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

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):

ALGORITHM = LOGIC + 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

$$man(dilbert).$$

$$person(X) \leftarrow man(X).$$

Query $?-\ person(X)$

Answer $X = dilbert$

- 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

$$reverse([X|Y], Z) \leftarrow append(U, [X], Z), reverse(Y, U). \quad (1)$$

vs

$$reverse([X|Y], Z) \leftarrow reverse(Y, U), append(U, [X], Z). \quad (2)$$

Query: $?- \; reverse([a|X], [b, c, d, b])$

- (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

$$a \leftarrow b_1, \ldots, b_m \ , \tag{1}$$

where a, b_1, ..., 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 $(\forall) b_1 \wedge \cdots \wedge b_m \rightarrow a$.
- 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, ..., b_m are.
- We can reduce P to its *grounding* $grnd(P) = \bigcup_{r \in P} grnd(r)$, where $grnd(r)$ 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:

$$p(X,Y,Z) \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$h(X,Z') \leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r).$$
$$p(0,0,b). \qquad h(0,0). \qquad t(a,b,r).$$

- Constant symbols: 0, a, b, r.
- $HU(P)$: $\{0,a,b,r\}$
- $HB(P)$: $\{\, p(0,0,0), p(0,0,a), \ldots, p(r,r,r),$
  $h(0,0), h(0,a), \ldots, h(r,r,r),$
  $t(0,0,0), t(0,0,a), \ldots, t(r,r,r)\}$
- Some Herbrand interpretations:

$$I_1 = \emptyset; \quad I_2 = HB(P); \quad I_3 = \{h(0,0), t(a,b,r), p(0,0,b)\}.$$

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

Fixpoint Characterization

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

Definition 10 (T_P Operator)

Let $T_P \colon 2^{HB(P)} \to 2^{HB(P)}$ be defined as

$$T_P(I) = \left\{ a \;\middle|\; \begin{array}{l} \text{there exists some } a \leftarrow b_1, \ldots, b_m \\ \text{in } grnd(P) \text{ such that } \{b_1, \ldots, b_m\} \subseteq I \end{array} \right\}.$$

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, $lfp(T_P)$, and the sequence $\langle T_P^i \rangle$, $i \geq 0$, converges to $lfp(T_P)$.

Proof: Use the fixed-point theorems of Knaster-Tarski and Kleene.

Fixpoint Characterization /2

Example 12

- For $P_3 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have

$$T^0_{P_3} = \emptyset,\ T^1_{P_3} = \{c\},\ T^2_{P_3} = \{c, b\},\ T^3_{P_3} = \{c, b, a\},\ T^4_{P_3} = T^3_{P_3}.$$

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

- For $P_4 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have

$$T^0_{P_4} = \emptyset, \quad T^1_{P_4} = \{c\}, \quad T^2_{P_4} = T^1_{P_4}.$$

  Hence $lfp(T_{P_4}) = \{c\}$.

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

$$a \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n \qquad (n, m \geq 0) \tag{2}$$

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

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)

$$man(dilbert).$$

$$single(X) \leftarrow man(X), not\ husband(X).$$

$$husband(X) \leftarrow fail.$$ % fail = "false" in Prolog

Query:

$$?-single(X).$$

Answer:

$$X = dilbert\ .$$

Example 15 (cont'd)

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

$$man(dilbert).$$
$$single(X) \leftarrow man(X), not\ husband(X).$$
$$husband(X) \leftarrow man(X), not\ single(X).$$

Query: $?-single(X).$

Result in Prolog ????

Problem: not a single intuitive model!

Two intuitive models:

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

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: $man(dilbert)$ is true,
    $single(dilbert)$, $husband(dilbert)$ are unknown

  - *Answer Set (alias Stable Model) Semantics* by Gelfond and Lifschitz [1988,1991].

    Alternative models: $M_1 = \{man(dilbert), single(dilbert)\}$,
    $M_2 = \{man(dilbert), husband(dilbert)\}$.

- 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

$$a_1 \vee \cdots \vee a_l \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$$

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

- Notation: $Body(r) = \{b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n\}$ and $Head(r) = \{a_1, \ldots, a_k\}$
- *In practice, function symbols are restricted or disregarded*;
- Generalizes normal programs (single literal head, i.e. $l = 1$)
- Notation: 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. $not\ a$) if $a \in M$ ($a \notin M$) for a ground atom a;
- a ground (variable-free) rule r,

$$a_1 \vee \cdots \vee a_k \leftarrow\ b_1, \ldots, b_m, not\ \ c_1, \ldots, not\ \ c_n$$

  if either (i) M does not satisfy some literal b_i or $not\ c_j$ in $Body(r)$, or (ii) M satisfies some $a_i \in Head(r)$;
- 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 $a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$ 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

Answer Sets /3

Definition 20 (Answer Set)

M is an *answer set* of a program P, if M is a minimal model of P^M.
$AS(P)$ 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 $AS(P)$ 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)

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

Candidate interpretations:

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

M_1 and M_2 are answer sets.
We can equivalently replace (r_1), (r_2) here by

$$husband(dilbert) \vee single(dilbert) \leftarrow man(dilbert).$$

(more natural reading).

Constraints

Programs with not might lack answer sets.

Example 22

$$P = \{\ p \leftarrow not\ \ p.\ \}$$

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

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

$$shaves(X,Y) \leftarrow barber(X), man(Y),\ not\ shaves(Y,Y).$$
$$man(bertrand).\ \ man(dilbert).$$
$$barber(bertrand).$$

- Adding $p \leftarrow q_1, \ldots, q_m, not\ \ r_1, \ldots, not\ \ r_n, not\ \ p.$

  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: $\leftarrow q_1, \ldots, q_m, not\ \ r_1, \ldots, not\ \ r_n.$ (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. $P = \{a \leftarrow not\ b\}$: $M = \{a\}$ $\quad$ $P \cup \{b \leftarrow\}$: $M = \{b\}$

Proposition 2 (Supportedness)

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

But: stable $\neq$ minimal + supported!

E.g. $P = \{a \leftarrow a;\ a \leftarrow not\ a\}$

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).

$$\text{E.g.} \quad P = \left\{ \begin{array}{ll} b \leftarrow not\ c; & a \leftarrow b; \\ c \leftarrow not\ b; & a \leftarrow not\ a \end{array} \right\}$$

Proposition 4 (Generalization of Stratified Semantics)

If in a normal P "not" is layered ("P is stratified"), then P has a unique answer set $M^\star(P)$, which coincides with the perfect model.

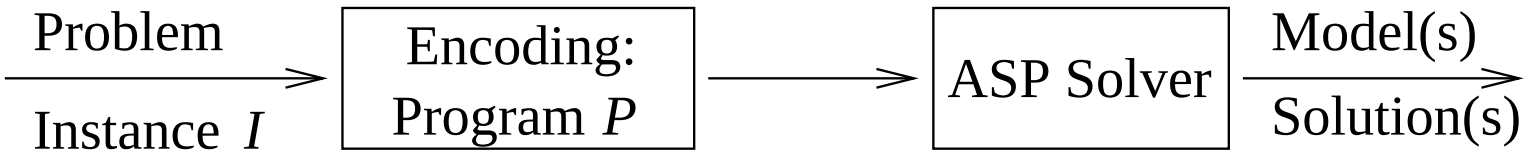
$M^\star(P)$ is characterized by an iterative fixpoint construction

$$\text{E.g.} \quad P = \left\{ \begin{array}{l} man(dilbert). \\ husband(X) \leftarrow man(X),\ married(X). \\ single(X) \leftarrow man(X), not\ husband(X). \end{array} \right\}$$

ASP Paradigm

General idea: answer sets are solutions!

Reduce solving a problem instance I to computing answer sets of an LP



- 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}

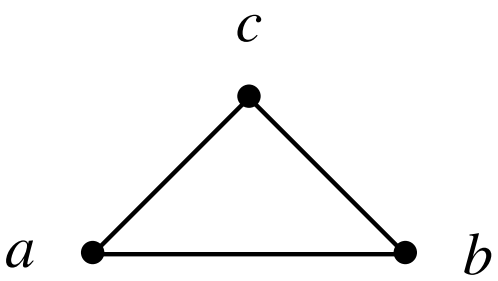
$$g(X) \vee r(X) \vee b(X) \leftarrow node(X) \quad \Big\} \textbf{ Guess}$$

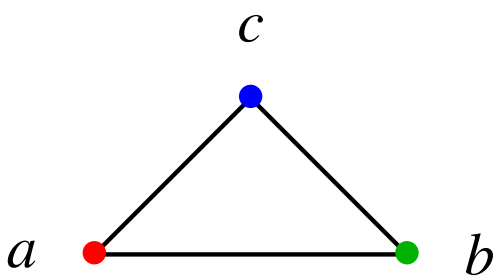
$$\left.\begin{array}{l} \leftarrow b(X), b(Y), edge(X,Y) \\ \leftarrow r(X), r(Y), edge(X,Y) \\ \leftarrow g(X), g(Y), edge(X,Y) \end{array}\right\} \textbf{ Check}$$

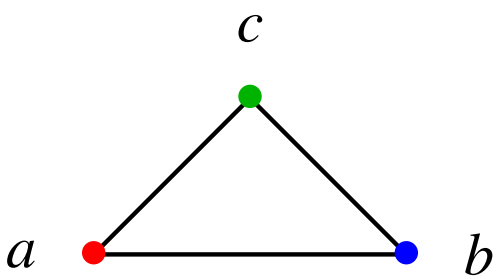
Data P_D: Graph $G = (V, E)$

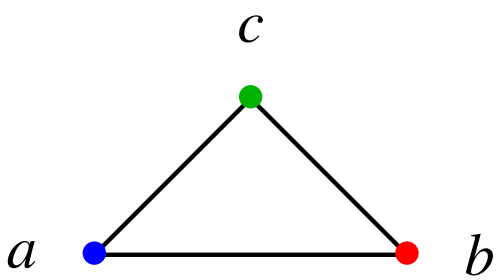
$$P_D = \{node(v) \mid v \in V\} \cup \{edge(v,w) \mid (v,w) \in E\}.$$

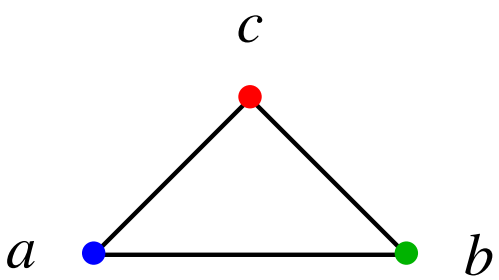
Correspondence 3-colorings $\rightleftharpoons$ models:
$v \in V$ is colored with $c \in \{r, g, b\}$ iff $c(v)$ is in the model of $P_{PS} \cup P_D$.

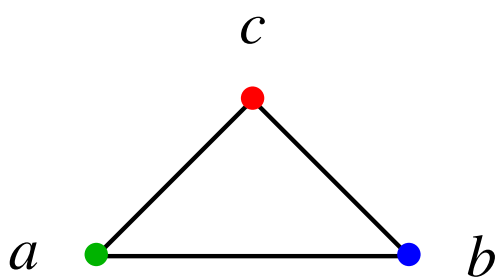


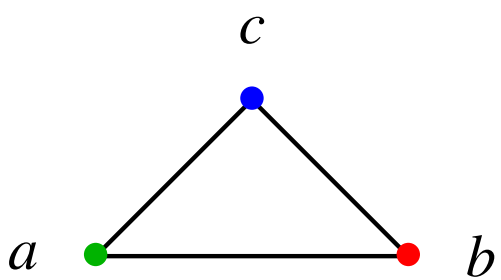












Example: 3-Coloring (ctd)

$$P_D = \{node(a), node(b),\\ node(c), edge(a,b),\\ edge(b,c), edge(a,c)\}$$

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.

$$inPath(X,Y) \vee outPath(X,Y) \leftarrow edge(X,Y). \quad \}\ \textbf{Guess}$$

$$\left.\begin{array}{l} \leftarrow inPath(X,Y),\ inPath(X,Y_1),\ Y \neq Y_1. \\ \leftarrow inPath(X,Y),\ inPath(X_1,Y),\ X \neq X_1. \\ \leftarrow node(X),\ not\ reached(X). \end{array}\right\}\ \textbf{Check}$$

$$\left.\begin{array}{l} reached(X) \leftarrow start(X). \\ reached(X) \leftarrow reached(Y),\ inPath(Y,X). \end{array}\right\}\ \textbf{Auxiliary Def.}$$

Example: Hamiltonian Path/Cycle

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

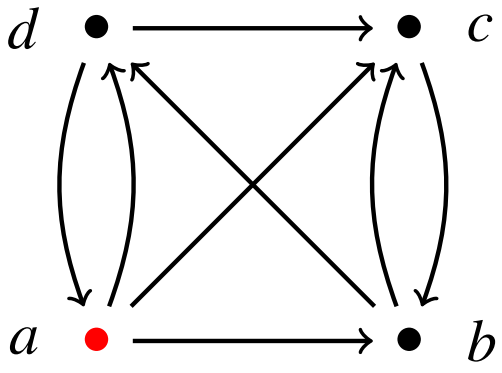
Problem: Find a path/cycle beginning at the starting node which contains all nodes of the graph.

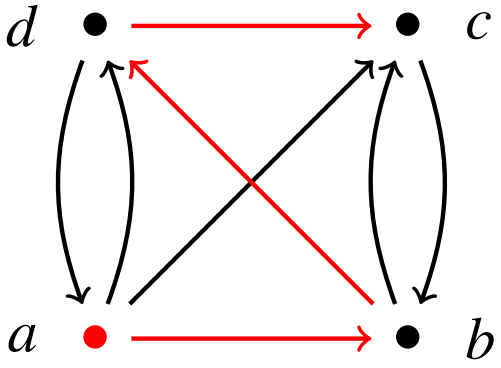
$$inPath(X,Y) \vee outPath(X,Y) \leftarrow edge(X,Y). \quad \}\ \textbf{Guess}$$

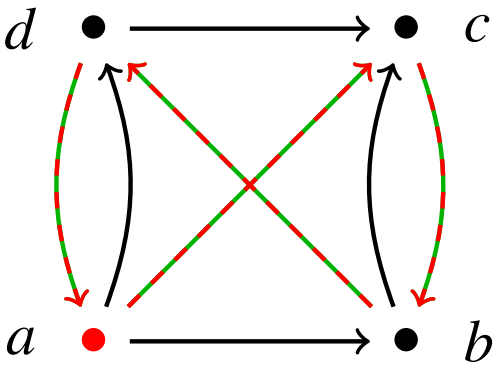
$$\left.\begin{array}{l} \leftarrow inPath(X,Y),\ inPath(X,Y_1),\ Y \neq Y_1. \\ \leftarrow inPath(X,Y),\ inPath(X_1,Y),\ X \neq X_1. \\ \leftarrow node(X),\ not\ reached(X). \\ \leftarrow not\ start\_reached. \end{array}\right\}\ \textbf{Check}$$

$$\left.\begin{array}{l} reached(X) \leftarrow start(X). \\ reached(X) \leftarrow reached(Y),\ inPath(Y,X). \\ start\_reached \leftarrow start(Y), inPath(X,Y). \end{array}\right\}\ \textbf{Auxiliary Def.}$$

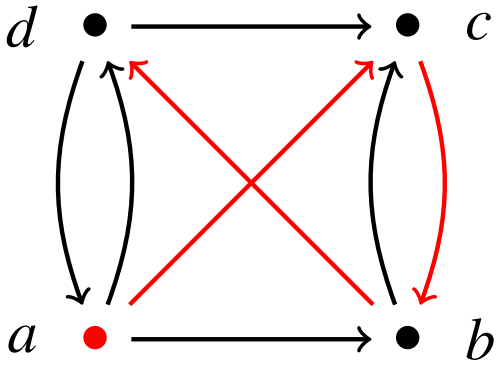
Example: Hamiltonian Path/Cycle (ctd)

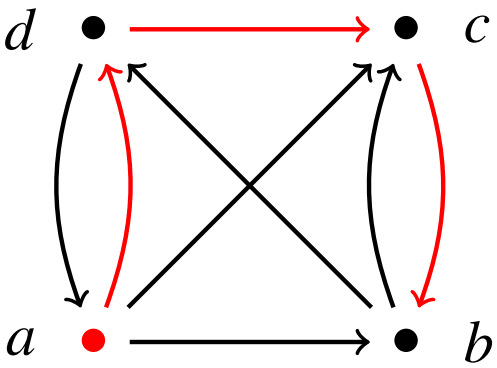






$$
\begin{aligned}
P_D = \{ & node(a), node(b), \\
& node(c), node(d), \\
& edge(a,b), edge(a,c) \\
& edge(c,b), edge(b,c) \\
& edge(b,d), edge(d,c) \\
& edge(d,a), edge(a,d) \\
& start(a) \}
\end{aligned}
$$





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

$$inTree(X,Y) \vee outTree(X,Y) \leftarrow edge(X,Y). \quad \}\ \textbf{Guess}$$

$$\left.\begin{array}{l} \leftarrow inTree(X,Y),\ start(Y). \\ \leftarrow inTree(X,Y),\ inTree(X_1,Y),\ X \neq X_1. \\ \leftarrow node(X),\ not\ reached(X). \end{array}\right\}\ \textbf{Check}$$

$$\left.\begin{array}{l} reached(X) \leftarrow start(X). \\ reached(X) \leftarrow reached(Y),\ inTree(Y,X). \end{array}\right\}\ \textbf{Auxiliary Def.}$$

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;*
- Σ_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).

$$\left.\begin{array}{l} b(X) \vee r(X) \vee g(X) \leftarrow node(X). \end{array}\right\} \textbf{Guess}$$

$$\left.\begin{array}{l} non\_col \leftarrow r(X), r(Y), edge(X,Y). \\ non\_col \leftarrow g(X), g(Y), edge(X,Y). \\ non\_col \leftarrow b(X), b(Y), edge(X,Y). \\ \leftarrow not\ non\_col. \end{array}\right\} \textbf{Check}$$

$$\left.\begin{array}{l} r(X) \leftarrow non\_col, node(X). \\ g(X) \leftarrow non\_col, node(X). \\ b(X) \leftarrow non\_col, node(X). \end{array}\right\} \textbf{Saturate}$$

  - atom 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 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 $node_1$ of its nodes
  Problem: find a 3-coloring of the subgraph G_1 of G on $node_1$ that can not be extended to a 3-coloring of G.

$$
\begin{array}{ll}
b(X) \vee r(X) \vee g(X) \leftarrow node(X). & \} \textbf{ Guess} \\
\left.\begin{array}{l}
\leftarrow C(X), C(Y), edge(X,Y), node1(X), node1(Y). \\
non\_col \leftarrow C(X), C(Y), edge(X,Y), not\ node1(X). \\
non\_col \leftarrow C(X), C(Y), edge(X,Y), not\ node1(Y). \\
\leftarrow not\ non\_col.
\end{array}\right\} & \textbf{Check} \\
C(X) \leftarrow non\_col, node(X), not\ node1(X). & \} \textbf{ Saturate}
\end{array}
$$

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

- Note: SS3COL is complete for $\Sigma_2^p = \mathsf{NP}^{\mathsf{NP}}$

Complexity of Disjunctive Logic Programs /3

Theorem

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

- Σ_2^p-complete in the propositional case;
- NExpTime^{NP}-complete in the datalog case;
- Σ_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 $\exists \mathbf{M} \forall \mathbf{M}' \psi(\mathbf{M}, \mathbf{M}')$, where ψ is a Boolean combination of existential sentences.
  - rewrite to $\exists \mathbf{M} \forall \mathbf{x} \forall \mathbf{M}' \exists \mathbf{y} \psi'(\mathbf{M}, \mathbf{x}, \mathbf{M}', \mathbf{y})$, where ψ' is quantifier-free.
  - Emulate Arithmetic to express $\forall \mathbf{M}' \exists \mathbf{y} \psi'$ 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 $\vee$ 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 "$\neg$" (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."

$$walk \leftarrow at(L), crossing(L), not\ \ train\_approaches(L).$$

$$walk \leftarrow\ at(L), crossing(L), \neg train\_approaches(L).$$

- "$\neg$" is syntactic sugar, easily compiled away: for each predicate p,
  - replace $\neg p$ by a fresh predicate $p_\neg$.
  - add the constraint $\leftarrow p(\mathbf{X}), p_\neg(\mathbf{X})$.

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):

  $:\sim \texttt{b}_1,\ldots,\texttt{b}_k,\ \texttt{not}\ \texttt{b}_{k+1},\ldots,\ \texttt{not}\ \texttt{b}_m\ .\quad [\texttt{Weight}:\texttt{Level}]$

  where
  - all b_i are atoms (resp. "classical" literals)
  - $Weight$, $Level$ 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 : `:~ b1,...,bk, not bk+1,..., not bm .  [Weight : Level]`

- as usual, consider the grounding $grnd(wc)$ 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 ℓ is

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

  where $\mathcal{V}_i(I, \ell) = \{(\theta, w) \mid wc_i\theta = \ :\sim B.\ [w, \ell] \in grnd(wc_i), I \not\models wc_i\theta\}$
- I is safe, if each $c(I, \ell)$ is well-defined (all w's are numbers)
- a safe $M \in AS(P)$ dominates a safe $M' \in AS(P)$, if $c(M, \ell) < c(M', \ell)$ for some ℓ and $c(M, \ell') = c(M', \ell')$ for all $\ell' > \ell$
- a safe $M \in AS(P)$ is ***best (optimal)***, if no $M' \in AS(P)$ dominates M

Weak Constraints: Examples

Example 25 (Default values for weights and levels)

```
a v b.   c :- b.
:~ a.
:~ b.
:~ c.

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

Answer set $\{\mathtt{b}, \mathtt{c}\}$ is discarded because it violates two weak constraints!

Weak Constraints: Examples /2

Example 26 (Weights vs. Levels)

Weights:

```
a v b.
:~ a. [1:]
:~ a. [1:]
:~ b. [2:]

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

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

Levels:

```
a v b1 v b2.
:~ a. [:1]
:~ b1. [:2]
:~ b2. [:2]

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

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).

:~ assign(X,P), assign(Y,P), X!=Y, same_skill(X,Y). [:2]
:~ assign(X,P), assign(Y,P), X!=Y, married(X,Y). [:2]
:~ 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 $edge(\_,\_,\_)$ and a starting node $start(\_)$.

Problem: find a cheapest roundtrip beginning at the starting node

$$inPath(X,Y,C) \vee outPath(X,Y,C) \leftarrow edge(X,Y,C). \quad \} \textbf{ Guess}$$

$$\left.\begin{array}{l} \leftarrow inPath(X,Y,C),\ inPath(X,Y_1,C_1),\ Y \neq Y_1. \\ \leftarrow inPath(X,Y,C),\ inPath(X_1,Y,C_1),\ X \neq X_1. \\ \leftarrow node(X),\ not\ reached(X). \\ \leftarrow not\ start\_reached.\ \%\ \text{for cycle} \end{array}\right\} \textbf{ Check}$$

$$\left.\begin{array}{l} reached(X) \leftarrow start(X). \\ reached(X) \leftarrow reached(Y),\ inPath(Y,X,C). \\ start\_reached \leftarrow start(Y), inPath(X,Y,C). \end{array}\right\} \textbf{ Auxiliary Def.}$$

$$:\sim inPath(X,Y,C).[C{:}1] \quad \} \textbf{ Optimize}$$

Example: Minimum Spanning Tree

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

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

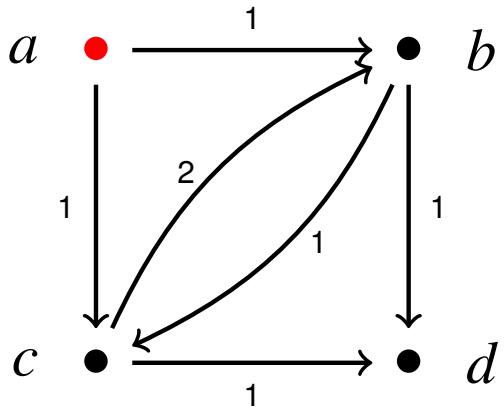
$$inTree(X,Y,C) \vee outTree(X,Y,C) \leftarrow edge(X,Y,C). \quad \} \textbf{ Guess}$$

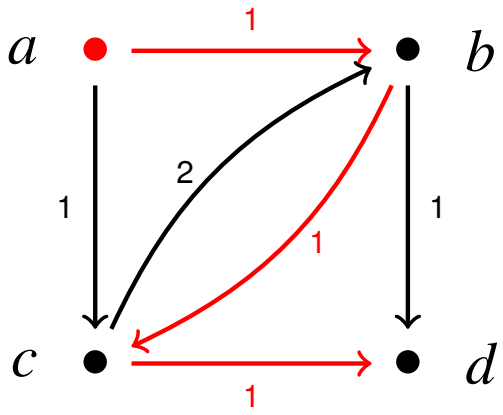
$$\left.\begin{array}{l} \leftarrow inTree(X,Y,C),\ start(Y). \\ \leftarrow inTree(X,Y,C),\ inTree(X_1,Y,C),\ X \neq X_1. \\ \leftarrow node(X),\ not\ reached(X). \end{array}\right\} \textbf{ Check}$$

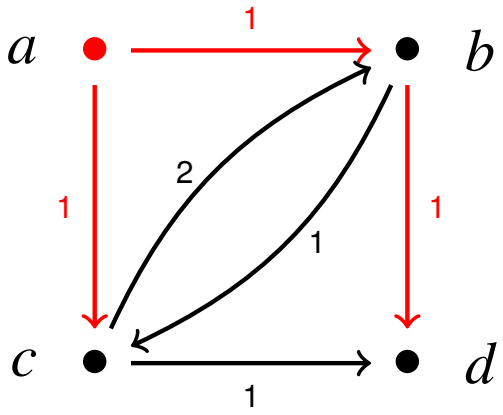
$$\left.\begin{array}{l} reached(X) \leftarrow start(X). \\ reached(X) \leftarrow reached(Y),\ inTree(Y,X,C). \end{array}\right\} \textbf{ Auxiliary Def.}$$

$$:\sim inPath(X,Y,C).[C{:}1] \quad \} \textbf{ Optimize}$$

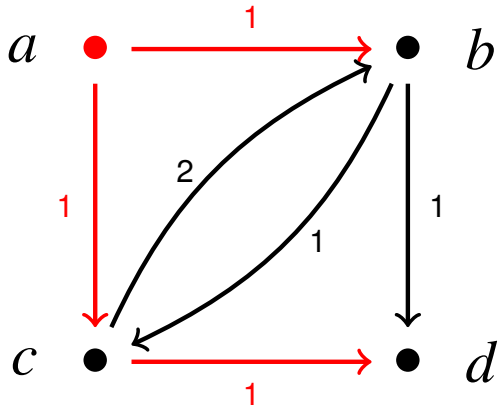
Example: Minimum Spanning Tree (ctd.)







$$
\begin{aligned}
P_D = \{ & node(a), node(b), \\
& node(c), node(d), \\
& edge(a,b,1), edge(a,c,1) \\
& edge(c,b,2), edge(b,c,1) \\
& edge(b,d,1), edge(c,d,1) \\
& start(a)\}
\end{aligned}
$$



Aggregates

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

```
select count(*) from empl;
```

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

$$\#count\{Emp, Dept, Job : empl(Emp, Dept, Job)\}$$

- these aggregate functions occur in aggregate atoms in rule bodies

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

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

Semantics of Programs with Aggregates

- Programs P are reduced to their groundings $grnd(P)$

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

Knowledge Representation Using Aggregates

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]
```

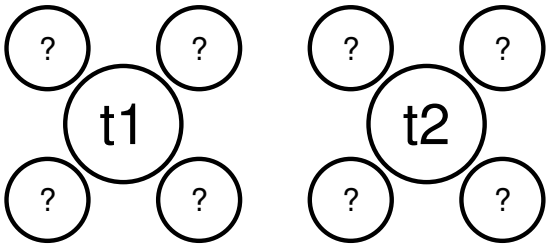
Example: Seating Problem

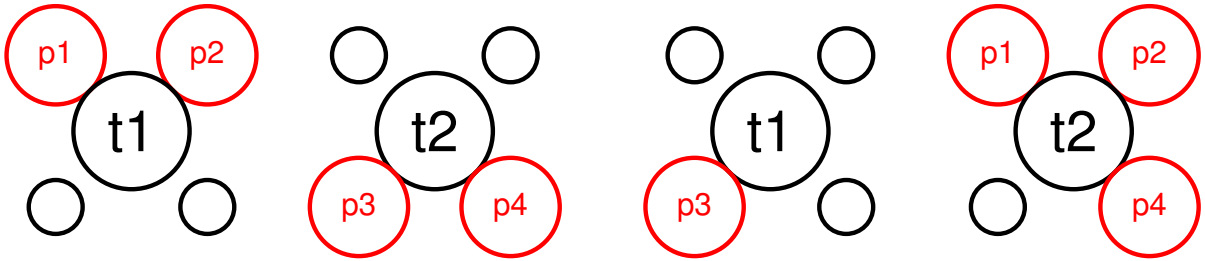
Problem: Given some tables of a given number of chairs each, generate a sitting arrangement for a number of given guests, such that:

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

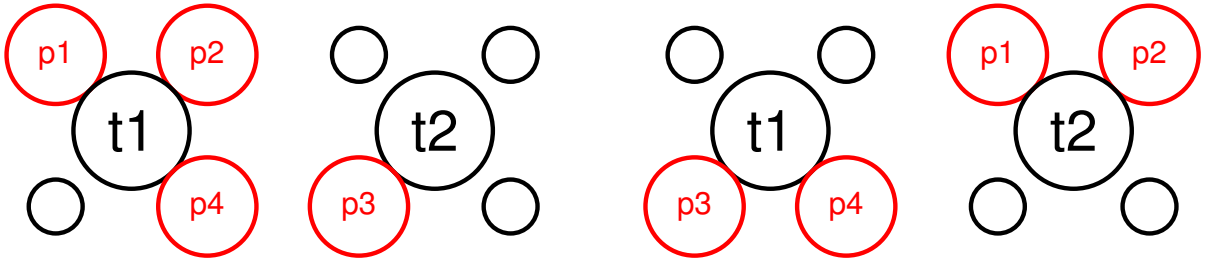
```
at(P,T) v not_at(P,T) :- person(P), table(T).
:- table(T), nchairs(C), not#count{P : at(P,T)} <= C.
:- person(P), not #count{T : at(P,T)} = 1.
:- like(P1,P2), at(P1,T), not at(P2,T).
:- dislike(P1,P2), at(P1,T), at(P2,T).
```

Example: Seating Problem (ctd.)





$$
\begin{aligned}
P_D = \{ & person(p1), person(p2), \\
& person(p3), person(p4), \\
& table(t1), table(t2), \\
& nchairs(4), \\
& like(p1, p2), \\
& dislike(p1, p3)\}
\end{aligned}
$$



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)

    $\Rightarrow$ 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

Ofer Arieli, Marc Denecker, Bert Van Nuffelen, and Maurice Bruynooghe.
Database repair by signed formulae.
In Dietmar Seipel and Jose Maria Turull Torres, editors, *FoIKS*, volume 2942 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 2004.

Chitta Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2002.

Gerd Brewka, Thomas Eiter, and Miroslaw Truszczyński.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.
doi>10.1145/2043174.2043195.

Marco Cadoli, Toni Mancini, Davide Micaletto, and Fabio Patrizi.
Evaluating asp and commercial solvers on the csplib.
In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI*, pages 68–72. IOS Press, 2006.

Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub.
Asp-core-2: Input language format, 2012.
https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf.

References II

Elvin Coban, Esra Erdem, and Ferhan Türe.
Comparing asp, cp, ilp on two challenging applications: wire routing and haplotype inference.
In *LaSh 2008: LOGIC AND SEARCH - Computation of structures from declarative descriptions, International Workshop, Leuven, Belgium: Katholieke Universiteit Leuven, November 2008*, pages 166–180, 2008.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov.
Complexity and Expressive Power of Logic Programming.
ACM Computing Surveys, 33(3):374–425, 2001.

Agostino Dovier, Andrea Formisano, and Enrico Pontelli.
An experimental comparison of constraint logic programming and answer set programming.
In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 1622–1625. AAAI Press, 2007.

Thomas Eiter, Georg Gottlob, and Heikki Mannila.
Disjunctive Datalog.
ACM Transactions on Database Systems, 22(3):364–417, September 1997.

Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner.
Answer set programming: A primer.
In Sergio Tessaris and Enrico Franconi et al., editors, *Reasoning Web, Fifth International Summer School 2008, Bressanone Italy, August 30–September 4, 2009, Tutorial Lectures*, number 5689 in LNCS, pages 40–110. Springer, 2009.

References III

Paolo Ferraris and Vladimir Lifschitz.
Mathematical foundations of answer set programming.
In *We Will Show Them! Essays in Honour of Dov Gabbay*, pages 615–664. King's College Publications, 2005.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

M. Gelfond and V. Lifschitz.
The Stable Model Semantics for Logic Programming.
In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

M. Gelfond and V. Lifschitz.
Classical Negation in Logic Programs and Disjunctive Databases.
New Generation Computing, 9:365–385, 1991.

Michael Gelfond.
Answer sets.
In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, Foundations of Artificial Intelligence, chapter 7, pages 285–316. Elsevier, 2007.

References IV

Georg Gottlob, Nicola Leone, and Helmut Veith.
Succinctness as a Source of Expression Complexity.
Annals of Pure and Applied Logic, 97(1–3):231–260, 1999.

Arend Heyting.
Die formalen Regeln der intuitionistischen Logik.
Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-Mathematische Klasse, pages 42–56, 1930.

Ph.G. Kolaitis and Ch. H. Papadimitriou.
Why Not Negation By Fixpoint ?
Journal of Computer and System Sciences, 43:125–144, 1991.

Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello.
The DLV System for Knowledge Representation and Reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.
Available via `http://www.arxiv.org/ps/cs.AI/0211004`.

Vladimir Lifschitz, David Pearce, and Agustín Valverde.
Strongly equivalent logic programs.
ACM Trans. Comput. Log., 2(4):526–541, 2001.

References V

Vladimir Lifschitz, David Pearce, and Agustín Valverde.
A characterization of strong equivalence for logic programs with variables.
In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 188–200. Springer, 2007.

Vladimir Lifschitz.
Answer Set Planning.
In *Proc. 16th International Conference on Logic Programming* (*ICLP '99*), pages 23–37. MIT Press, 1999.

Vladimir Lifschitz.
Answer Set Programming and Plan Generation.
Artificial Intelligence, 138:39–54, 2002.

Vladimir Lifschitz.
Twelve definitions of a stable model.
In *ICLP*, pages 37–51, 2008.

Victor W. Marek and Mirosław Truszczyński.
Stable Models and an Alternative Logic Programming Paradigm.
In K. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren, editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

References VI

Ilkka Niemelä.
Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3–4):241–273, 1999.

David Pearce.
Equilibrium logic.
Ann. Math. Artif. Intell., 47(1-2):3–41, 2006.

J.S. Schlipf.
Complexity and Undecidability Results in Logic Programming.
Annals of Mathematics and Artificial Intelligence, 15(3/4):257–288, 1995.

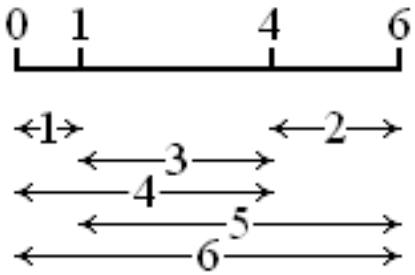
A. van Gelder, K.A. Ross, and J.S. Schlipf.
The Well-Founded Semantics for General Logic Programs.
Journal of the ACM, 38(3):620–650, 1991.

Stefan Woltran.
A common view on strong, uniform, and other notions of equivalence in answer-set programming.
Theory and Practice of Logic Programming, 8(2):217–234, 2008.

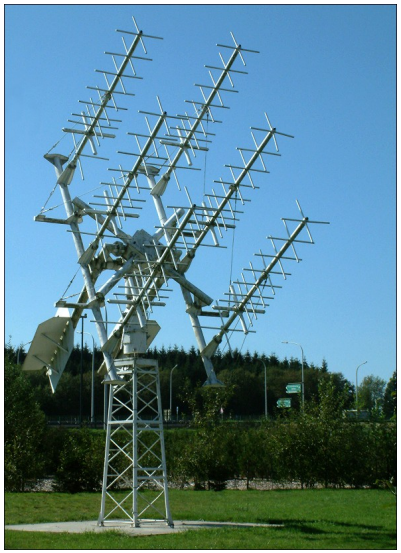
Assignments

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
      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

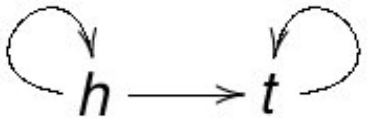
Basis for semantic characterization of answer sets:

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)

$$b_1 \wedge \ldots \wedge b_m \wedge \neg b_{m+1} \wedge \ldots \wedge \neg b_n \rightarrow a_1 \vee \cdots \vee a_l, \tag{3}$$

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



➥ 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. $M, w \models \phi \wedge \psi$ if $M, w \models \phi$ and $M, w \models \psi$,
  4. $M, w \models \phi \vee \psi$ 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. $(X, Y) \models \phi \wedge \psi$ if $(X, Y) \models \phi$ and $(X, Y) \models \psi$,
  4. $(X, Y) \models \phi \vee \psi$ 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

- $P = \{a \leftarrow not\ b;\ b \leftarrow not\ a\}$ has the eq. models (a, a) and (b, b)
- $P = \{a \leftarrow not\ a\}$ 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)

$P = \{ a \leftarrow not\ b \}$, $P' = \{ a \}$, $Q = \{ b \leftarrow not\ a \}$.

- 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 $E = c_1 \wedge \cdots \wedge c_m$, where $c_i = l_{i,1} \vee \cdots \vee l_{i,k_i}$, and each $l_{i,j}$ is a literal over $X = \{x_1, \ldots, x_n\}$.

  Construct the following program P:

$$
\begin{array}{rll}
x_i \leftarrow not\ \neg x_i & & \text{for each } x_i \in X \\
\neg x_i \leftarrow not\ x_i & & \\
c_i \leftarrow l_{i,j} & & \text{for all } l_{i,j} \\
sat \leftarrow c_1, \ldots, c_m & & \\
sat \leftarrow not\ sat & &
\end{array}
$$

  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 NEXPTIME.
- The problems are also 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 Σ_1^1-complete.

Note: Σ_1^1 is in the *Analytic Hierarchy*, and more expressive than the classes Σ_n^0, $n \geq 0$, in the *Arithmetic Hierarchy*.

Cautious reasoning from positive normal programs is r.e. (Σ_1^0)-complete.

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

- answer set existence is expressible by a second-order sentence of form $\exists \mathbf{T} \forall \mathbf{x} \exists \mathbf{y} \phi(\mathbf{T}, \mathbf{x}, \mathbf{y})$, where $\phi(\mathbf{T}, \mathbf{x}, \mathbf{y})$ is quantifier-free.
- Every Σ_1^1 sentence is convertible to a sentence Φ of this form (by second-order Skolemization)
  Φ is expressible (over the Herbrand universe) by answer set existence.

Complexity: FO Logic Programs /2

- Suppose wlog $\phi_i(\mathbf{T}\mathbf{x}, \mathbf{y}) = \bigvee_{i=1}^{n} \phi_i(\mathbf{T}, \mathbf{x}, \mathbf{y})$ is in DNF.
- Then, the program

$$
\begin{aligned}
T(\mathbf{x}) &\leftarrow not\ \ T'(\mathbf{x}) && \text{for each } T \in \mathbf{T} \\
T'(\mathbf{x}) &\leftarrow not\ \ T(\mathbf{x}) \\
sat(\mathbf{x}) &\leftarrow \phi_i^{\star}(\mathbf{T}, \mathbf{x}, \mathbf{y}) && \text{for each } i = 1, \ldots, n \\
sat(\mathbf{x}) &\leftarrow not\ \ sat(\mathbf{x}) \\
eq(x, x) &\leftarrow
\end{aligned}
$$

where $\phi^{\star}(\mathbf{T}, \mathbf{x}, \mathbf{y})$ is $\phi(\mathbf{T}, \mathbf{x}, \mathbf{y})$ but

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

has some answer set iff Φ is true on its Herbrand universe.

Disjunctive Programs: Propositional Case

Claim: For disjunctive propositional P, answer set existence is Σ_2^p-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 $\mathsf{NP}^{\mathsf{NP}} = \Sigma_2^p$ 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 $F = d_1 \vee \cdots \vee d_m$ is a DNF, where $d_i = l_{i,1} \wedge \cdots \wedge l_{i,k_i}$, 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 Φ

$$
\begin{array}{rll}
x_i \vee \neg x_i \leftarrow & & \text{for each } x_i \\
y_i \vee y'_i \leftarrow & & \text{for each } y_j \\
sat \leftarrow \sigma(l_{i,1}), \ldots, \sigma(l_{i,k_i}) & & \text{for all } i \\
y_i \leftarrow sat & & \text{for each } y_j \\
y'_i \leftarrow sat & & \\
sat \leftarrow not\ sat & &
\end{array}
$$

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 Φ 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

- $\#aggr \in \{\#count, \#sum, \#min, \#max\}$;
- 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 $\#times$ (product)

Aggregate Atoms /2

Example 36

- $\#count\{Emp, Dept : empl(Emp, Dept, Job)\} < 10$
- $not\ \#count\{Emp, Dept : empl(Emp, Dept, Job)\} < 10$
- $20 \leq \#sum\{Sal, Emp : sal(Emp, Sal)\} \leq 50$

Aggregate atoms may occur in rule bodies:

Example 37

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

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

Aggregate Atom Semantics

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