).
blue(Y)  :- w, vertex(Y).
green(Y) :- w, vertex(Y).

w :- not w.
```

Prove that this program solves a coNP-hard problem:

Claim 5

For the program $P_{not\_3\_colorable}$,

- *if the input graph is not 3-colorable, then there is exactly one stable model;*
- *if the input graph is 3-colorable, then there is no stable model.*

Table of Contents

Aggregate Literals in ASP I

See [AF18].

- A ground term is a variable-free term, for example, 7 and $3 + 3$.
- A ground aggregate element (gag) is an expression $t_1, \ldots, t_m : B$ where $m \geq 1$, each t_i is a ground term, and B is a set of literals.
- If *fun* is an aggregation function and θ a comparison operator $(=, <, \leq, >, \geq, \neq)$, then an aggregate literal (or simply aggregate) has the form
$$fun(\vec{t_1} : B_1; \ldots; \vec{t_n} : B_n) \ \theta \ t \tag{3}$$
where each $\vec{t_i} : B_i$ is a gag, and t is a ground term.
- Let I be an interpretation. Let $T_I = \{\vec{t_i} \mid 1 \leq i \leq n \text{ and } I \models B_i\}$. Let V_I be the multiset of all first elements of sequences in T_I. Then I is said to satisfy (3) if $fun(V_I) \ \theta \ t$ holds true.

Aggregate Literals in ASP II

Example 9

```
#sum {12,1:a; 12,2:b; 12,2:c} = 24
```

The three gags involved are `12,1:a`, `12,2:b`, and `12,2:c`.
For $I = \{a, b, c\}$, we have $T_I = \{\langle 12,1\rangle, \langle 12,2\rangle\}$[a], and therefore $V_I = \{\{12, 12\}\}$. Since $12 + 12 = 24$, I is a satisfying interpretation.

[a] Duplicates are removed! Angular brackets $\langle\rangle$ added for readability.

Example 10

```
#sum {12:a; 12:b} = 24
```

The two gags involved are `12:a` and `12:b`.
For $I = \{a, b\}$, we have $T_I = \{12\}$, and therefore $V_I = \{\{12\}\}$. Clearly, I is a falsifying interpretation.

Caveat

- clingo ASP allows aggregate literals in rule heads; and
- clingo ASP allows negative literals in rule heads (see slide 17).

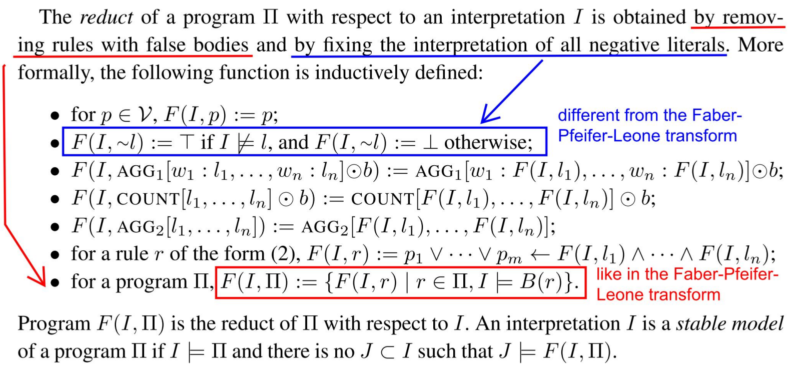
We will only consider rules with heads of the form $h_1 \vee \cdots \vee h_\ell$ where each h_i is an atom. Such a rule is called normal if $\ell = 1$.

Stable Models of Programs with Aggregate Literals I

- How can stable model semantics be generalized to aggregate literals? In particular, what is the "right" extension of the Gelfond-Lifschitz transform?
- These are nontrivial questions. Consider, for example:

  ```
  a :- #sum {1:not a} < 1.
  ```

  clingo returns two answer sets: $\{\}$ and $\{a\}$. In what respect can $\{a\}$ then be a minimal stable model?
- Note:
  - the empty interpretation $\{\}$ falsifies `#sum {1:not a} < 1`, and therefore satisfies the above rule;
  - the interpretation $\{a\}$ necessarily satisfies every rule with head `a`.



The *reduct* of a program Π with respect to an interpretation I is obtained by removing rules with false bodies and by fixing the interpretation of all negative literals. More formally, the following function is inductively defined:

- for $p \in \mathcal{V}$, $F(I, p) := p$;
- $\boxed{F(I, \sim l) := \top \text{ if } I \not\models l, \text{ and } F(I, \sim l) := \bot \text{ otherwise;}}$ different from the Faber-Pfeifer-Leone transform
- $F(I, \text{AGG}_1[w_1 : l_1, \ldots, w_n : l_n] \odot b) := \text{AGG}_1[w_1 : F(I, l_1), \ldots, w_n : F(I, l_n)] \odot b$;
- $F(I, \text{COUNT}[l_1, \ldots, l_n] \odot b) := \text{COUNT}[F(I, l_1), \ldots, F(I, l_n)] \odot b$;
- $F(I, \text{AGG}_2[l_1, \ldots, l_n]) := \text{AGG}_2[F(I, l_1), \ldots, F(I, l_n)]$;
- for a rule r of the form (2), $F(I, r) := p_1 \vee \cdots \vee p_m \leftarrow F(I, l_1) \wedge \cdots \wedge F(I, l_n)$;
- for a program Π, $\boxed{F(I, \Pi) := \{F(I, r) \mid r \in \Pi, I \models B(r)\}.}$ like in the Faber-Pfeifer-Leone transform

Program $F(I, \Pi)$ is the reduct of Π with respect to I. An interpretation I is a *stable model* of a program Π if $I \models \Pi$ and there is no $J \subset I$ such that $J \models F(I, \Pi)$.

Taken from [AFG15]. Let Π be the program whose only rule is

$$a \leftarrow \text{SUM}[1 : \sim a] < 1$$

- $F(\{\}, \Pi) = \{\}$ (because $1 \not< 1$)
- $F(\{a\}, \Pi) = \{a \leftarrow \text{SUM}[1 : \bot] < 1\} \equiv \{a \leftarrow \top\}$ (the empty sum is 0).

Sum

	Program	clingo answers
P_1	`c :- #sum {12,v:not a; 12,w:not b} = 24.`	$\{c\}$
P_2	`c :- #sum {12,v:not a; 12,w:not b} = 25.`	$\{\}$
P_3	`a :- #sum {12,v:not a; 12,w:not b} = 24.`	UNSATISFIABLE
P_4	`a :- #sum {12,v:not a; 12,w:not b} = 25.`	$\{\}$

The Potassco User Guide says:

> *The weight refers to the first element of a term tuple. [...] As indicated by the curly braces, the elements within aggregates are treated as members of a set. Hence, duplicates are not accounted for twice.*

Cardinality Rules

Execute

```
gringo myprogram.lp -t
```

to see the grounding of the program in `myprogram.lp`. The program

```
1 {a;b}.
b :- a.
a :- b.
```

is grounded as:

```
a:-b.
b:-a.
1<=#count{0,a:a;0,b:b}.
```

Disjunctive Heads vs. Cardinality Rules

See also Slide 57.

P

```
a|b.
```

$\{a\}$ and $\{b\}$ are the only stable models of P.

Q

```
1 {a;b}.
```

Output of gringo:

```
1<=#count{0,a:a;0,b:b}.
```

Well, on slide 79, we excluded from consideration rules whose head is an aggregate literal...

clingo returns the models $\{a\}$, $\{b\}$, and $\{a, b\}$.

Thus, P and Q are not equivalent.

Note: clingo returns the model $\{a, b\}$ which does not look minimal.

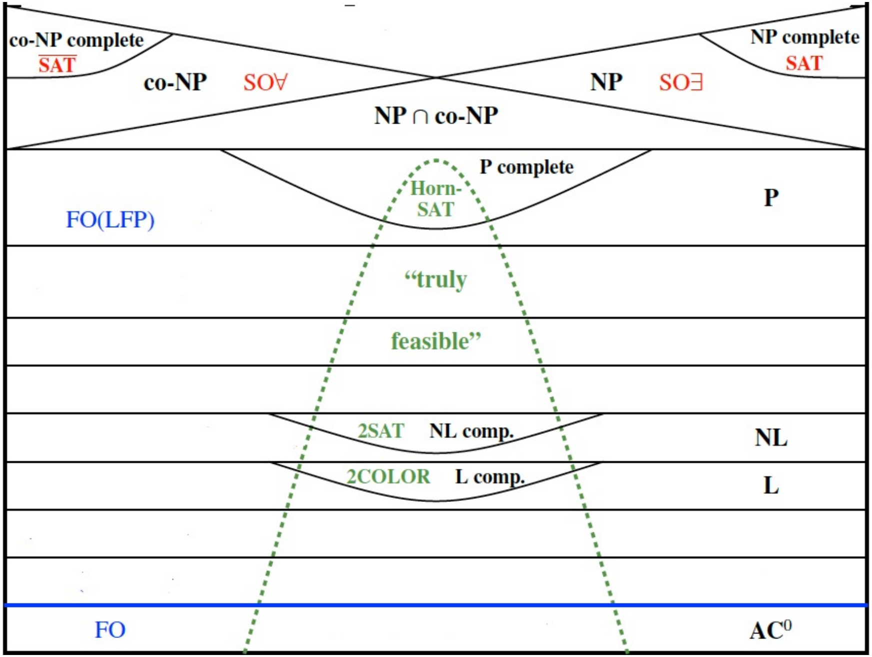
Table of Contents

Recall of Complexity Classes I

Class		Mnemonic	Second-order quantifiers
FO	relational database problems solvable in first-order logic		
L	deterministic logarithmic space	[Rei08] reachability in undirected graphs	
NL	nondeterministic logarithmic space	reachability in directed graphs	
P	deterministic polynomial time		
NP	nondeterministic polynomial time	there is a succinct witness of “yes”	$\exists$
coNP		there is a succinct witness of “no”	$\forall$
$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{coNP}}$	NP with coNP-oracle		$\exists\forall$
$\Pi_2^{\mathrm{P}} = \mathrm{coNP}^{\mathrm{NP}}$	coNP with NP-oracle		$\forall\exists$

Recall of Complexity Classes II

- We know $\mathrm{FO} \subsetneq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$. It is an open conjecture that all these classes are distinct.
- The complement of a decision problem p, denoted $\overline{p}$, is the decision problem resulting from reversing the yes and no answers. Thus, $\mathrm{coNP} = \{\overline{p} \mid p \in \mathrm{NP}\}$. coNP is not the complement of NP.
- It is an open conjecture that $\mathrm{coNP} \neq \mathrm{NP}$. Obviously, if $\mathrm{P} = \mathrm{NP}$, then $\mathrm{coNP} = \mathrm{NP}$.
- $\mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}^{\mathrm{coNP}}$, because a coNP-oracle can be used as an NP-oracle (and vice versa). For example, instead of asking *"Is this graph 3-colorable?"* to an NP-oracle, we can ask *"Is this graph not 3-colorable?"* to a coNP-oracle and negate the answer.
- Formally, P is a class of decision problems. FP is the class of function problems solvable in deterministic polynomial time. For example, *"Determine the number of edges of a given graph"* is not a decision problem, and is obviously in FP.


co-NP complete
$\overline{\text{SAT}}$
co-NP
SO∀
NP
SO∃
NP complete
SAT
NP ∩ co-NP
P complete
Horn-SAT
FO(LFP)
P
"truly
feasible"
2SAT
NL comp.
NL
2COLOR
L comp.
L
FO
AC^0

Digression on the Class NP

Definition 11 (Nondeterministic Polynomial Time)

NP contains all (and only) those decision problems that can be solved by a guess-and-check program such that

1. the guess (a.k.a. certificate or witness) is of polynomial size; and
2. can be checked in polynomial time.

- Since $2 \implies 1$, the item 1 is actually redundant, and can hence be omitted. Indeed, a polynomial-time algorithm could not even read (and hence not check) an exponential-size guess.
- $\mathrm{P} \subseteq \mathrm{NP}$, because we can guess "yes" and check this guess in polynomial time (by solving the problem!).
- NP-hard problems cannot be solved in polynomial time unless $\mathrm{P} = \mathrm{NP}$.
- Observation (not a theorem): Most (all?) natural problems in NP are either in P or NP-complete. It is today not known whether this observation holds for graph isomorphism and factoring ($=$ Does n have a nontrivial factor smaller than k?).
- ASP can solve problems in Σ_2^P, which includes NP.

Logics Capturing Complexity Classes

Theorem 12 (Fagin)

NP=*Existential Second-Order logic*

Extract from [Lib04]:

To test if a graph is 3-colorable, we have to check that there exist three disjoint sets A, B, C covering the nodes of the graph such that for every edge $(a, b) \in E$, the nodes a and b cannot belong to the same set. The sentence below does precisely that:

$$\exists A \exists B \exists C \left(\begin{array}{l} \forall x \left[\begin{array}{r} (A(x) \wedge \neg B(x) \wedge \neg C(x)) \\ \vee\ (\neg A(x) \wedge B(x) \wedge \neg C(x)) \\ \vee\ (\neg A(x) \wedge \neg B(x) \wedge C(x)) \end{array} \right] \\ \wedge \\ \forall x, y\ E(x, y) \rightarrow \neg \left[\begin{array}{r} (A(x) \wedge A(y)) \\ \vee\ (B(x) \wedge B(y)) \\ \vee\ (C(x) \wedge C(y)) \end{array} \right] \end{array} \right) \tag{1.2}$$

Conjecture 1 (Gurevich)

There is no logic that captures P *over the class of all finite structures.*

Exercise: Presence of a Directed Path From a to b.

Is there a directed path from a to b ($a \neq b$) in a directed simple graph with edge-set E?

$$\exists T/2 \left(\begin{array}{l} \forall u \forall v\, (T(u,v) \rightarrow E(u,v)) \wedge \\ \exists v\, (T(a,v)) \wedge \neg \exists u\, (T(u,a)) \\ \exists u\, (T(u,b)) \wedge \neg \exists v\, (T(b,v)) \\ \forall u \forall v\, ((T(u,v) \wedge v \neq b) \rightarrow \exists w\, (T(v,w))) \wedge \\ \forall v \forall w\, ((T(v,w) \wedge v \neq a) \rightarrow \exists u\, (T(u,v))) \wedge \\ \neg \exists u \exists v \exists w\, (T(u,v) \wedge T(u,w) \wedge v \neq w) \wedge \\ \neg \exists u \exists v \exists w\, (T(u,w) \wedge T(v,w) \wedge u \neq v) \end{array} \right)$$

Informally, we guess a subset T of edges and check in FO that T constitutes a path from a to b: a is a source with positive outdegree; b is a sink with positive indegree; b is the only sink; a is the only source; the outdegree of any vertex is at most 1; and the indegree of any vertex is at most 1 (which excludes cycles).

Encoding in ASP

```
% t is a subset of e.
t(U,V) :- e(U,V), not out(U,V).
out(U,V) :- e(U,V), not t(U,V).
hasOutgoing(U) :- t(U,V).
hasIncoming(V) :- t(U,V).
% t must allow starting from a, and ending in b.
:- not hasOutgoing(a).    :- not hasIncoming(b).
% in t, a and b must be source and sink, respectively.
:- t(U,a).                :- t(b,V).
% b is the only sink, and a the only source.
:- t(U,V), V!=b, not hasOutgoing(V).
:- t(U,V), U!=a, not hasIncoming(U).
% in- and outdegrees can be at most 1.
:- t(U,V), t(U,W), V!=W.
:- t(U,W), t(V,W), U!=V.
```

Exercise: Absence of a Directed Path From a to b.

Is there no directed path from a to b ($a \neq b$)?

$$\exists T/2 \left(\begin{array}{l} \forall u \forall v \,(E(u,v) \rightarrow T(u,v)) \wedge \\ \forall u \forall v \forall w \,(T(u,v) \wedge T(v,w) \rightarrow T(u,w)) \wedge \\ \neg T(a,b) \end{array} \right)$$

Informally, we guess a subset T that includes E and is transitive, but does not contain $T(a,b)$.

Since the problem “Is there no directed path from a to b?” is in $\exists$SO, it is in NP. It follows that the complementary problem “Is there a directed path from a to b?” is in coNP.

Encoding in ASP

```
% The following program finds a stable model if and
% only if there is no directed path from a to b.
t(U,V) :- e(U,V).
t(U,V) :- t(U,W), t(W,V).
:- t(a,b).
```

Encoding in Linear Datalog (which is in NL)

- Our exercise showed that reachability is in NP ∩ coNP.
- Of course, you already knew from *Bases de données II* that reachability can be solved in linear Datalog, and hence is in NL. Thus, ASP programs for reachability that go beyond Datalog overkill the problem.

```
t(U,V) :- e(U,V).
t(U,V) :- t(U,X), e(X,V).
yes()  :- t(a,b).
no()   :- not yes().
```

References I

Mario Alviano and Wolfgang Faber.
Aggregates in answer set programming.
KI, 32(2-3):119–124, 2018.

Mario Alviano, Wolfgang Faber, and Martin Gebser.
Rewriting recursive aggregates in answer set programming: back to monotonicity.
TPLP, 15(4-5):559–573, 2015.

Thomas Eiter, Georg Gottlob, and Heikki Mannila.
Disjunctive datalog.
ACM Trans. Database Syst., 22(3):364–418, 1997.

Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner.
Answer set programming: A primer.
In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutiérrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer, 2009.

Wolfgang Faber, Gerald Pfeifer, and Nicola Leone.
Semantics and complexity of recursive aggregates in answer set programming.
Artif. Intell., 175(1):278–298, 2011.

References II

Leonid Libkin.
Elements of Finite Model Theory.
Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

Omer Reingold.
Undirected connectivity in log-space.
J. ACM, 55(4):17:1–17:24, 2008.

Miroslaw Truszczynski.
An introduction to the stable and well-founded semantics of logic programs.
In Michael Kifer and Yanhong Annie Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*, pages 121–177. ACM / Morgan & Claypool, 2018.