Statistical Relational Extension of Answer Set Programming

@Reasoning Web School 2022

Joohyung Lee

Global AI Center
Samsung Research

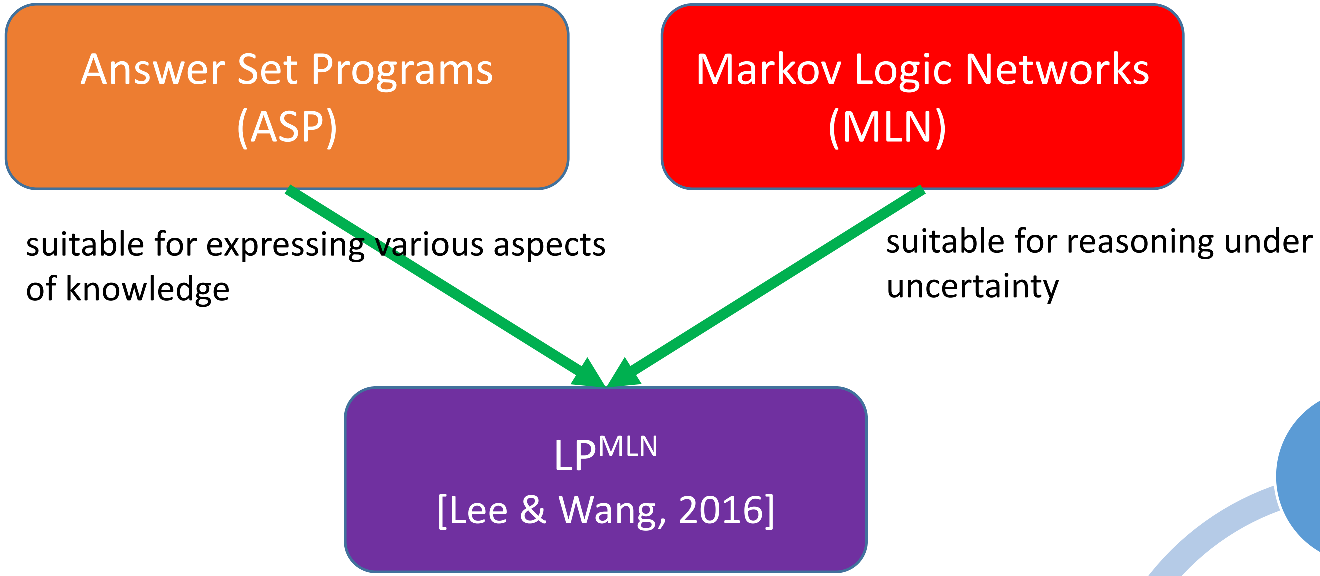
School of Computing and AI
Arizona State University

Combining Logic and Probability

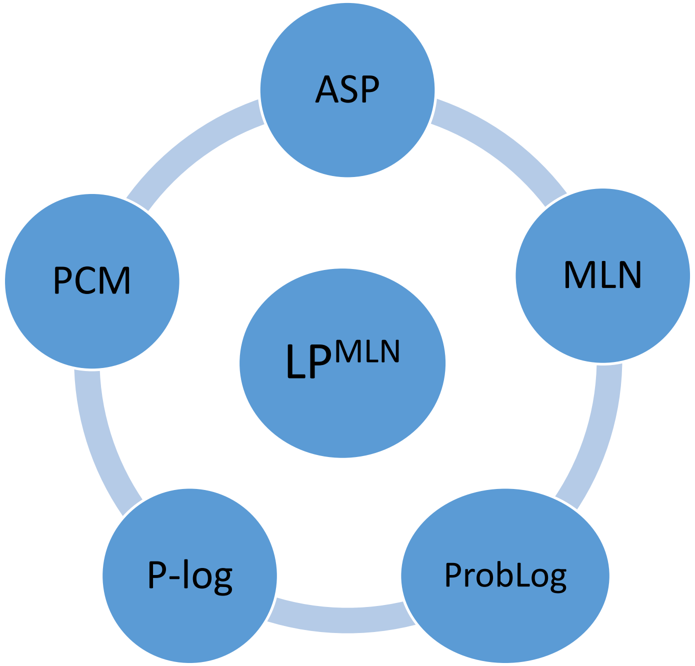
- The main goal of the representation in SRL is to express probabilistic models in a compact way that reflects the relational structure of the domain, and ideally supports efficient learning and inference.
  - BLP, BLOG, PRM, MLN, PSL, ProbLog, RBN, RDN, . . .

- Related to Neuro-symbolic AI

What the Tutorial is About



Relationship between LP^{MLN} and several other formalisms were established:
[Lee & Wang, 2016; Lee, Meng & Wang 2015; Lee & Wang, 2015]



Answer Set Programming

- ASP (Answer Set Programming) is a declarative programming paradigm that is based on the stable model semantics
- ASP is effective and widely used on knowledge intensive domains and combinatorial search problems
- However, the deterministic nature of ASP limits its application in domains involving probability and inconsistencies

Answer Set Programming

- **Declarative programming paradigm combining**
  - a rich yet simple modeling language
  - with high-performance solving capacities

- **ASP is useful for knowledge-intensive tasks and combinatorial search problems**

- **ASP has its roots in**
  - logic programming
  - knowledge representation
  - constraint solving (in particular SAT)
  - (deductive) databases

ASP = LP + KR + SAT + DB

Markov Logic

- Markov logic combines first-order logic with Markov networks
- A Markov logic network consists of a set of weighted first-order formulas
- The probability of a world is proportional to the exponential of the sum of the formulae that are true in the world
- The idea is to view logical formulas as soft constraints on the set of possible worlds

Markov Logic vs. ASP

Markov Logic

+ Uncertainty with knowledge base

– Based on classical first-order logic

Can't handle inductive definition, causality, …

ASP

+ Rich KR constructs (choice rules, aggregates, …)

+ Rule-based semantics

Can handle transitive closure, causality

– Does not handle (probabilistic) uncertainty well

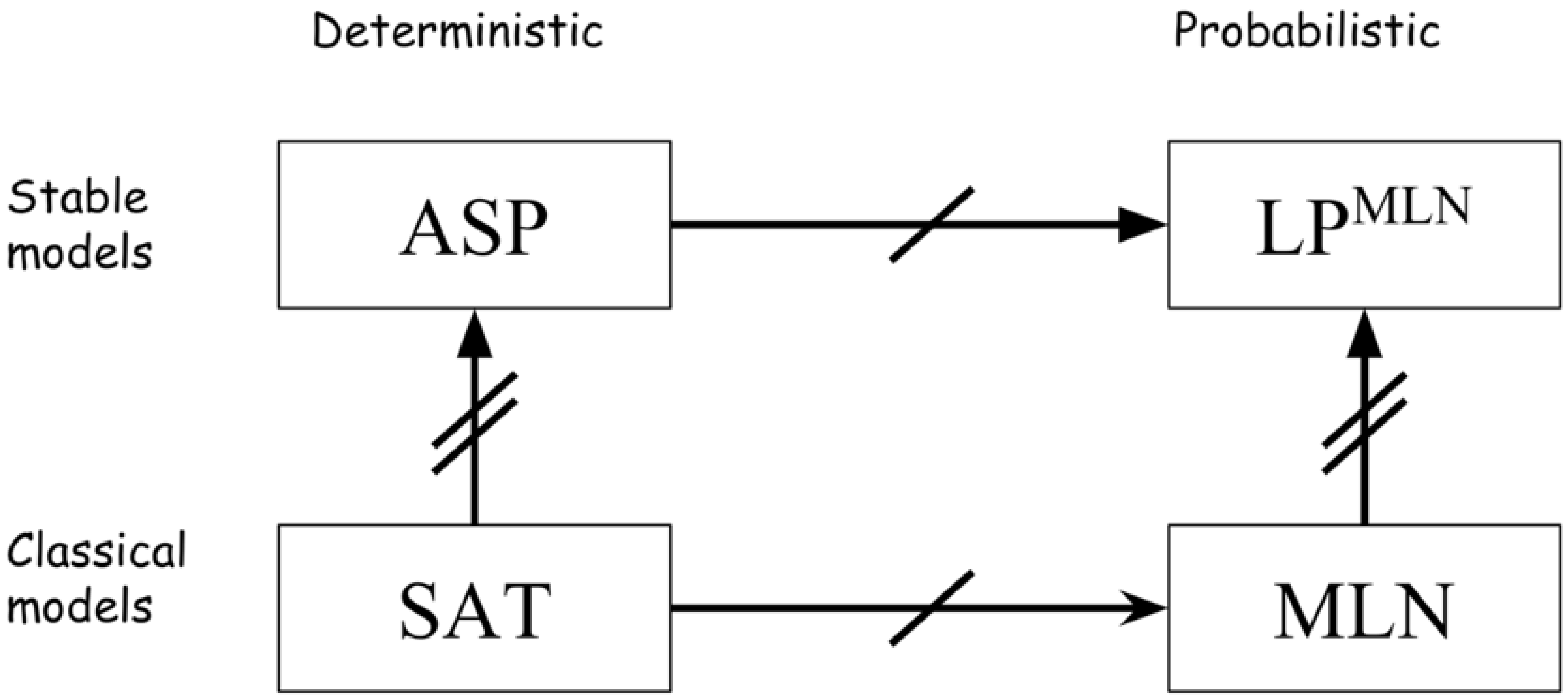
LP^{MLN}

- **A logic formalism with weighted rules under the stable model semantics, following the log-linear models of Markov Logic**

- **It provides versatile methods to overcome the deterministic nature of the stable model semantics, such as:**
  - Resolving inconsistencies in answer set programs
  - Define ranking/probability distribution over stable models
  - Apply methods from machine learning to compute KR formalisms

LP^{MLN}

- A simple approach to combining answer set programming (ASP) and Markdov Logic (MLN)



Outline

1. Introduction
2. **Intro to ASP**
3. Stable Model Semantics
4. Syntax and Semantics of LPMLN
5. Relating LPMLN to Other Languages
6. Inference in LPMLN
7. Learning in LPMLN
8. Extension to Embrace Neural Networks

Problem Solving

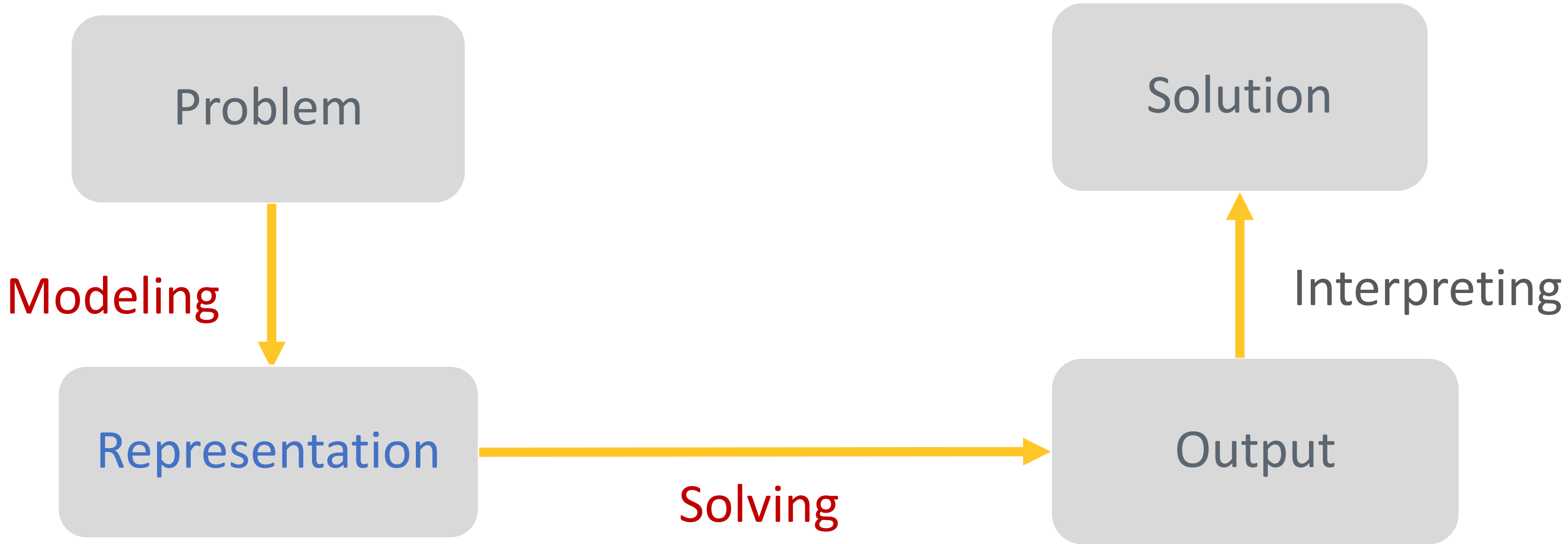
<u>“What is the problem?”</u> versus <u>“How to solve the problem?”</u>

Traditional Programming

<u>"What is the problem?"</u> versus <u>"How to solve the problem?"</u>

Declarative Programming

<u>"What is the problem?"</u> versus <u>"How to solve the problem?"</u>



What is Answer Set Programming

- **Declarative programming paradigm suitable for knowledge intensive and combinatorial search problems**

- **Theoretical basis: stable model semantics (Gelfond and Lifschitz, 1988)**

- **Expressive representation language**
  - defaults
  - negation as failure
  - recursive definitions
  - aggregates
  - preferences
  - etc.

What is Answer Set Programming, cont’d

ASP solvers

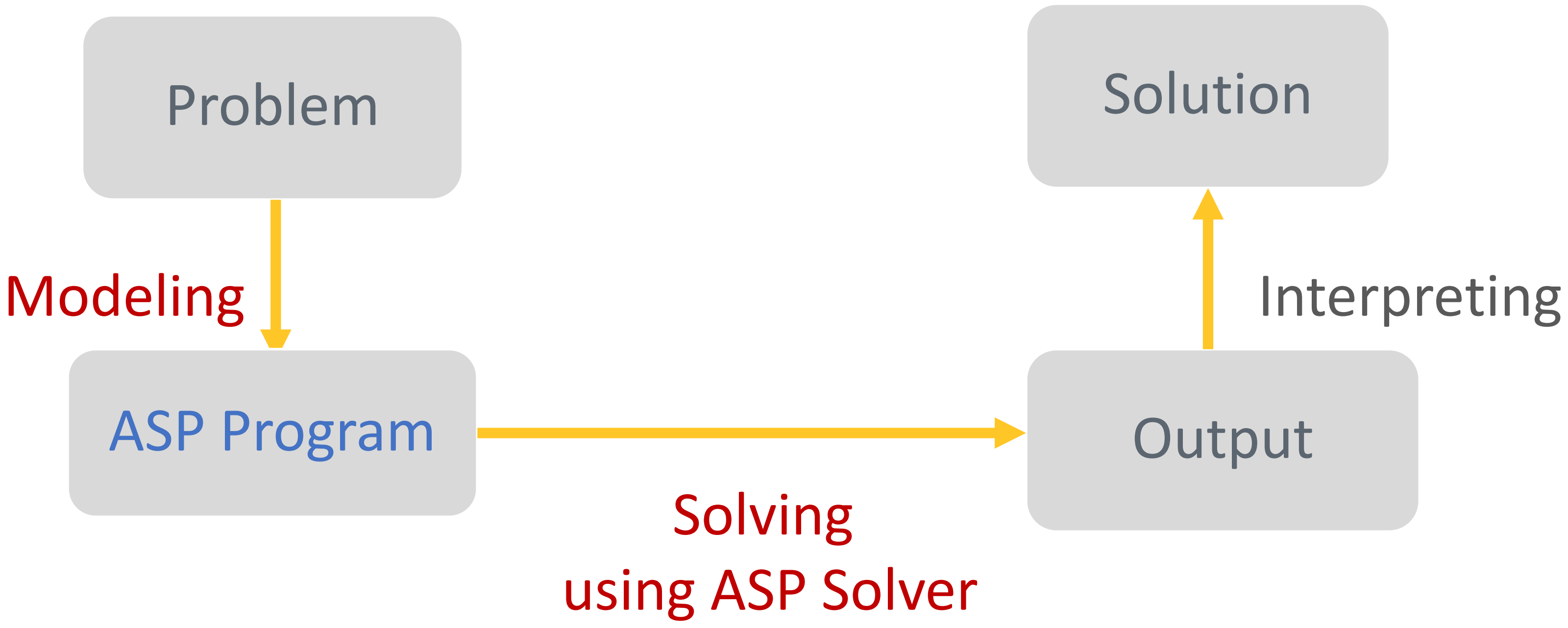
- smodels (Helsinki University of Technology, 1996)
- dlv (Vienna University of Technology, 1997)
- cmodels (University of Texas at Austin, 2002)
- pbmodels (University of Kentucky, 2005)
- Clasp/clingo (University of Potsdam, 2006) – winning several first places at ASP, SAT, Max-SAT, PB, CADE competitions
- Wasp (University of Cabria, 2013)
- dlv-hex for computing HEX programs
- oClingo for reactive answer set programming
- …

ASP Core 2: Standard language

Declarative Problem Solving using ASP

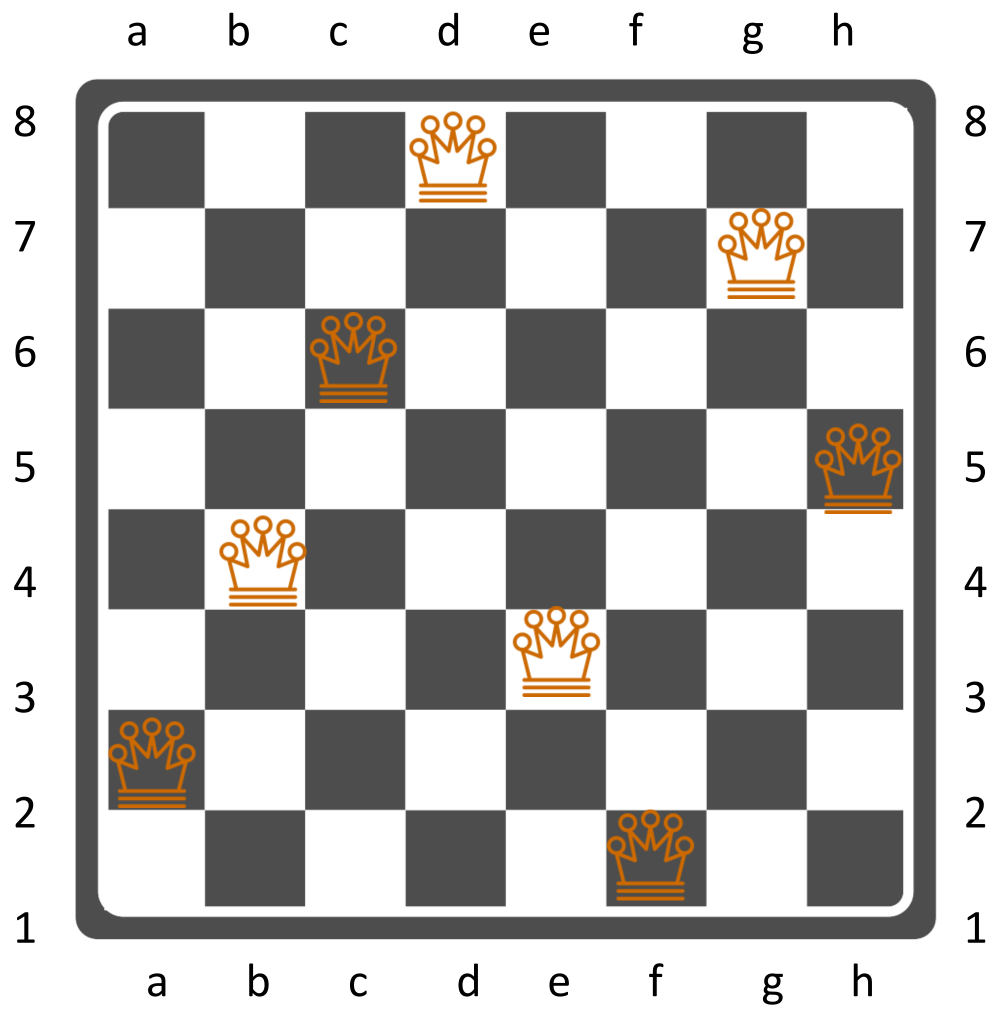
The basic idea is

- to present the given problem by a set of rules,
- to find answer sets for the program using an ASP solver,
- and to extract the solutions from the answer sets.



N-Queens Puzzle

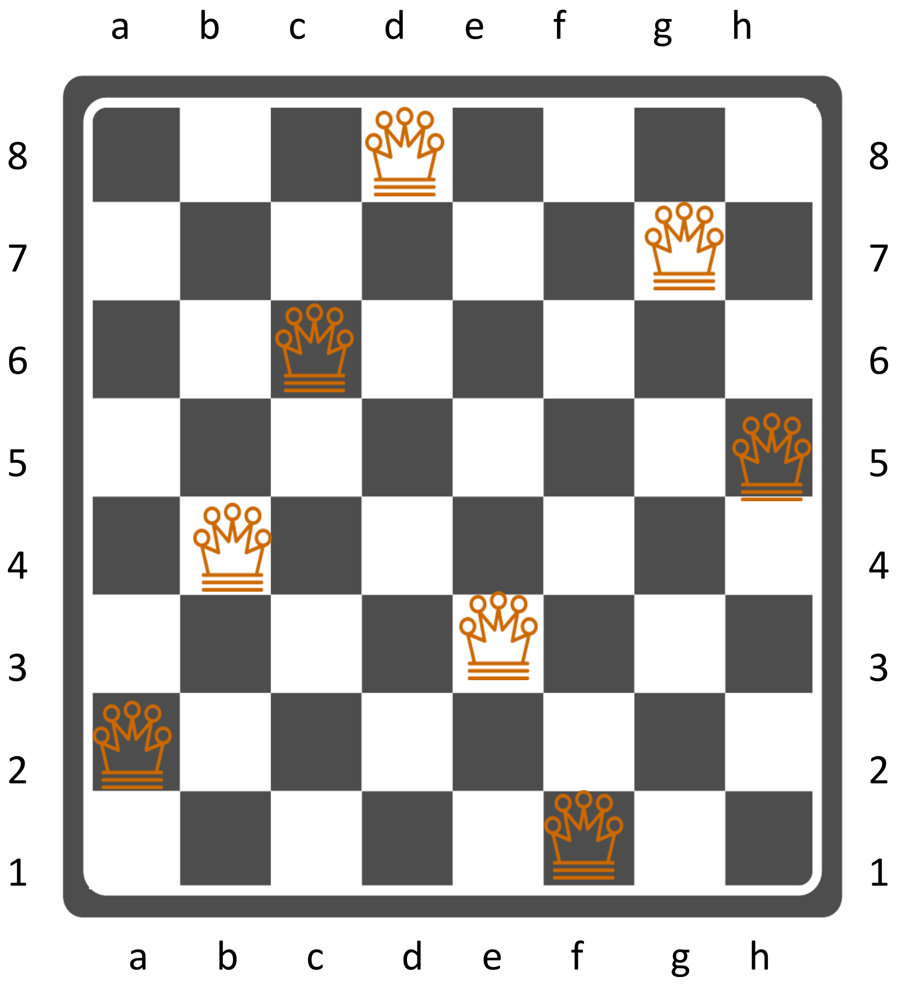
No two queens can share the same row, column, or diagonal



n	# sol
3	none
4	2
5	20
6	4
7	40
8	92

N-Queens Puzzle, cont'd

No two queens can share the same row, column, or diagonal




```
% Each row has exactly one queen
1 {queen(R,1..n)} 1 :- R=1..n.

% No two queens are on the same column
:- queen(R1,C), queen(R2,C), R1!=R2.

% No two queens are on the same diagonal
:- queen(R1,C1), queen(R2,C2), R1!=R2, |R1-R2|=|C1-C2|.
```

Finding One Solution for the 8-Queens Puzzle

```
$ clingo queens.lp -c n=8
clingo version 5.2.1
Reading from queens.lp
Solving...
Answer: 1
queen(4,1) queen(6,2) queen(8,3) queen(2,4) queen(7,5) queen(1,6)
queen(3,7) queen(5,8)
SATISFIABLE


Models       : 1+
Calls        : 1
Time         : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.004s
```

Finding All Solutions for the 8-Queens Puzzle

```
$ clingo queens.lp -c n=8 0
clingo version 5.2.1
Reading from queens.lp
Solving...
Answer: 1
queen(4,1) queen(6,2) queen(8,3) queen(2,4) queen(7,5) queen(1,6)
queen(3,7) queen(5,8)
Answer: 2
[[ truncated ]]
Answer: 92
queen(5,1) queen(1,2) queen(8,3) queen(4,4) queen(2,5) queen(7,6)
queen(3,7) queen(6,8)
SATISFIABLE

Models       : 92
Calls        : 1
Time         : 0.011s (Solving: 0.01s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.010s
```

Outline

1. Introduction
2. Intro to ASP
3. **Stable Model Semantics**
4. Syntax and Semantics of LPMLN
5. Relating LPMLN to Other Languages
6. Inference in LPMLN
7. Learning in LPMLN
8. Extension to Embrace Neural Networks

Stable Model Semantics

Syntax of Propositional Rules

- **We consider rules as the restricted form of formulas in which implications occur in a limited way.**
  - We write $F \leftarrow G$ to denote $G \rightarrow F$

- **A (propositional) rule is a formula of the form $F \leftarrow G$ where F and G are implication-free ($\bot, \top, \neg, \wedge, \vee$ are allowed in F and G)**
  - We often write $F \leftarrow \top$ simply as F

- **Example: Is each of the following a propositional rule?**
  - $p \leftarrow (q \vee \neg r)$
  - $p \rightarrow (q \rightarrow r)$
  - $(p \vee q) \wedge \neg r$

- **A propositional program is a set of propositional rules.**

Representing Interpretations as Sets

- **We identify an interpretation with the set of atoms that are true in it.**
  - **Example:** interpretations of signature $\{p, q\}$

    $\emptyset$ $\{p\}$ $\{q\}$ $\{p, q\}$

  - **Example:** for signature $\{p, q\}$, the formula $p \vee q$ has three models:

Minimal Models: Definition

About a model I of a formula F, we say that it is minimal if no other model of F is a subset of I.

- **Example:** For signature $\{p, q\}$, the formula $p \vee q$ has three models: $\{p\}, \{q\}, \cancel{\{p, q\}}$.
- The minimal models are
  - $\{p\}$ *and* $\{q\}$

Exercise: Find all minimal models of the program

$$\{p \leftarrow q, \quad q \vee r\}.$$

$\{p, q\}, \{r\}$

Minimal Models: A Question

Statement: If two formulas are equivalent under propositional logic, then they have the same minimal models.

Question: Is the converse true, that two formulas having the same minimal models are equivalent?

$p \vee q$ $\quad$ $(p \vee q) \wedge \neg(p \wedge q)$

$\{p\}, \{q\}$

Informal Reading: Rationality Principle

- **Informally, program Π can be viewed as a specification for stable models—sets of beliefs that could be held by a rational reasoner associate with Π.**

$$p \leftarrow \top$$
$$r \leftarrow p \wedge q$$

$\{p\}$

Informal Reading: Rationality Principle, cont'd

Stable models will be represented by collections of atoms. In forming such sets the reasoner must be guided by the following informal principles:

- Satisfy the rules of Π. In other words, if one believes in the body of a rule, one must also believe in its head.
- Adhere to the "the rationality principle," which says, "Believe nothing you are not forced to believe.

$$p$$

$$r \leftarrow p \wedge q$$

Stable Models of Programs with Negation

Prolog vs. ASP

? p

```
p :- not q
q :- not p
```

$p \leftarrow \neg q$
$q \leftarrow \neg p$

Prolog does not terminate on query p or q

```
?- p.
   ERROR: Out of local stack
      Exception: (729,178)
```

clingo returns

```
Answer: 1
p
Answer: 2
q
```

Finite ASP programs are guaranteed to terminate

Negation as Failure

Q: How do we extend the definition of a stable model in the presence of negation?

$p,$	$p,$	$p,$	$p,$
$q,$	$q,$	$q,$	
$r \leftarrow p,$	$r \leftarrow p \land \neg s,$	$r \leftarrow p \land \neg s,$	$r \leftarrow p \land \neg s,$
$s \leftarrow q$	$s \leftarrow q.$		$s \leftarrow q.$
$\{p, q, r, s\}$	$\{p, q, s\}$	$\{p, q, r\}$	$\{p, r\}$

Add r to the model if p is included under the condition that s is not included in the model and will not be included in the future.

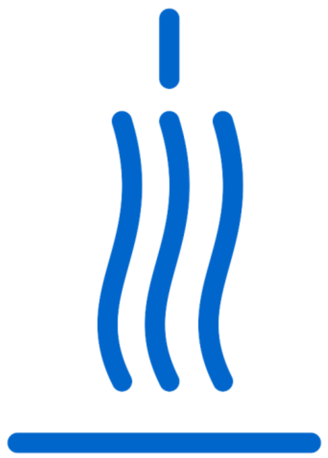
Informal Reading: Rationality Principle

Informally, program Π can be viewed as a specification for stable models--sets of beliefs that could be held by a rational reasoner associated with Π.

Stable models will be represented by collections of atoms.

In forming such sets the reasoner must be guided by the following informal principles:

- Satisfy the rules of Π.
  - If one believes in the body of a rule, one must also believe in its head.
- Adhere to the “the rationality principle.”
  - “Believe nothing you are not forced to believe.”

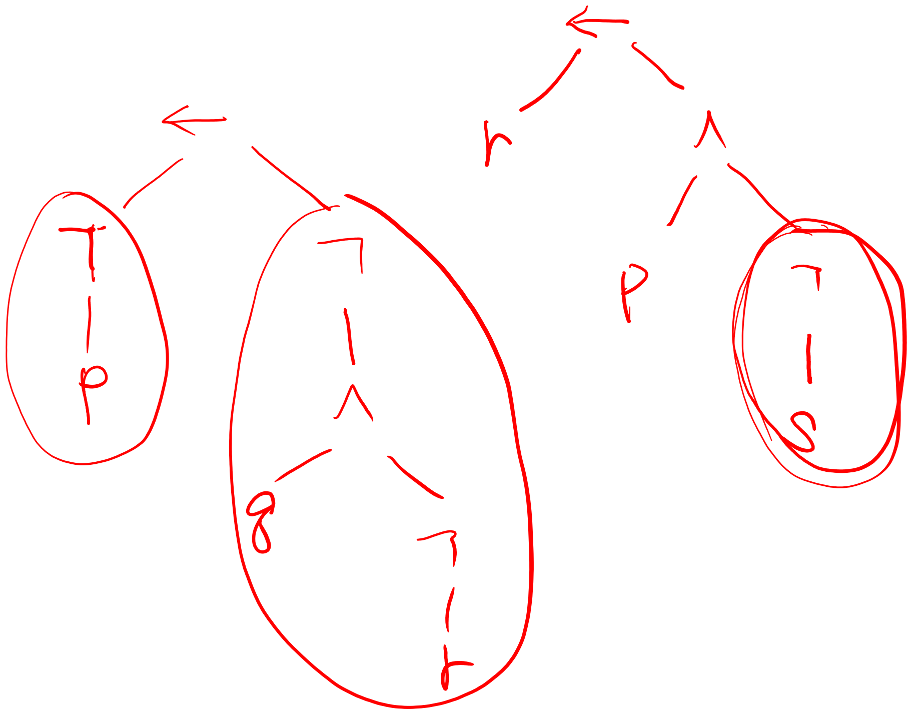


Critical Part

A **critical part** of a propositional rule is a subformula of its head or body that begins with negation but is not part of another subformula that begins with negation.

Example: Find the critical parts of the formulas

- $r \leftarrow p \wedge \neg s$
- $\neg p \leftarrow \neg(q \wedge \neg r)$
- $p \leftarrow \neg\neg p$
- $p \vee \neg p$



Stable Models of Programs with Negation

- The **reduct** Π^X of Π relative to an interpretation X is the **positive** propositional program obtained from Π by replacing each critical part $\neg H$ of each of its rules
  - by $\top$ if X satisfies $\neg H$;
  - by $\bot$ otherwise

- X is a **stable model** of Π If X is a minimal model of the reduct Π^X

Example:

Γ

$$p,$$
$$q,$$
$$r \leftarrow p \wedge \neg s,$$
$$s \leftarrow q.$$

$\Gamma^{\{p,q,s\}}$ → SM

p

q

$r \leftarrow p \wedge \bot$

$s \leftarrow q$

$\Gamma^{\{p,q\}}$ → Not SM

$p.$

$q.$

$r. \leftarrow p. \wedge \top$

$s. \leftarrow q$

$\Gamma^{\{p,q,r\}}$

not SM

p

q

$r \leftarrow p \wedge \top$

$s \leftarrow q$

Steps to Find Stable Models (Succinct)

Given a propositional program Π

1. **Guess an interpretation X**
2. **Find the reduct of Π relative to X (i.e., Π^X)**
3. **Check if X is a minimal model of Π^X (note that Π^X is a positive program; has no negation)**
   a. If yes, conclude X is a stable model of Π
   b. If no, conclude X is **not** a stable model of Π

Steps to Find Stable Models (Verbose)

Given a propositional program Π

1. **Guess an interpretation X**
2. **Find the reduct of Π relative to X (i.e., Π^X)**
3. **Check if X satisfies Π^X (Alternatively, check if X satisfies Π)**
   a. If yes, continue
   b. If no, conclude X is **not** a stable model of Π
4. **Check if no other interpretation that is smaller than X satisfies Π^X. I.e., for each interpretation Y that is smaller than X,**
   a. If Y satisfies Π^X, conclude X is **not** a stable model of Π
   b. Else continue
5. **Conclude X is a stable model of Π**

NOTES:

- Every stable model is a model.
- The red part can't be replaced with Π .

Classical Equivalence vs. Stable Models

- **Equivalent propositional programs can have different stable models.**

- **Example:**

$$p \leftarrow \neg q \, , \quad q \leftarrow \neg p \, , \quad p \vee q$$

$p \vee \neg p$ and $q \vee \neg q$

Minimal Models vs. Stable Models

Recall the definition:

X is a stable model of Π if X is a minimal model of Π^X

Claim: For any program Π,

X is a stable model of Π if X is a minimal model of Π

True or false?

$\Pi = p \vee \neg p$ $\qquad \emptyset$

$X = \emptyset \quad \Pi^{\emptyset} = p \vee \top \Leftrightarrow \top \qquad \emptyset \qquad$ SM

$X = \{p\} \quad \Pi^{\{p\}} = p \vee \bot \Leftrightarrow p \qquad \{p\} \qquad$ SM

Choice Rule

- **Stable models of** $p \vee \neg p$ $\emptyset$ $\{p\}$

- **Stable models of** $(p \vee \neg p) \wedge (q \vee \neg q)$ $\emptyset$, $\{p\}$, $\{q\}$, $\{p, q\}$

- **Stable models of** $(p_1 \vee \neg p_1) \wedge (p_2 \vee \neg p_2) \wedge \cdots \wedge (p_n \vee \neg p_n)$

- **We abbreviate the formula** $(p_1 \vee \neg p_1) \wedge (p_2 \vee \neg p_2) \wedge \cdots \wedge (p_n \vee \neg p_n)$ **as** $\{p_1; \ldots; p_n\}$ **and call it choice rule.**

Choice Rules in Clingo

Choice rules describe several ways to form a stable model.

```
{p(a);q(b)}.
```

says choose which of the atoms `p(a),q(b)` **to include in the model**

```
% clingo choice.lp 0

Answer: 1

Answer: 2 q(b)

Answer: 3 p(a)

Answer: 4 p(a) q(b)
```

Choice Rules with Intervals and Pools

```
{p(1..3)}.
```

has the same meaning as

```
{p(1);p(2);p(3)}.
```

```
{p(a;b;c)}.
```

has the same meaning as

```
{p(a);p(b);p(c)}.
```

Choice Rules with Cardinality Bounds

```
1 {p(1..3)} 2.
```

describes the subsets of {1,2,3} that consists of 1 or 2 elements.

```
Answer: 1 p(2)

Answer: 2 p(3)

Answer: 3 p(2) p(3)

Answer: 4 p(1)

Answer: 5 p(1) p(3)

Answer: 6 p(1) p(2)
```

Choice Rules with Variables

```
1 {p(X);q(X)} 1 :- X=1..2.
```

1 {p(1) ; q(1)} 1

1 {p(2); q(2)} 1

```
Answer: 1
q(1) p(2)
Answer: 2
q(1) q(2)
Answer: 3
p(1) p(2)
Answer: 4
p(1) q(2)
```

X is a global variable

Local vs. Global Variables

```
{p(I): I=1..7}.
```

- I is a local variable
- A local variable is a variable such that all its occurrences in the rule are in between { ... }
- Other variables are global variables
- The rule expands into

```
{p(1); p(2); p(3); p(4); p(5); p(6); p(7)}.
```

- **Q: How many stable models are there?**

(a) 0 (b) 7 (c) 64 (d) 128

Local vs. Global Variables, cont'd

```
{p(I)} : I=1..7.
```

- I is a global variable because it has an occurrence outside { … }

- The rule expands into

```
{p(1)}.
{p(2)}.
{p(3)}.
{p(4)}.
{p(5)}.
{p(6)}.
{p(7)}.
```

- Q: How many stable models are there?

(a) 0 (b) 7 (c) 64 (d) 128

Local vs. Global Variables, cont'd

```
{q(I,J): J=1..3} :- I = 1..2.
```

{q(1, J) : J=1..3}
⇒ {q(1,1), q(1,2), q(1,3)}

Q: How many stable models are there?

(a) 6 (b) 8 (c) 12 (d) 64

{q(2, J) : J = 1..3}
→ {q(2,1), q(2,2), q(2,3)}

The rule expands into

```
{q(1,1);q(1,2);q(1,3)}.
{q(2,1);q(2,2);q(2,3)}.
```

Constraints

A **constraint** is a rule that has no head, e.g., :- p(1)

- which can be understood as ⊥← p(1)

Constraints are often used with choice rules to weed out "undesirable" stable models, for which the constraint is "violated."

{p(X): X=1..3}.
:- p(1).

$\emptyset$ $\{p(1), p(2)\}$
$\{p(1)\}$, $\{p(2), p(3)\}$
$\{p(2)\}$, $\{p(1), p(3)\}$
$\{p(3)\}$, $\{p(1), p(2), p(3)\}$

{p(X): X=1..3}.
:- not p(1).

$\emptyset$ $\{p(1), p(2)\}$
$\{p(1)\}$, $\{p(2), p(3)\}$
$\{p(2)\}$, $\{p(1), p(3)\}$
$\{p(3)\}$, $\{p(1), p(2), p(3)\}$

{p(X): X=1..3}.
:- not p(1), not p(2).

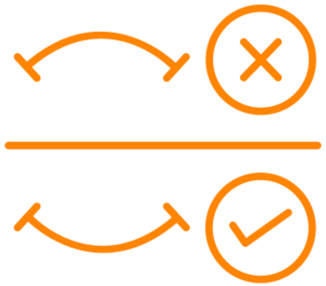
$\emptyset$ $\{p(1), p(2)\}$
$\{p(1)\}$, $\{p(2), p(3)\}$
$\{p(2)\}$, $\{p(1), p(3)\}$
$\{p(3)\}$, $\{p(1), p(2), p(3)\}$

Queens Puzzle

Generate-(Define)-Test

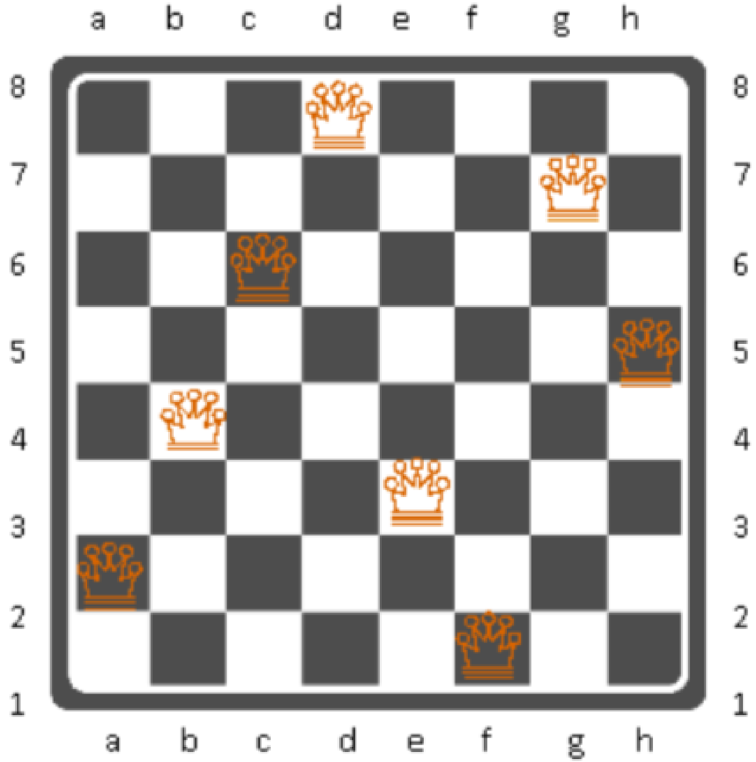
A way to organize rules in ASP

- GENERATE part: generates a “search space” – a set of potential solutions
- DEFINE part: defines new atoms in terms of other atoms
- TEST part: weed out the elements of the search space that do not represent solutions



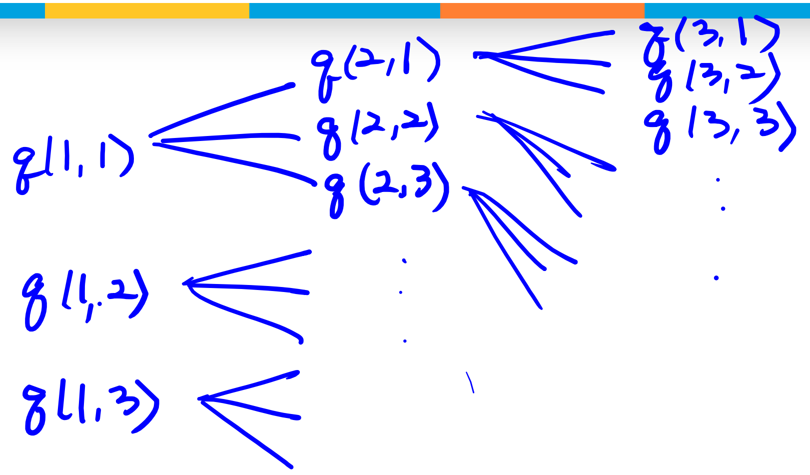
N-Queens Puzzle

No two queens can share the same row, column, or diagonal.



n	# sol
3	none
4	2
5	10
6	4
7	40
8	92

N-Queens in ASP




```
% Each row has exactly one queen
1 {queen(R,1..n)} 1 :- R=1..n.

%    or

                 =3                  =3
{queen(R,1..n)}=1 :- R=1..n.
```

⇒ {q(1, 1..3)} = 1 (R=1) ⇒ {q(1,1); q(1,2); q(1,3)} = 1

{q(2, 1..3)} = 1 (R=2) {q(2,1); q(2,2); q(2,3)} = 1

{q(3, 1..3)} = 1 (R=3) {q(3,1); q(3,2); q(3,3)} = 1

N-Queens in ASP

```
% Each row has exactly one queen
{queen(R,1..n)}=1 :- R=1..n.


% No two queens are on the same column
:- queen(R1,C), queen(R2,C), R1!=R2.


% No two queens are on the same diagonal
:- queen(R1,C1), queen(R2,C2), R1!=R2, |R1-R2|=|C1-C2|.
```

Finding One Solution for the 8-Queens Puzzle

```
$ clingo queens.lp -c n=8
clingo version 5.2.1
Reading from queens.lp
Solving...
Answer: 1
queen(4,1) queen(6,2) queen(8,3) queen(2,4) queen(7,5) queen(1,6)
queen(3,7) queen(5,8)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.004s
```

Finding All Solutions for the 8-Queens Puzzle

```
$ clingo queens.lp -c n=8 0
clingo version 5.2.1
Reading from queens.lp
Solving...
Answer: 1
queen(4,1) queen(6,2) queen(8,3) queen(2,4) queen(7,5) queen(1,6)
queen(3,7) queen(5,8)
Answer: 2
[[ truncated ]]
Answer: 92
queen(5,1) queen(1,2) queen(8,3) queen(4,4) queen(2,5) queen(7,6)
queen(3,7) queen(6,8)
SATISFIABLE

Models       : 92
Calls        : 1
Time         : 0.011s (Solving: 0.01s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.010s
```

Outline

1. Introduction
2. Intro to ASP
3. Stable Model Semantics
4. **Syntax and Semantics of LPMLN**
5. Relating LPMLN to Other Languages
6. Inference in LPMLN
7. Learning in LPMLN
8. Extension to Embrace Neural Networks

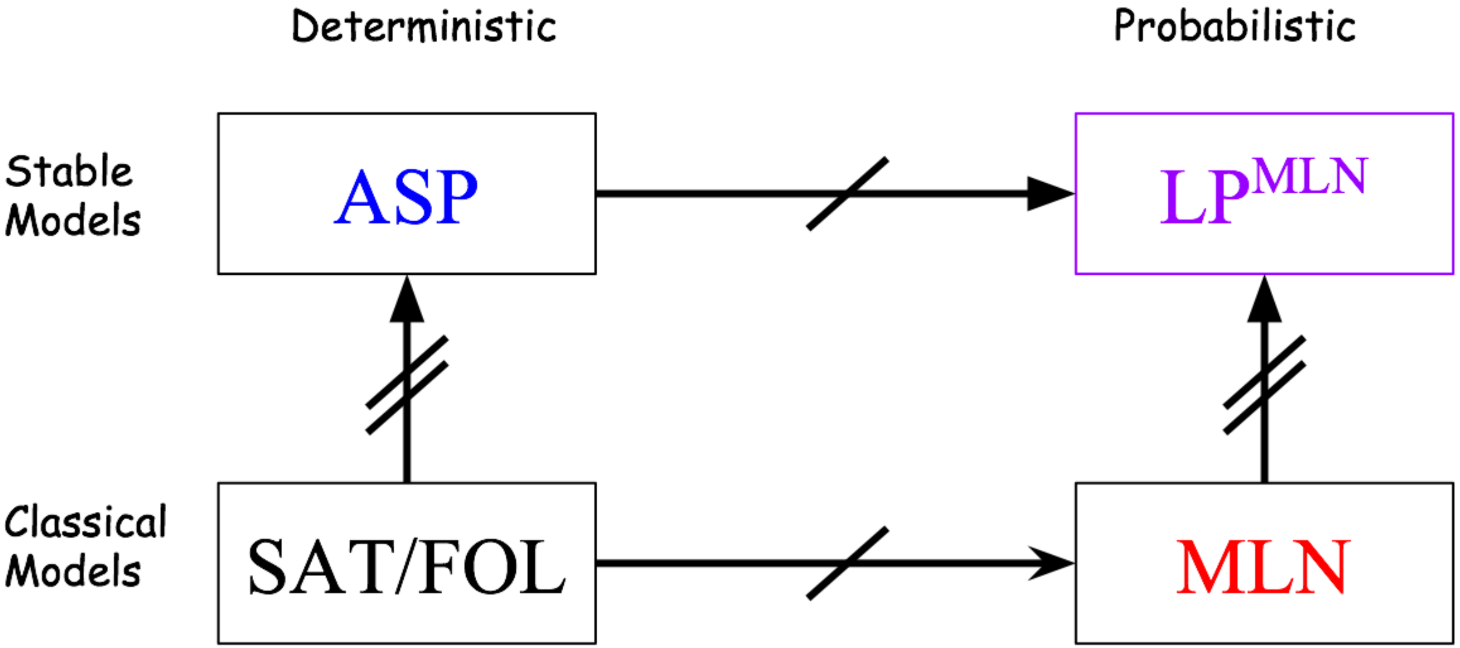
Language LP^{MLN}

- **A probabilistic extension of Answer Set Programs, following the log-linear models of Markov Logic**

- **It provides versatile methods to overcome the deterministic nature of the stable model semantics, such as:**
  - Resolving inconsistencies in answer set programs
  - Defining ranking/probability distribution over stable models
  - Applying methods from machine learning to compute KR formalisms

Language LP^{MLN}

- Overcomes the weakness of ASP in handling uncertainty.
- Overcomes the weakness of MLN in handling expressive commonsense reasoning.



Example

KB_1

```
bird(x) <- residentBird(x).
bird(x) <- migratoryBird(x).
<- residentBird(x), migratoryBird(x).
```

KB_2

```
residentBird(Jo).
```

KB_3

```
migratoryBird(Jo).
```

Example

KB_1

```
bird(x) <- residentBird(x).
bird(x) <- migratoryBird(x).
<- residentBird(x), migratoryBird(x).
```

Unsatisfiable!

no answer set, no information

KB_2

```
residentBird(Jo).
```

KB_3

```
migratoryBird(Jo).
```

LP^{MLN} (1 of 3)

- **Syntactically, it's a simple extension of answer set programs where each rule is prepended by weights**
  - infinite weight (∞) tells the rule expresses a definite knowledge
- **Each stable model gets weights from the rules that are true in the stable model**
  - a stable model does not have to satisfy all rules
  - the more rules true, the more likely the stable model

LP^{MLN} (2 of 3)

- **Adopting the log-linear models of MLN, language LP^{MLN} provides a simple and intuitive way to incorporate the concept of weights into the stable model semantics**
  - While MLN is an undirected approach, LP^{MLN} is a directed approach, where the directionality comes from the stable model semantics
- **Probabilistic answer set computation can be reduced to sampling and optimization problems**

Syntax of LP^{MLN}

- w: R where
  - w is a real number or α for denoting the infinite weight
  - R is an ASP rule
- Variables are understood in terms of grounding same as in MLN

Semantics of LP^{MLN}

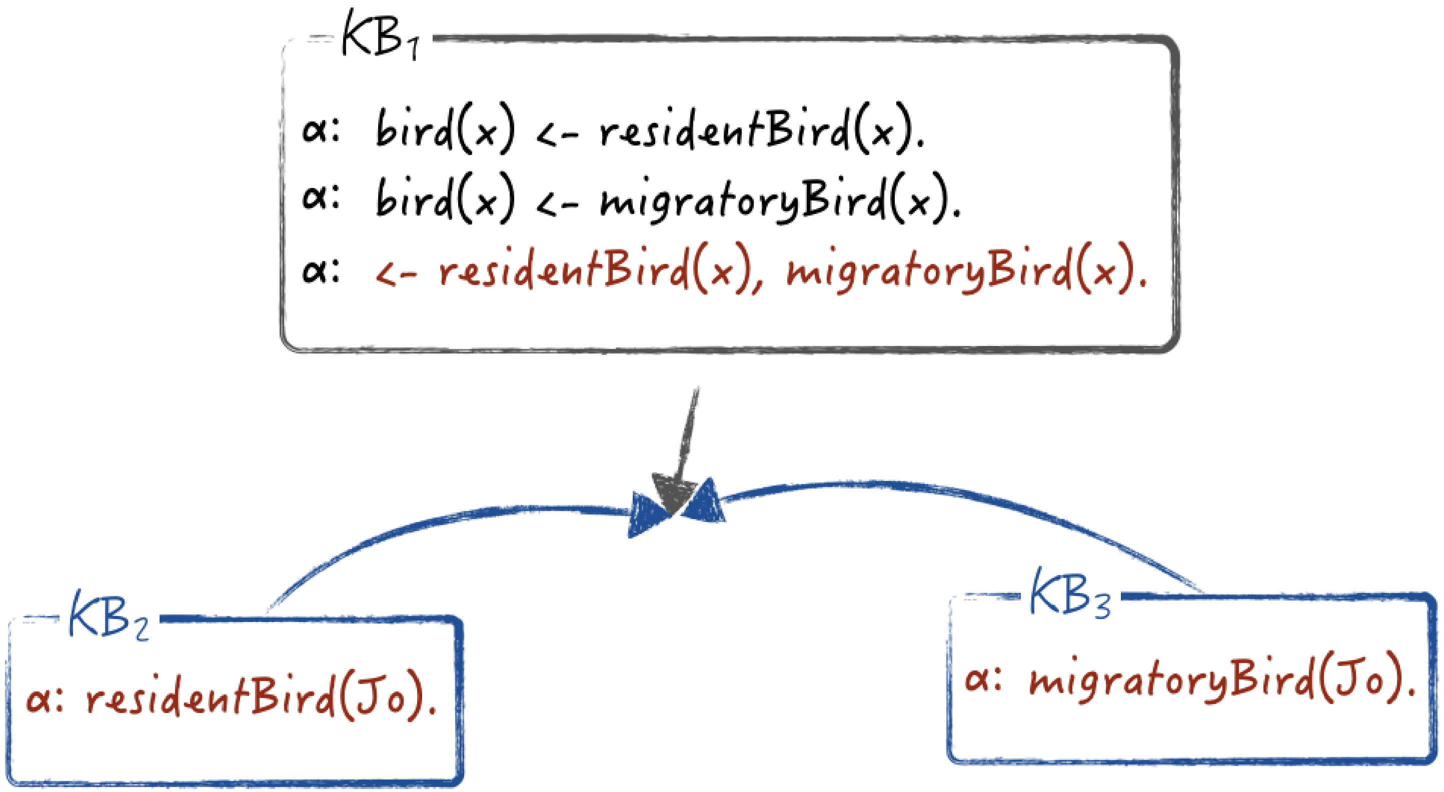
- Π_I denotes the set of rules $w : R$ in Π such that $I \models R$
- I is a **soft stable model** of Π if I is a (standard) stable model of Π_I
- The unnormalized weight of an interpretation I under Π is defined as

$$W_\Pi(I) = \begin{cases} exp\left(\sum\limits_{w:R \,\in\, \Pi_I} w\right) & \text{if } I \text{ is a soft stable model of } \Pi \\ 0 & \text{otherwise} \end{cases}$$

- The normalized weight (probability) of an interpretation I under Π, denotes $P_\Pi(I)$, is defined as

$$P_\Pi(I) = \lim_{\alpha \to \infty} \frac{W_\Pi(I)}{\sum_J W_\Pi(J)}.$$

Example 1


KB1
α: bird(x) <- residentBird(x).
α: bird(x) <- migratoryBird(x).
α: <- residentBird(x), migratoryBird(x).
KB2
α: residentBird(Jo).
KB3
α: migratoryBird(Jo).

Example 1

$$
\begin{array}{llll}
KB_1 & \alpha: & Bird(x) \leftarrow ResidentBird(x) & (r1)\\
 & \alpha: & Bird(x) \leftarrow MigratoryBird(x) & (r2)\\
 & \alpha: & \leftarrow ResidentBird(x), MigratoryBird(x) & (r3)\\
KB_2 & \alpha: & ResidentBird(Jo) & (r4)\\
KB_3 & \alpha: & MigratoryBird(Jo) & (r5)
\end{array}
$$

I	Π_I	$W_\Pi(I)$	$P_\Pi(I)$
$\emptyset$	$\{r_1, r_2, r_3\}$	$e^{3\alpha}$	0
$\{R(Jo)\}$	$\{r_2, r_3, r_4\}$	$e^{3\alpha}$	0
$\{M(Jo)\}$	$\{r_1, r_3, r_5\}$	$e^{3\alpha}$	0
$\{B(Jo)\}$	$\{r_1, r_2, r_3\}$	0	0
$\{R(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_4\}$	$e^{4\alpha}$	1/3
$\{M(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_5\}$	$e^{4\alpha}$	1/3
$\{R(Jo), M(Jo)\}$	$\{r_4, r_5\}$	$e^{2\alpha}$	0
$\{R(Jo), M(Jo), B(Jo)\}$	$\{r_1, r_2, r_4, r_5\}$	$e^{4\alpha}$	1/3

P(R(Jo)) =

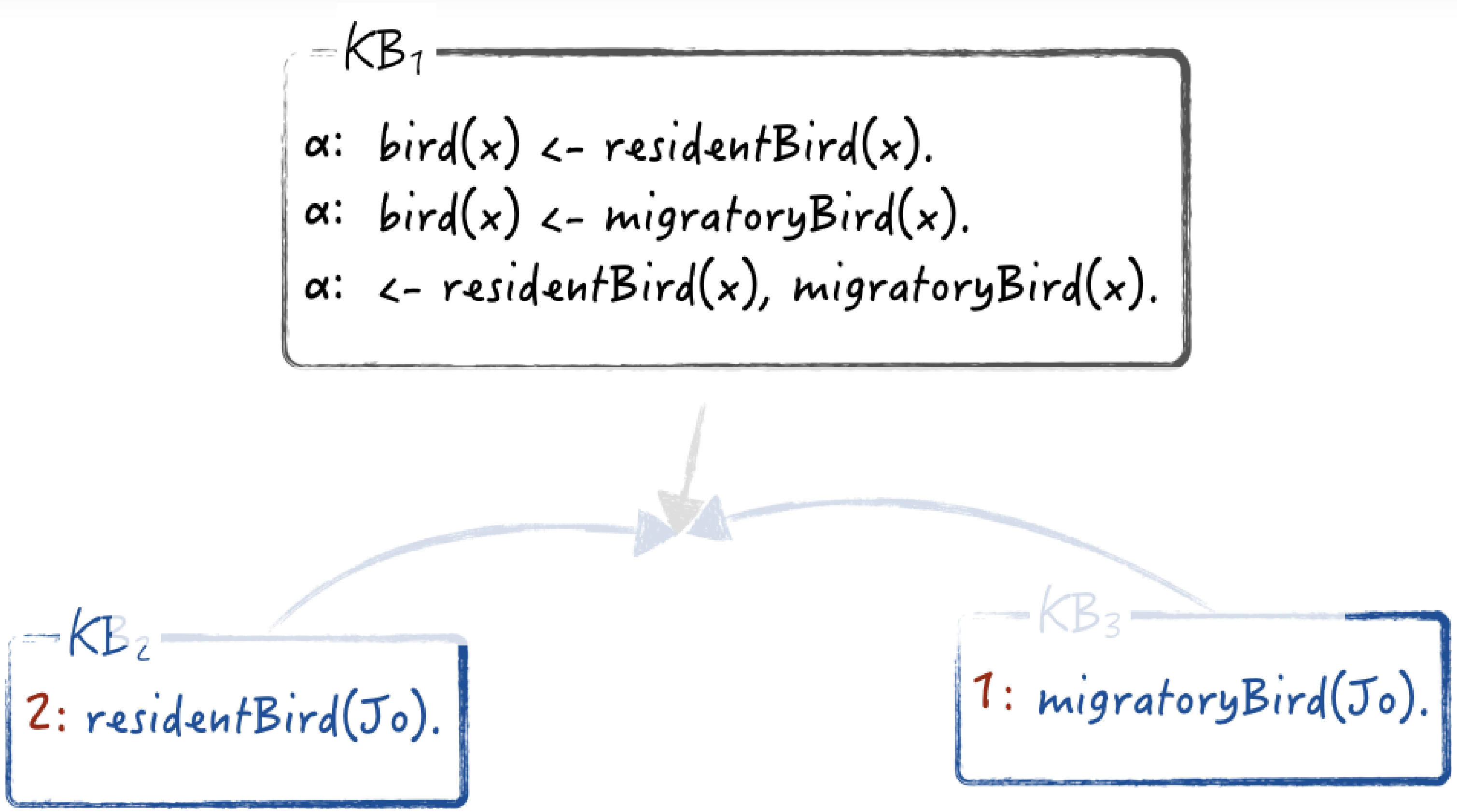
P(B(Jo)) =

P(B(Jo) | R(Jo)) =

P(R(Jo) | B(Jo)) =

P(R(Jo) & M(Jo)) =

Example 2


KB1
α: bird(x) <- residentBird(x).
α: bird(x) <- migratoryBird(x).
α: <- residentBird(x), migratoryBird(x).
KB2
2: residentBird(Jo).
KB3
1: migratoryBird(Jo).

Example 2

$$\begin{array}{llll} KB_1 & \alpha : & Bird(x) \leftarrow ResidentBird(x) & (r1) \\ & \alpha : & Bird(x) \leftarrow MigratoryBird(x) & (r2) \\ & \alpha : & \leftarrow ResidentBird(x), MigratoryBird(x) & (r3) \end{array}$$

$$KB_2' \quad 2 : \quad ResidentBird(Jo) \qquad (r4')$$

$$KB_3' \quad 1 : \quad MigratoryBird(Jo) \qquad (r5')$$

I	Π_I	$W_\Pi(I)$	$P_\Pi(I)$
$\emptyset$	$\{r_1, r_2, r_3\}$	$e^{3\alpha}$	$\frac{e^0}{e^2+e^1+e^0}$ = 0.09
$\{R(Jo)\}$	$\{r_2, r_3, r_4'\}$	$e^{2\alpha+2}$	0
$\{M(Jo)\}$	$\{r_1, r_3, r_5'\}$	$e^{2\alpha+1}$	0
$\{B(Jo)\}$	$\{r_1, r_2, r_3\}$	0	0
$\{R(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_4'\}$	$e^{3\alpha+2}$	$\frac{e^2}{e^2+e^1+e^0}$ = 0.67
$\{M(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_5'\}$	$e^{3\alpha+1}$	$\frac{e^1}{e^2+e^1+e^0}$ = 0.24
$\{R(Jo), M(Jo)\}$	$\{r_4', r_5'\}$	e^3	0
$\{R(Jo), M(Jo), B(Jo)\}$	$\{r_1, r_2, r_4', r_5'\}$	$e^{2\alpha+3}$	0

$P(R(Jo)) = 0.67$

$P(M(Jo)) = 0.24$

$P(\neg R(Jo) \wedge \neg M(Jo)) = 0.09$

$P(B(Jo)) = 0.67 + 0.24 = 0.91$

$P(R(Jo) \mid B(Jo)) = \frac{0.67}{0.67+0.24}$

$= 0.74.$

Reward-Based Weight

REWARD-BASED WEIGHT

$$W_\Pi(I) = exp(\sum_{w:R\in\Pi, I\models R} w)$$

Probability

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_J W_\Pi(J)}.$$

Penalty-Based Weight

PENALTY-BASED WEIGHT

$$W_{\Pi}^{pnt}(I) = exp(-\sum_{w:R\in\Pi, I\not\models R} w)$$

Probability

$$P_{\Pi}^{pnt}(I) = \lim_{\alpha\to\infty} \frac{W_{\Pi}^{pnt}(I)}{\sum_{J} W_{\Pi}^{pnt}(J)}$$

Example (Penalty-based)

$$
\begin{array}{llll}
KB_1 & \alpha: & Bird(x) \leftarrow ResidentBird(x) & (r1) \\
 & \alpha: & Bird(x) \leftarrow MigratoryBird(x) & (r2) \\
 & \alpha: & \leftarrow ResidentBird(x), MigratoryBird(x) & (r3)
\end{array}
$$

$$KB_2' \quad 2: \quad ResidentBird(Jo) \qquad (r4')$$

$$KB_3' \quad 1: \quad MigratoryBird(Jo) \qquad (r5')$$

I	Π_I	$W_{\Pi}^{\text{pnt}}(I)$	$P_{\Pi}^{\text{pnt}}(I)$	
$\emptyset$	$\{r_1, r_2, r_3\}$	e^{-3}	$\frac{e^{-3}}{e^{-1}+e^{-2}+e^{-3}}$	≈ 0.09
$\{R(Jo)\}$	$\{r_2, r_3, r_4'\}$	$e^{-\alpha-1}$	0	
$\{M(Jo)\}$	$\{r_1, r_3, r_5'\}$	$e^{-\alpha-2}$	0	
$\{B(Jo)\}$	$\{r_1, r_2, r_3\}$	0	0	
$\{R(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_4'\}$	e^{-1}	$\frac{e^{-1}}{e^{-1}+e^{-2}+e^{-3}}$	$= 0.67$
$\{M(Jo), B(Jo)\}$	$\{r_1, r_2, r_3, r_5'\}$	e^{-2}	$\frac{e^{-2}}{e^{-1}+e^{-2}+e^{-3}}$	$= 0.24$
$\{R(Jo), M(Jo)\}$	$\{r_4', r_5'\}$	$e^{-3\alpha}$	0	
$\{R(Jo), M(Jo), B(Jo)\}$	$\{r_1, r_2, r_4', r_5'\}$	$e^{-\alpha}$	0	

Reward vs. Penalty based Weights

Theorem. For any LPMLN program Π and any interpretation I,

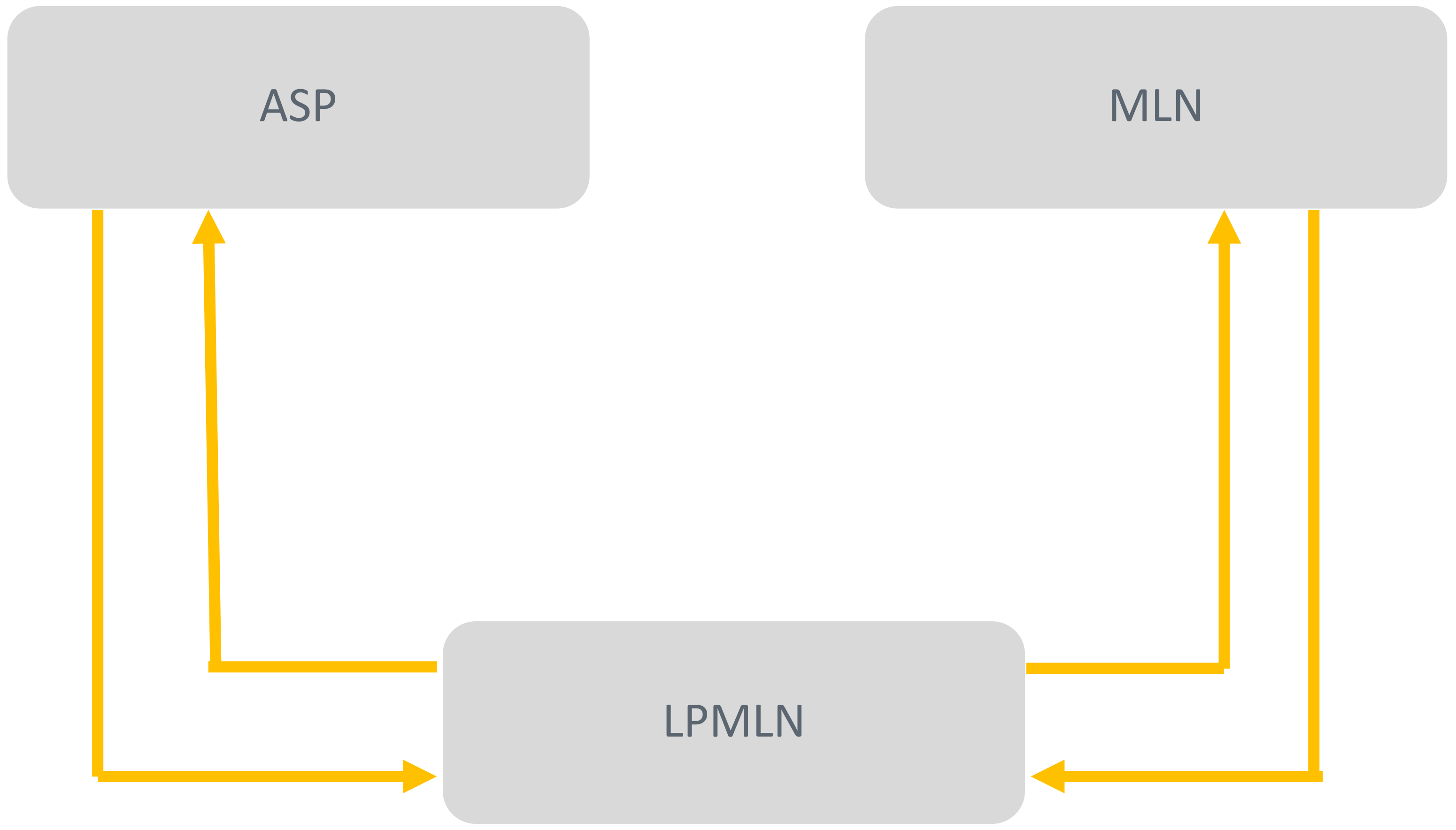
$$W_{\Pi}(I) \propto W_{\Pi}^{pnt}(I)$$

$$P_{\Pi}(I) = P_{\Pi}^{pnt}(I)$$

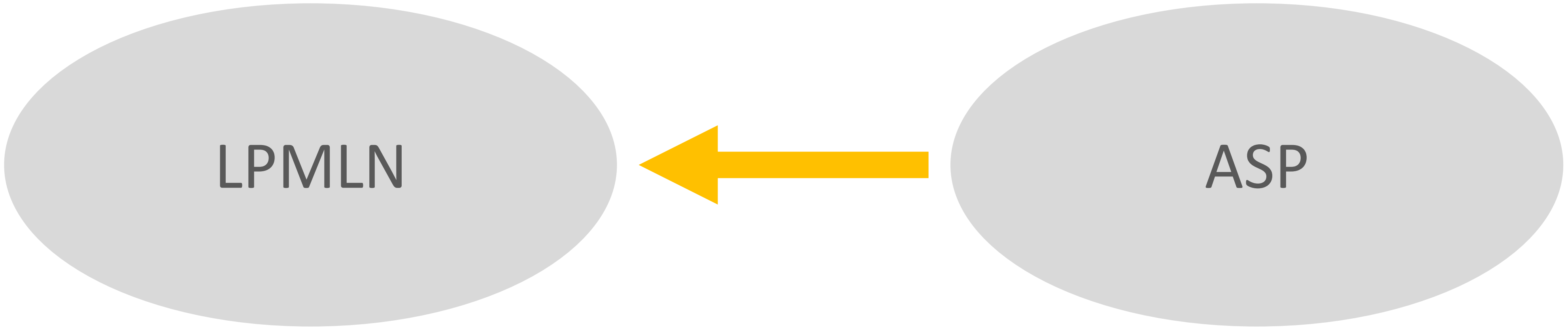
Outline

1. Introduction
2. Intro to ASP
3. Stable Model Semantics
4. Syntax and Semantics of LPMLN
5. **Relating LPMLN to Other Languages**
6. Inference in LPMLN
7. Learning in LPMLN
8. Extension to Embrace Neural Networks

LPMLN vs. ASP vs. MLN



From ASP to LP^{MLN}



ASP as a Special Case of LP$^{\text{MLN}}$

- Any answer set program Π can be viewed as a special case of an LP$^{\text{MLN}}$ program P_Π by assigning the infinite weight to each rule

Π		P_Π	
	$p \leftarrow \text{not } q$		$\alpha:\ p \leftarrow \text{not } q$
	$q \leftarrow \text{not } p$		$\alpha:\ q \leftarrow \text{not } p$

$\{p\}$
$\{q\}$

P_π

I	$W(I)$	$P(I)$
ϕ	e^0	0
$\{p\}$	$e^{2\alpha}$	$\frac{1}{2}$
$\{q\}$	$e^{2\alpha}$	$\frac{1}{2}$
$\{p, q\}$	0	0

- **Theorem:** For any answer set program Π, the (deterministic) stable models of Π are exactly the (probabilistic) stable models of LP$^{\text{MLN}}$ program P_Π whose weight is $e^{k\alpha}$, where k is the number of all ground rules in Π

Example

- If Π has at least one (deterministic) stable model, then all (probabilistic) stable models of P_Π have the same probability, and are thus the stable models of Π as well

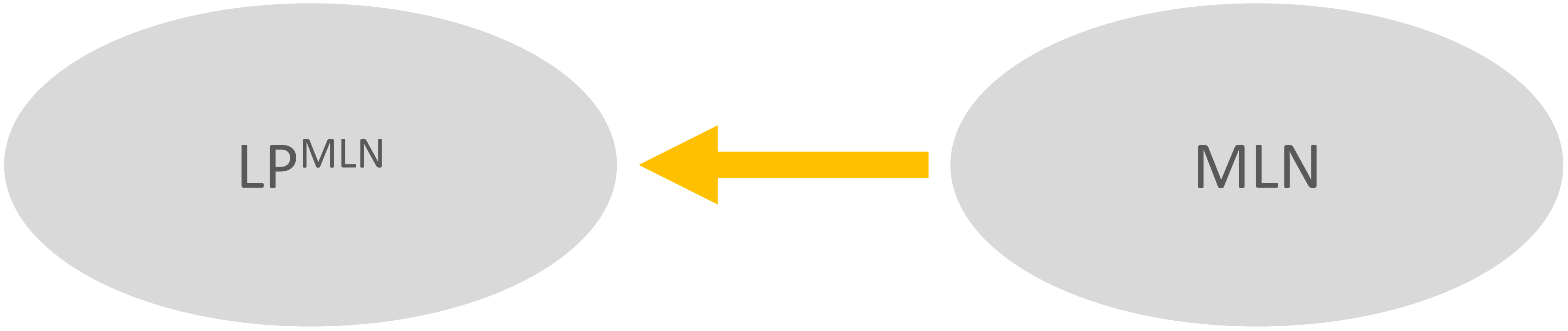
- **Q**: What if Π has no stable models?

$$
\begin{array}{ll}
\Pi & \text{Bird(Jo)} \leftarrow \text{ResidentBird(Jo)} \\
& \text{Bird(Jo)} \leftarrow \text{MigratoryBird(Jo)} \\
& \bot \leftarrow \text{ResidentBird(Jo), MigratoryBird(Jo)} \\
& \text{ResidentBird(Jo)} \\
& \text{MigratoryBird(Jo)}
\end{array}
$$

$$
\begin{array}{ll}
P_\Pi & \alpha:\ \text{Bird(Jo)} \leftarrow \text{ResidentBird(Jo)} \\
& \alpha:\ \text{Bird(Jo)} \leftarrow \text{MigratoryBird(Jo)} \\
& \alpha:\ \bot \leftarrow \text{ResidentBird(Jo), MigratoryBird(Jo)} \\
& \alpha:\ \text{ResidentBird(Jo)} \\
& \alpha:\ \text{MigratoryBird(Jo)}
\end{array}
$$

- **Q**: What are the stable models P_Π? {B(Jo), R(Jo)}, {B(Jo), M(Jo)}, {B(Jo)}

From MLN to LP^{MLN}



Outline

1. Introduction
2. Intro to ASP
3. Stable Model Semantics
4. Syntax and Semantics of LPMLN
5. Relating LPMLN to Other Languages
6. **Inference in LPMLN**
7. Learning in LPMLN
8. Extension to Embrace Neural Networks

Weak Constraints (1 of 3)

- A weak constraint has the form

  :~ F. [Weight @ Level]

  :~

- Weight is an integer and Level is a nonnegative integer

Weak Constraints (2 of 3)

- Let Π be a program $\Pi_1 \cup \Pi_2$, where Π_1 is a usual ASP program and Π_2 is a set of weak constraints.

- We call I a stable model of Π if it is a stable model of Π_1 .

- For every stable model I of Π and any nonnegative integer l, the penalty of I at level L, denoted by $Penalty_{\Pi}(I, L)$, is defined as

$$\sum_{\substack{:\sim\ F[w@l] \in \Pi_2, \\ I \models F}} w.$$

- ex:

```
{p; q}.
:~ p.   [10@0]
:~ q.   [5@1]
```

Pen({p}, 0) = 10

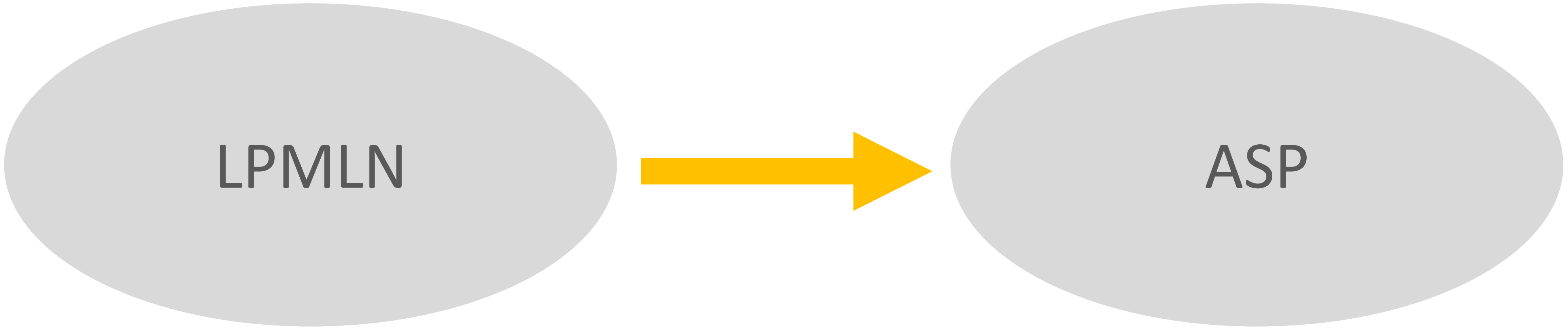
Pen({p}, 1) = 0

Weak Constraints (3 of 3)

- For any two stable models I and I' of Π, we say I is **dominated** by I' if
  - there is some level L such that $Penalty_\Pi(I', L) < Penalty_\Pi(I, L)$ and
  - for all integers $K > L$, $Penalty_\Pi(I', K) = Penalty_\Pi(I, K)$

- A stable model of Π is called **optimal** if it is not dominated by another stable model of Π

From LPMLN to ASP: Weak Constraints



In clingo

```
% test

{p;q}.
:~ p. [10@0]
:~ q. [5@1]
```

```
$ clingo test

Answer: 1
Optimization: 0 0
OPTIMUM FOUND


Models        : 1
  Optimum     : yes
Optimization : 0 0
```

```
$ clingo test --opt-mode=enum 0
Solving...
Answer: 1


Optimization: 0 0
Answer: 2
q
Optimization: 5 0
Answer: 3
p
Optimization: 0 10
Answer: 4
p q
Optimization: 5 10
OPTIMUM FOUND


Models            : 4
```

Translation lpmln2asp

Soft Rules:

$$w_i : Head_i \leftarrow Body_i$$

$$unsat(i) \leftarrow Bodyi, not\ Head_i$$
$$Head_i \leftarrow Body_i, not\ unsat(i)$$
$$:\sim unsat(i)\ [w_i@0]$$

Hard Rules:

$$\alpha : Head_i \leftarrow Body_i$$

$$unsat(\cancel{I})i \leftarrow Bodyi, not\ Head_i$$
$$Head_i \leftarrow Body_i, not\ unsat(i)$$
$$:\sim unsat(i)\ [1@1]$$

Theorem: For any LP^{MLN} program Π, the most probable stable models of Π are precisely the optimal stable models of $\mathrm{lpmln2asp}(\Pi)$.

Example

Theorem: For any LP$^{\mathrm{MLN}}$ program Π, the most probable stable models of Π are precisely the optimal stable models of lpmln2asp(Π).

LP$^{\mathrm{MLN}}$ program

$$\alpha : p \qquad (r_1)$$

$$10 : q \leftarrow p \qquad (r_2)$$

$$-20 : q \qquad (r_3)$$

Q: What is the most probable stable model?

I	W(I)
{}	e^{10}
{p}	e^{α}
{q}	e^{10-20}
{p,q}	$e^{\alpha+10-20}$

Example

LP^{MLN} program

$$\alpha : p \qquad (r_1)$$

$$10 : q \leftarrow p \qquad (r_2)$$

$$-20 : q \qquad (r_3)$$

Clingo program

$$unsat(1) \ :- \ not \ \ p.$$
$$p :- not \ unsat(1).$$
$$:\sim unsat(1). \ [1@1]$$

$$unsat(2) \ :- p, not \ \ q.$$
$$q :- p, not \ unsat(2).$$
$$:\sim unsat(2) \ [10@0]$$

$$unsat(3) \ :- not \ \ q.$$
$$q :- \ not \ unsat(3).$$
$$:\sim unsat(3). \ [-20@0]$$

Clingo Output

Solving…
Answer: 1
p unsat(2) unsat(3)
Optimization: 0 -10

OPTIMUM FOUND

% The number in blue is the penalty at level 1.
% The number in red is the penalty at level 0.

Implementation of LPMLN2ASP

The most probable stable models correspond to optimal stable models

Weight of stable models can be calculated with

$$W_{\Pi}^{\text{pnt}}(I) = exp\Bigg(- \sum_{\mathtt{unsat}(i, w_i, \mathbf{c}) \in \phi(I)} w_i \Bigg).$$

The corresponding stable model of the corresponding ASP program lpmln2asp(Π)

Marginal probability of an atom a

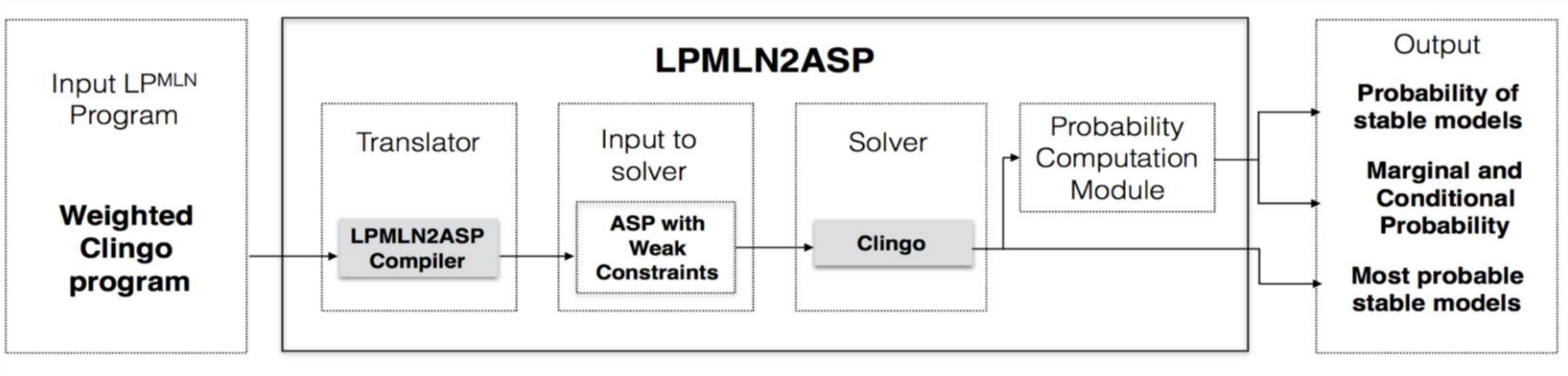
$$P_\Pi(a) = \sum_{J \models a} P_\Pi(J)$$

Conditional probability of an atom a given evidence E

$$P_\Pi(a \mid E) = \sum_{J \models a} P_{\Pi \cup E}(J)$$

(E is encoded as a set of ASP constraints)

System Architecture



- http://github.com/azreasoners/lpmln
- lpmln2asp can compute MAP inference, marginal and conditional probability
- MAP inference is directly computed by clingo
- Probability calculations are computed by a probability computation module

Input Language of lpmln-infer

- The input language resembles the input language of clingo
- Hard rules are encoded exactly the same as clingo rules
- Soft rules are clingo rules with weight prepended

```
% File: bird.lpmln
bird(X) :- residentbird(X).
bird(X) :- migratorybird(X).
:- residentbird(X), migratorybird(X).
2 residentbird(jo).
1 migratorybird(jo).
```

Example: Finding Most Probable Stable Models

```
% bird.lpmln
bird(X) :- residentbird(X).
bird(X) :- migratorybird(X).
:- residentbird(X), migratorybird(X).
2 residentbird(jo).
1 migratorybird(jo).
```

```
$ lpmln-infer bird.lpmln
```

Answer: 1
unsat(5,"1") unsat(4,"2")
Optimization: 3000
Answer: 2
unsat(5,"1") residentbird(jo) bird(jo)
Optimization: 1000
OPTIMUM FOUND

Example: Probabilities of All Stable Models

```
% bird.lpmln
bird(X) :- residentbird(X).
bird(X) :- migratorybird(X).
:- residentbird(X), migratorybird(X).
2 residentbird(jo).
1 migratorybird(jo).
```

```
$ lpmln-infer bird.lpmln -all
```

```
[unsat(5,"1"), unsat(4,"2")] :  0.09003057317038046
[residentbird(jo), bird(jo), unsat(5,"1")] :  0.6652409557748219
[bird(jo), migratorybird(jo), unsat(4,"2")] :  0.24472847105479767
```

Example: Marginal Probability of Query

```
% bird.lpmln
bird(X) :- residentbird(X).
bird(X) :- migratorybird(X).
:- residentbird(X), migratorybird(X).
2 residentbird(jo).
1 migratorybird(jo).
```

query atoms

```
$ lpmln-infer bird.lpmln -q residentbird
```

```
residentbird(jo) 0.665240955775
```

- **The command is same as**

  `$ lpmln-infer bird.lpmln -q residentbird -exact`

- **Alternatively one can use sampling-based inference**

  `$ lpmln-infer bird.lpmln -q residentbird -mcasp`

Example: Conditional Probability of Query

P(residentbird(jo) | bird(jo))

```
% bird.lpmln
bird(X) :- residentbird(X).
bird(X) :- migratorybird(X).
:- residentbird(X), migratorybird(X).
2 residentbird(jo).
1 migratorybird(jo).
```

```
% bird-evid.db
:- not bird(jo).
```

```
$ lpmln-infer bird.lpmln -e bird-evid.db -q residentbird
```

evidence file: set of asp constraints

residentbird(jo) : 0.7310585786300049

Example: Debugging in ASP

```
% bird1.lpmln
bird(X) :- residentbird(X).
bird(X) :- migratorybird(X).
:- residentbird(X), migratorybird(X).
residentbird(jo).
migratorybird(jo).
```

translate hard rules

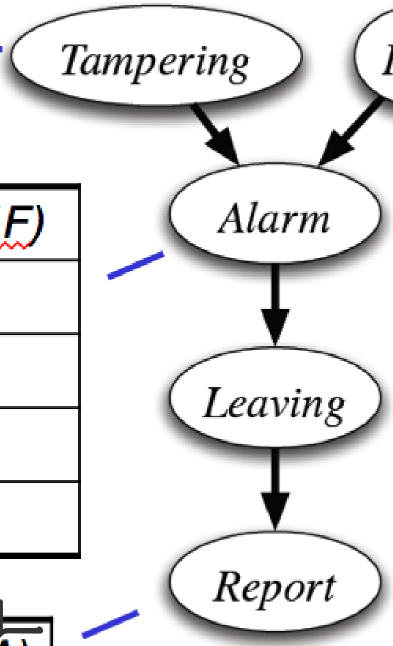
```
$ lpmln-infer bird1.lpmln -all -hard
```

✓ [bird(jo), migratorybird(jo), unsat(4,"a")] : 0.3333333333333333
[residentbird(jo), bird(jo), unsat(3,"a",jo), migratorybird(jo)] : 0.3333333333333333
[residentbird(jo), bird(jo), unsat(5,"a")] : 0.3333333333333333

Representing Bayesian networks in LP^{MLN}

Recall: Example

P(Tampering=t)
0.02



P(Fire=t)
0.01

Tampering T	Fire F	P(Alarm=t|T,F)
t	t	0.5
t	f	0.85
f	t	0.99
f	f	0.0001

Fire F	P(Smoke=t |F)
t	0.9
f	0.01

Alarm	P(Leaving=t|A)
t	0.88
f	0.001

Leaving	P(Report=t|A)
t	0.75
f	0.01

Representing Bayesian Networks in LP^{MLN}

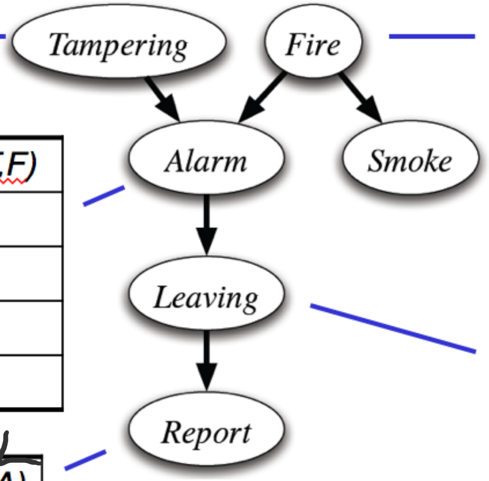
Encode CPT using auxiliary atoms

$\ln\left(\frac{p}{1-p}\right)$: pf

$$p(pf=t) = \frac{e^{\ln\left(\frac{p}{1-p}\right)}}{e^{\ln\left(\frac{p}{1-p}\right)} + e^{0}} = \frac{\frac{p}{1-p}}{\frac{p}{1-p}+1} = p$$

$$p(pf=f) = \frac{1}{\frac{p}{1-p}+1} = 1-p$$

```
@log(0.02/0.98) pf(t).
@log(0.01/0.99) pf(f).
@log(0.5/0.5)   pf(a,t1f1).
@log(0.85/0.15)  pf(a,t1f0).
@log(0.99/0.01)  pf(a,t0f1).
@log(0.0001/0.9999) pf(a,t0f0).
@log(0.9/0.1)  pf(s,f1).
@log(0.01/0.99) pf(s,f0).
@log(0.88/0.12)   pf(l,a1).
@log(0.001/0.999) pf(l,a0).
@log(0.75/0.25) pf(r,l1).
@log(0.01/0.99) pf(r,l0).
```



P(Tampering=t)
0.02

P(Fire=t)
0.01

Tampering T	Fire F	P(Alarm=t|T,F)
t	t	0.5
t	f	0.85
f	t	0.99
f	f	0.0001

Fire F	P(Smoke=t |F)
t	0.9
f	0.01

Alarm	P(Leaving=t|A)
t	0.88
f	0.001

Leaving	P(Report=t|A)
t	0.75
f	0.01

Representing Bayesian Networks in LP^{MLN}

Encode DAG in rules:

tampering :- pf(t).

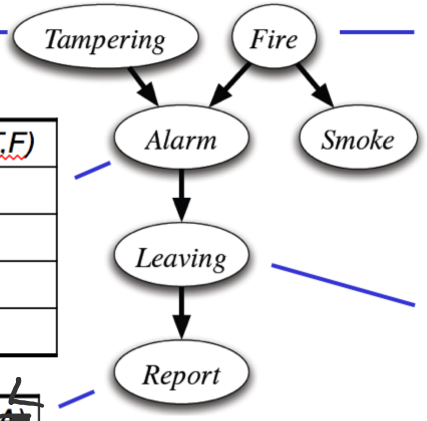
fire :- pf(f).

alarm :- tampering, fire, pf(a,t1f1).
alarm :- tampering, not fire, pf(a,t1f0).
alarm :- not tampering, fire, pf(a,t0f1).
alarm :- not tampering, not fire, pf(a,t0f0).

smoke :- fire, pf(s,f1).
smoke :- not fire, pf(s,f0).

P(Tampering=t)
0.02

P(Fire=t)
0.01



Tampering T	Fire F	P(Alarm=t|T,F)
t	t	0.5
t	f	0.85
f	t	0.99
f	f	0.0001

Fire F	P(Smoke=t |F)
t	0.9
f	0.01

Alarm	P(Leaving=t|A)
t	0.88
f	0.001

Leaving	P(Report=t|A)
t	0.75
f	0.01

leaving :- alarm, pf(l,a1).
leaving :- not alarm, pf (l,a0).

report :- leaving, pf(r,l1).
report :- not leaving, pf(r,l0).

Representing Bayesian Networks in LP^{MLN}

```
// fire-bayes.lpmln

@log(0.02/0.98) pf(t).
@log(0.01/0.99) pf(f).
@log(0.5/0.5)   pf(a,t1f1).
@log(0.85/0.15)  pf(a,t1f0).
@log(0.99/0.01)  pf(a,t0f1).
@log(0.0001/0.9999) pf(a,t0f0).

@log(0.9/0.1)  pf(s,f1).
@log(0.01/0.99) pf(s,f0).

@log(0.88/0.12)   pf(l,a1).
@log(0.001/0.999) pf(l,a0).

@log(0.75/0.25) pf(r,l1).
@log(0.01/0.99) pf(r,l0).
```

```
tampering :- pf(t).

fire :- pf(f).

alarm :- tampering, fire, pf(a,t1f1).
alarm :- tampering, not fire, pf(a,t1f0).
alarm :- not tampering, fire, pf(a,t0f1).
alarm :- not tampering, not fire, pf(a,t0f0).

smoke :- fire, pf(s,f1).
smoke :- not fire, pf(s,f0).

leaving :- alarm, pf(l,a1).
leaving :- not alarm, pf (l,a0).

report :- leaving, pf(r,l1).
report :- not leaving, pf(r,l0).
```

Example Run

- **To compute P(fire | alarm, ¬tampering)**
  - Write into fire-evid.db contains

```
:- not alarm.
:- tampering.
```

  - Call

$ lpmln-infer fire-bayes.lpmln –e fire-evid.db –q fire

Diagnostic Inference

Compute the probability of the cause given the effect

To compute P(fire = t | leaving = t), the user can invoke

```
$ lpmln-infer fire-bayes.lpmln -e fire-evid.db -q fire
```

where fire-evid.db contains the line

```
:- not leaving.
```

This outputs

```
fire : 0.35215453804538244
```

Predictive Inference

Compute the probability of effect given the cause.

To compute P(leaving = t | fire = t), the user can invoke

```
$ lpmln-infer fire-bayes.lpmln -e fire-evid.db -q leaving
```

where fire-evid.db contains the line

```
:- not fire.
```

This outputs

```
leaving 0.862603541626
```

Mixed Inference

Combine predictive and diagnostic inference.

To compute P(alarm = t | fire = f, leaving = t), the user can invoke

```
$ lpmln-infer fire-bayes.lpmln -e fire-evid.db -q alarm
```

where fire-evid.db contains two lines

```
:- fire.
:- not leaving.
```

This outputs

```
alarm : 0.9386803111482813
```

Intercausal inference (Explaining Away)

Reasons about the mutual causes (effects) of a common effect

Knowing that there was tampering explains away alarm, and hence affecting the probability of fire.

P(fire = t | alarm = t, tampering = t) using lpmln-infer outputs

```
fire : 0.005906674542232707
```

P(fire = t | alarm = t, tampering = f) using lpmln-infer outputs

```
fire : 0.9900990099009899
```

Representing Probabilistic Graph Problems

Example: Probabilistic Path (1 of 2)

- **ASP encoding of graph problems can be easily turned into probabilistic extensions. E.g.,**
  - "given that there is a path between two nodes, what is the most likely graph?": MAP inference
  - "given two nodes, what is the probability that there exists a path between them?": probabilistic query

- **We put ln(p/(1-p)) as the weight of the rule edge(X, Y)**

```
@log(0.3/0.7)  edge(0, 1).
@log(0.2/0.8)  edge(1, 2).
     ...
```

Example: Probabilistic Path (2 of 2)

- **We represent path relation as hard rules:**

```
path(X,Y) :- edge(X, Y).
path(X,Y) :- path(X, Z), path(Z, Y), Y != Z.
```

- **Probabilistic Traveling Salesman:** ”Given a graph with uncertain edges, what is the probability that there is a Hamiltonian circuit? “

Example: Network Connectivity (1 of 3)

```
node(1..4).

@log(0.8/0.2)  fail(2).
@log(0.5/0.5)  fail(3).
@log(0.2/0.8)  fail(4).

edge(1,2).   edge(2,4).     edge(1,3).     edge(3,4).
edge(2,3).

connected(X,Y) :- edge(X, Y), not fail(X), not fail(Y).
connected(X,Y) :- connected(X,Z), connected(Z,Y).
```

Example: Network Connectivity (2 of 3)

Q: What is the probability that 1 and 4 are connected?

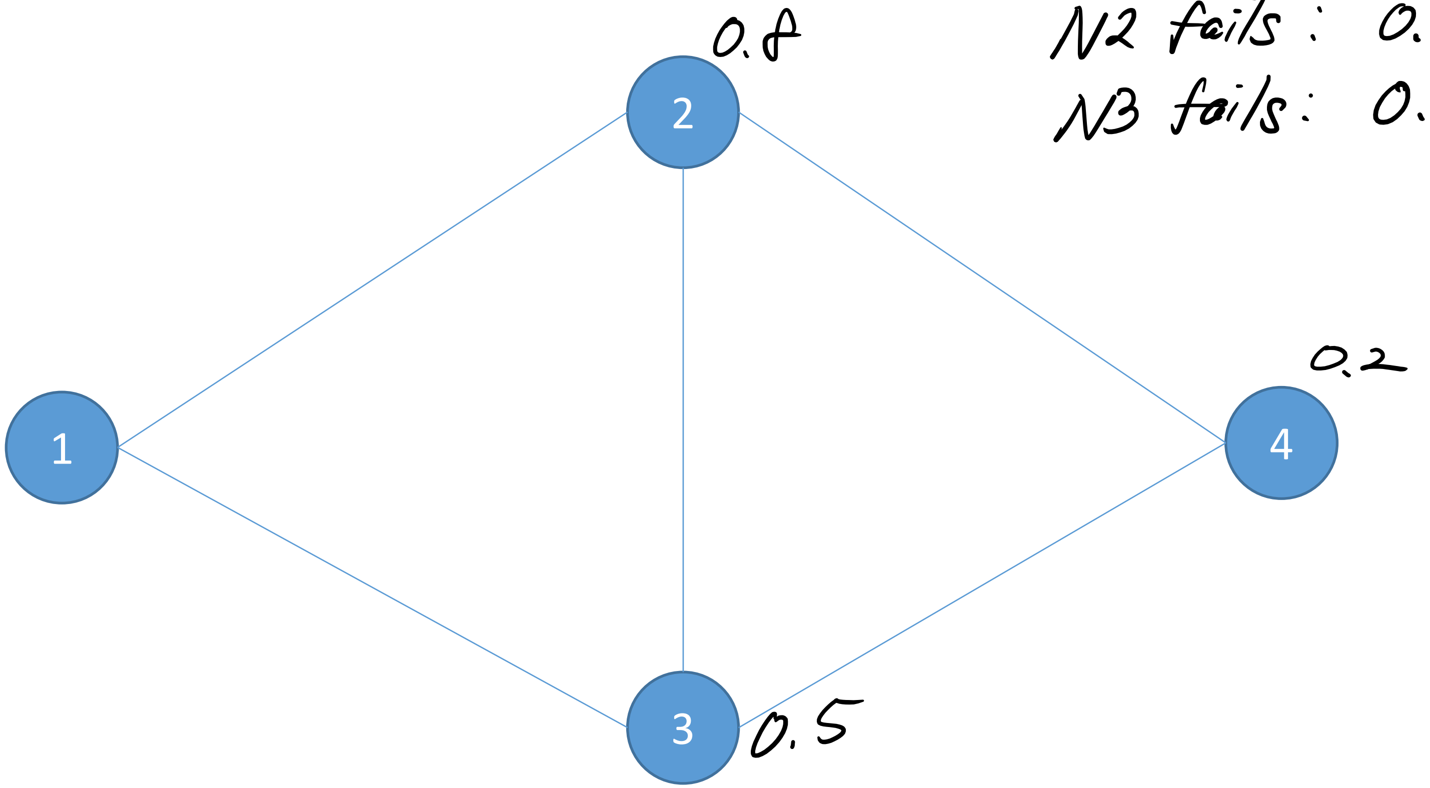
A. 0.32 **B. 0.4** **C. 0.16** **D. 0.6**

All okay : $0.2 \times 0.5 \times 0.8 = 0.08$

N2 fails : $0.8 \times 0.5 \times 0.8 = 0.32$

N3 fails : $0.2 \times 0.5 \times 0.8 = 0.08$

0.48



Example: Network Connectivity (3 of 3)

```
$ lpmln-infer networks.lpmln -q connected

connected(1, 2) : 0.19999999999999998
connected(2, 4) : 0.16
connected(1, 3) : 0.5
connected(3, 4) : 0.4
connected(2, 3) : 0.1
connected(1, 4) : 0.48000000000000004
```

Example: Virus (1 of 2)

person(a;b;c;d;e;f;g).

1.5 has_disease(X) :- carries_virus(X).

1.1 carries_virus(Y) :- contact(X, Y), carries_virus(X).

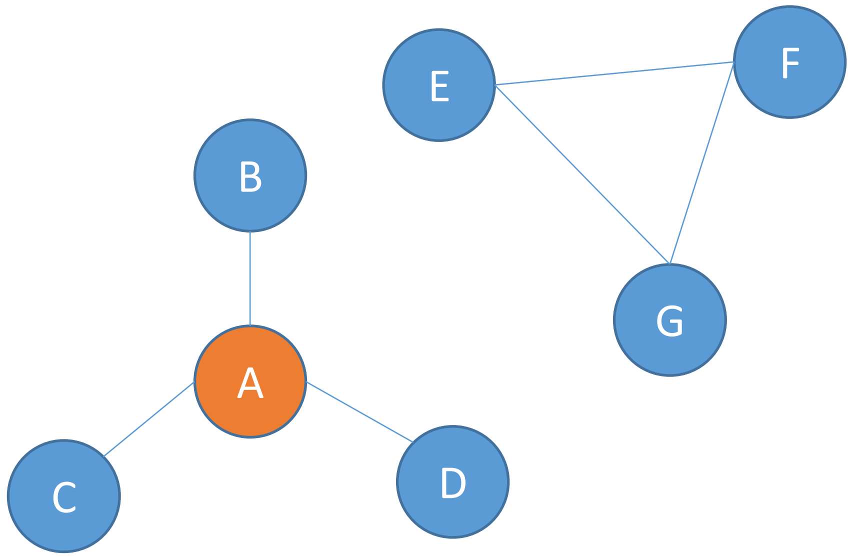
carries_virus(a).

contact(a,(b;c;d)).

contact(e,(f;g)).

contact(f,g).

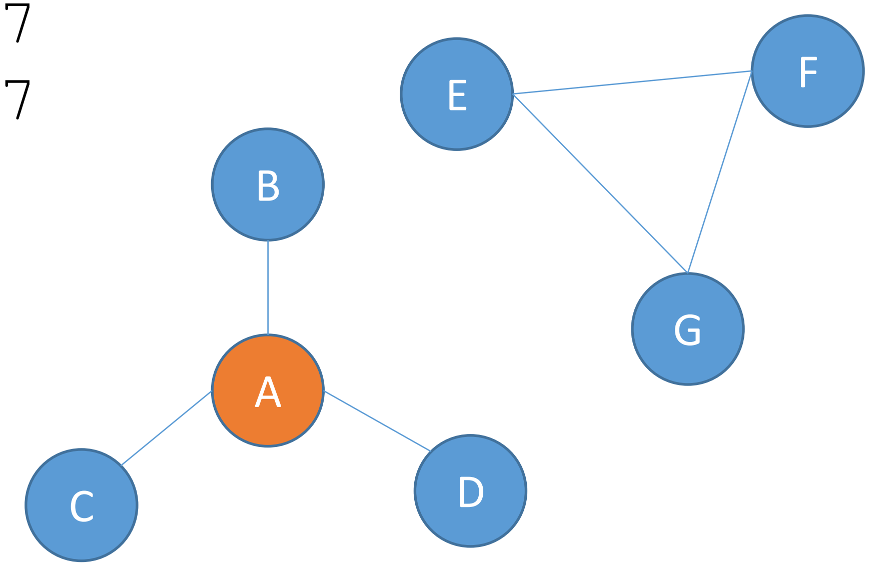
contact(X,Y) :- contact(Y,X).



Example: Virus (2 of 2)

```
$ lpmln-infer input.lpmln -exact -q carries_virus,has_disease
```

```
carries_virus("A") : 1.0000000000000002
carries_virus("B") : 0.7860727393281469
carries_virus("C") : 0.78607273932814 7
carries_virus("D") : 0.78607273932814 7
has_disease("B") : 0.6426730081063122
has_disease("C") : 0.6426730081063122
has_disease("D") : 0.6426730081063122
has_disease("A") : 0.8175744761936435
```



Outline

1. Introduction
2. Intro to ASP
3. Stable Model Semantics
4. Syntax and Semantics of LPMLN
5. Relating LPMLN to Other Languages
6. Inference in LPMLN
7. **Learning in LPMLN**
8. Extension to Embrace Neural Networks

Example

- LP^{MLN} weight learning can be used to learn the certainty degree of hypothesis
- Hypothesis can involve recursive definitions

$$\Pi_{Virus}$$

$$
\begin{aligned}
w_1 &: HasDisease(x) \leftarrow CarriesVirus(x).\\
w_2 &: CarriesVirus(y) \leftarrow Contact(x, y),\\
&\qquad\qquad\qquad\qquad CarriesVirus(x).\\
&\dots\\
\alpha &: CarriesVirus(A).\\
\alpha &: Contact(A, B).\\
\alpha &: Contact(B, C).
\end{aligned}
$$

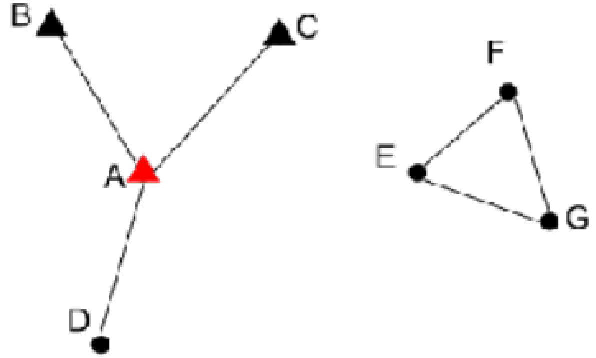
Training Data:

```
:- not carries_virus("F").
:- not carries_virus("G").
:- carries_virus("B").
:- carries_virus("C").
...
:- not has_disease("A").
:- not has_disease("E").
:- has_disease("B").
...
```

Example

"Markov Logic has the drawback that it cannot express (non-ground) inductive definitions" (Fierens et al. 2015) because it relies on classical models.

$$w_1 \ : \ HasDisease(x) \leftarrow CarriesVirus(x).$$
$$w_2 \ : \ CarriesVirus(y) \leftarrow Contact(x, y), CarriesVirus(x).$$



Person	MLN	LP^{MLN}	carries_virus (ground truth)
B	0.823968	0.6226904833	Y
C	0.813969	0.6226904833	Y
D	0.818968	0.6226904833	N
E	0.688981	0	N
F	0.680982	0	N
G	0.680982	0	N

Where do we get weights?

- **It can be manually specified by the user**
  - which may be okay for a simple program
- **A systematic assignment of weights for a complex program could be challenging**

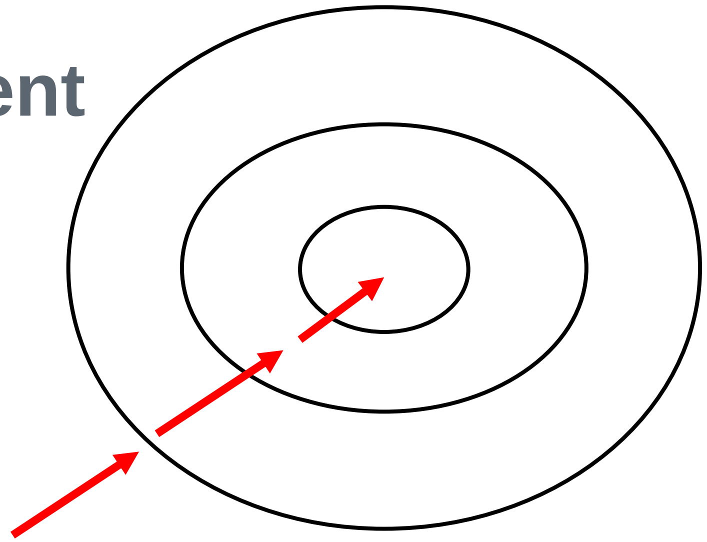
Virus Transmission

```
W1 has_disease(X) :-
        carries_virus(X).

W2 carries_virus(Y) :-
        contact(X, Y),
        carries_virus(X).
```

Gradient Ascent Method for Finding MLE

- **Gradient ascent algorithm use the gradient scaled by a learning rate, λ, to update the weight vector w in each step:**
  - Initialize the weights $w = \{w_1, \ldots, w_m\}$
  - Repeat the following until the weight converges:
    - $w_j := w_j + \lambda \cdot \frac{\partial L}{\partial w_j}$ for $j \in \{1, \ldots, m\}$
- **Move in direction of steepest ascent scaled by learning rate:**



Learning in LPMLN

- **Data is a relational database**

- **For now assume that it gives a complete interpretation (data = an interpretation)**

- **Learning parameters (weights)**

- **Learning structure (rules)**
  - A form of inductive logic programming
  - Also related to learning features for Markov nets

LPMLN Weight Learning (1 of 4)

A parameterized LP^{MLN} program:

- Defined similar to an LP^{MLN} program except that soft weights are replaced with distinct parameters to be learned.

Weight Learning:

- Find the Maximum Likelihood Estimation (MLE) of the parameters, given one complete interpretation as observed data

```
% parameterized program
w1: has_disease(X) :- carries_virus(X).
w2: carries_virus(Y) :- contact(X, Y), carries_virus(X).

% Observed Data (a soft stable model)
carries_virus(E)  -carries_virus(H)   has_disease(A)  -has_disease(H)   ...
```

what are the values of w_1 and w_2 that maximizes the probability of the observed data?

LPMLN Weight Learning (2 of 4)

Gradient Ascent

The given soft stable model as observed data

$$w_i^{j+1} \leftarrow w_i^j + \lambda \cdot \frac{\partial ln P_\Pi(I)}{\partial w_i}$$

$$\frac{\partial ln P_\Pi(I)}{\partial w_i} = \underbrace{-n_i(I)}_{\text{number of ground instances of rule } i\text{, that are not satisfied by I}} + \underbrace{\underset{J \in SM[\Pi]}{E}[n_i(J)]}_{\text{The expectation of number of false ground instances of rule } i}$$

number of ground instances of rule i, that are not satisfied by I

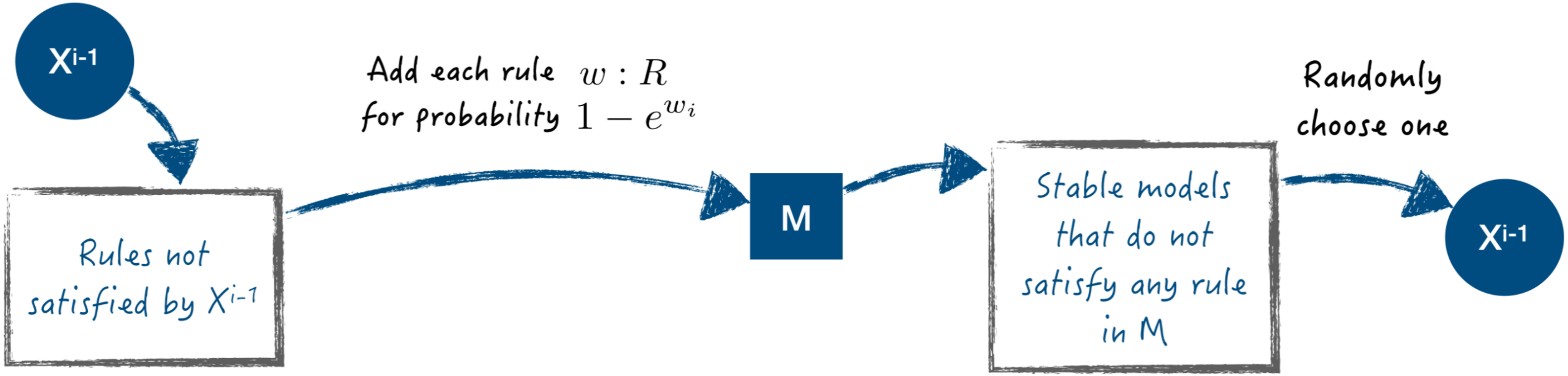
The expectation of number of false ground instances of rule i

Intractable!

LPMLN Weight Learning (3 of 4)

Algorithm MC-ASP

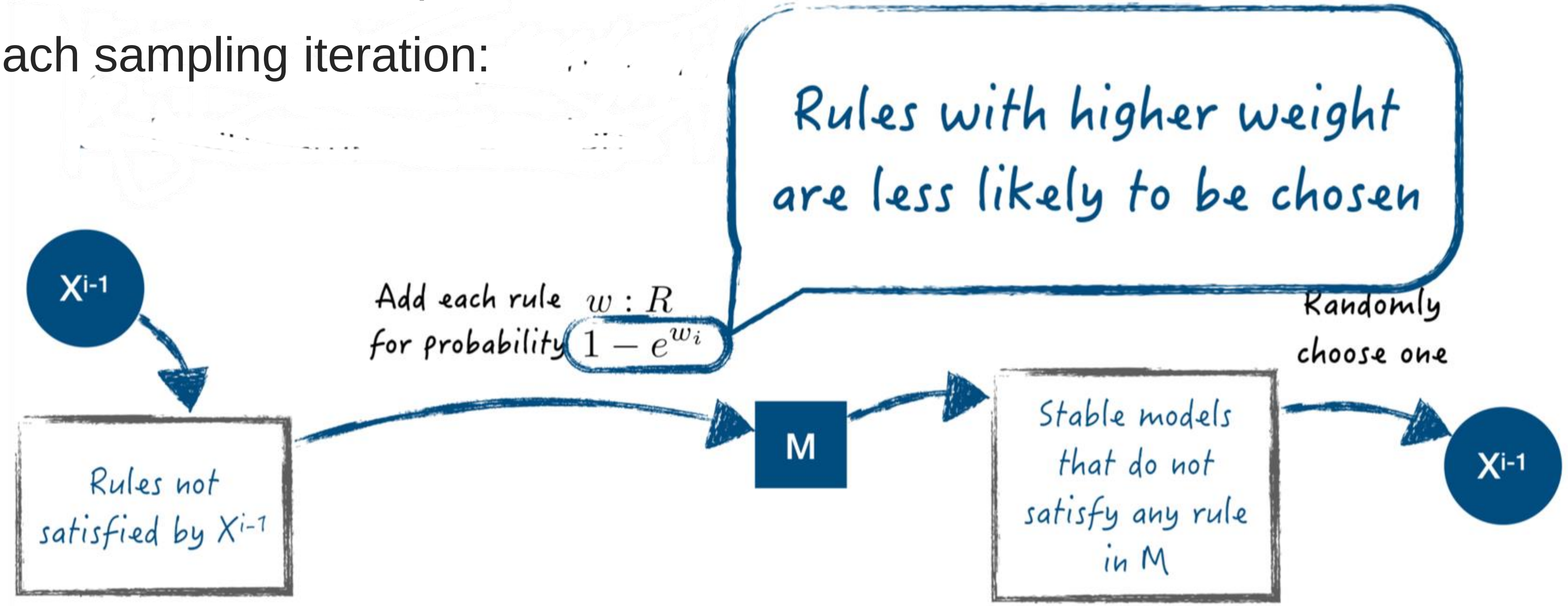
- Adapted from MC-SAT for Markov Logic (Poon and Domingos, 2006)
- Start from a random probabilistic stable model
- Each sampling iteration:



LPMLN Weight Learning (4 of 4)

Algorithm MC-ASP

- Adapted from MC-SAT for Markov Logic (Poon and Domingos, 2006)
- Start from a random probabilistic stable model
- Each sampling iteration:



Outline

1. Introduction
2. Intro to ASP
3. Stable Model Semantics
4. Syntax and Semantics of LPMLN
5. Relating LPMLN to Other Languages
6. Inference in LPMLN
7. Learning in LPMLN
8. **Extension to Embrace Neural Networks**

NeurASP

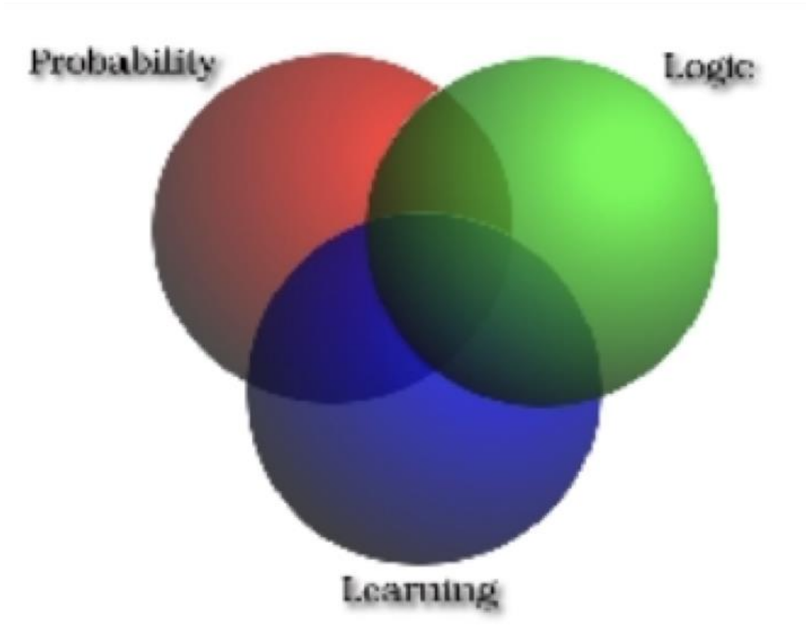
- NeurASP = Neural Networks + Prob. Answer Set Programs

- *“A first desirable property of frameworks that integrate two other frameworks A and B, is to have the original frameworks A and B as a special case of the integrated one.”*

- *“one should not only integrate logic with neural networks in neuro-symbolic computation, but also probability.* “

  —— De Raedt, Luc, et al. 2019

- DeepProbLog, NeurASP, NeuroLog, …



Simple Answer Set Programs

choices

```
digit(d1)=0 | … | digit(d1)=9.
digit(d2)=0 | … | digit(d2)=9.
addition(A, B, N) ← digit(A)=N1,
                     digit(B)=N2,
                     N = N1 + N2.
```

This program has 10 x 10 answer sets (a.k.a. stable models):

$I_{0,0}$ = {digit(d1)=0, digit(d2)=0, addition(0,0,0)},

$I_{0,1}$ = {digit(d1)=0, digit(d2)=1, addition(0,1,1)},

…

Probabilistic ASP

probabilistic choices

```
p1,0:digit(d1)=0 | … | p1,9:digit(d1)=9.
P2,0:digit(d2)=0 | … | p2,9:digit(d2)=9.
addition(A, B, N) ← digit(A)=N1,
                     digit(B)=N2,
                     N = N1 + N2.
```

$\rightarrow$

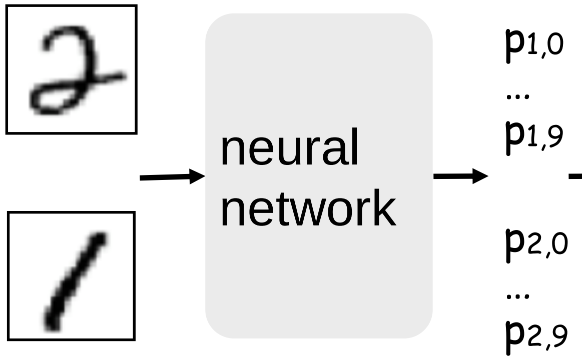
$$P_\Pi(\mathtt{addition(d_1,d_2,3)})$$

$$= P_\Pi(\mathtt{I_{0,3}}) + P_\Pi(\mathtt{I_{1,2}}) + P_\Pi(\mathtt{I_{2,1}}) + P_\Pi(\mathtt{I_{3,0}})$$

$$= p_{1,0} \times p_{2,3} + p_{1,1} \times p_{2,2} + p_{1,2} \times p_{2,1} + p_{1,3} \times p_{2,0}$$

NeurASP: Inference

NeurASP = Neural Networks + Prob. Answer Set Programs




```
p1,0:digit(d1)=0 | … | p1,9:digit(d1)=9.
P2,0:digit(d2)=0 | … | p2,9:digit(d2)=9.
addition(A, B, N) ← digit(A)=N1,
                    digit(B)=N2,
                    N = N1 + N2.
```

$$P_\Pi(\texttt{addition}(\texttt{d}_1,\texttt{d}_2,3))$$

$$= P_\Pi(\texttt{I}_{0,3}) + P_\Pi(\texttt{I}_{1,2}) + P_\Pi(\texttt{I}_{2,1}) + P_\Pi(\texttt{I}_{3,0})$$

$$= p_{1,0} \times p_{2,3} + p_{1,1} \times p_{2,2} + p_{1,2} \times p_{2,1} + p_{1,3} \times p_{2,0}$$

NeurASP: Semantics

The probability of a stable model I of Π is defined as the product of the probability of each atom $c = v$ in $I|_{\sigma^{nn}}$, divided by the number of stable models of Π that agree with $I|_{\sigma^{nn}}$ on σ^{nn}. That is, for any interpretation I,

$$P_\Pi(I) = \begin{cases} \frac{\prod_{c=v \in I|_{\sigma^{nn}}} P_\Pi(c=v)}{Num(I|_{\sigma^{nn}}, \Pi)} & \text{if } I \text{ is a stable model of } \Pi; \\ 0 & \text{otherwise.} \end{cases}$$

An *observation* is a set of ASP constraints (i.e., rules of the form $\perp \leftarrow Body$). The probability of an observation O is defined as

$$P_\Pi(O) = \sum_{I \models O} P_\Pi(I)$$

($I \models O$ denotes that I satisfies O).

NeurASP Example: Sudoku (Inference)

Task: given an image of Sudoku board and a pre-trained neural network to identify the value in each cell, predict the solution.

- Use NN `identify` to identify the digits in each of the 81 grid cells.

```
nn(identify(81,img), [empty,1,2,3,4,5,6,7,8,9]).
```

- Assign one number to each cell `i` for `i` ∈ {1, ..., 81}.

```
a(R,C,N) ← identifyᵢ(img)=N, R=i/9, C=i\9, N≠empty.
{a(R,C,1); …; a(R,C,9)}=1 ← identifyᵢ(img)=empty, R=i/9, C=i\9.
```

- No number repeats in the same row, column, and 3×3 box.

```
← a(R,C1,N), a(R,C2,N), C1≠C2.
← a(R1,C,N), a(R2,C,N), R1≠R2.
← a(R1,C1,N), a(R2,C2,N), R1≠R2, C1≠C2, ((R1/3)×3+C1/3) = ((R2/3)×3+C2/3).
```

NeurASP Advantages (Inference)

- Edit Master text styles
  - Second level
    - Third level

	Method	Input	Number of Data for Training	Accuracy of Solution
NeurASP	Convolutional Neural Network + ASP	Image of Sudoku	25	100%
(Park 2018)	Convolutional Neural Network	Text Representation of Sudoku (9×9 numbers)	1 Million	70.0%
(Palm et al. 2018)	Graph Neural Network	Text Representation of Sudoku (9×9 numbers)	216,000	96.6%

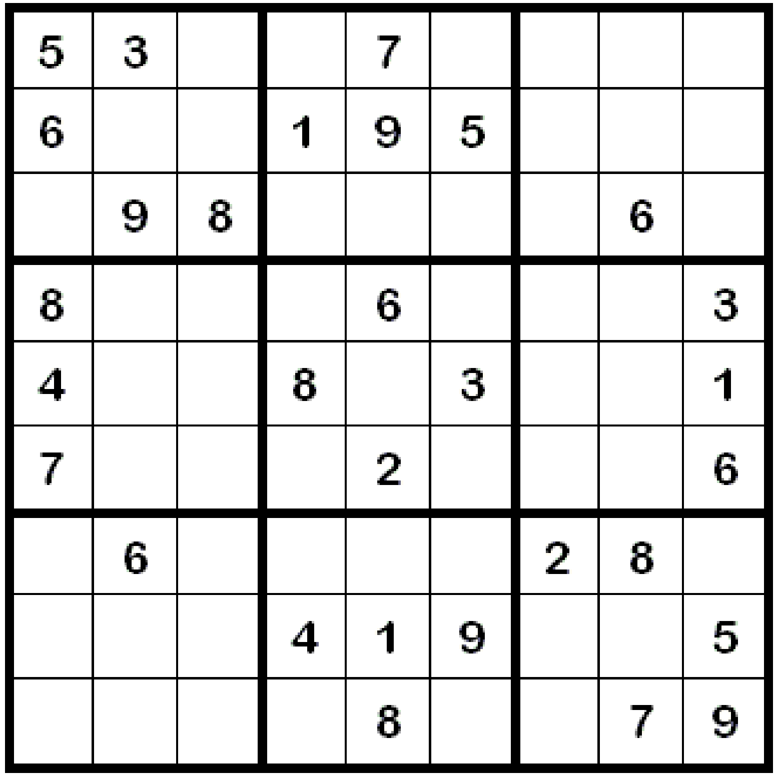
NeurASP Advantages (Inference)

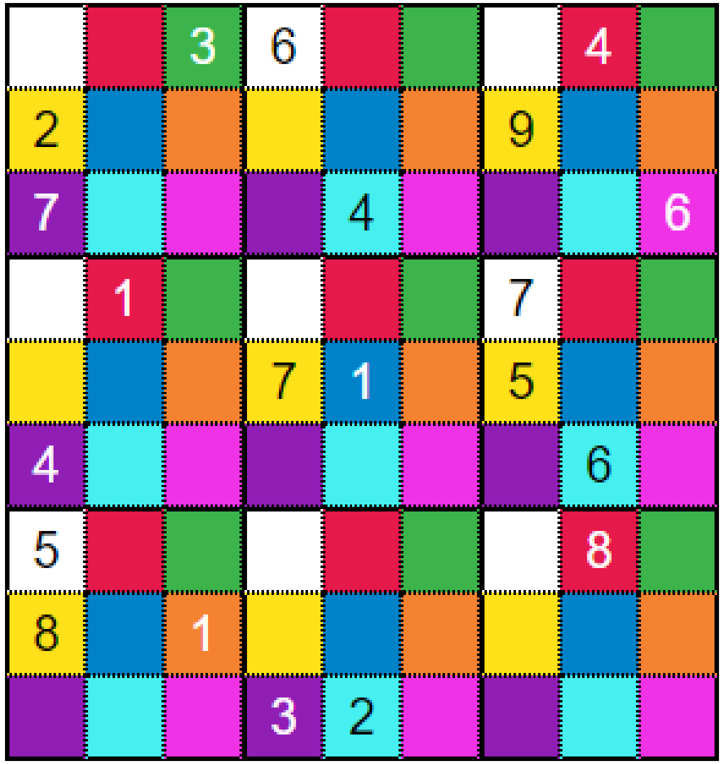
- Edit Master text styles
  - Second level
    - Third level

	Method	Input	Number of Data for Training	Accuracy of Solution
NeurASP	Convolutional Neural Network + ASP	Image of Sudoku	25	100%
(Park 2018)	Convolutional Neural Network	Text Representation of Sudoku (9×9 numbers)	1 Million	70.0%
(Palm et al. 2018)	Graph Neural Network	Text Representation of Sudoku (9×9 numbers)	216,000	96.6%

NeurASP Advantages (Inference)

- Edit Master text styles
  - Second level



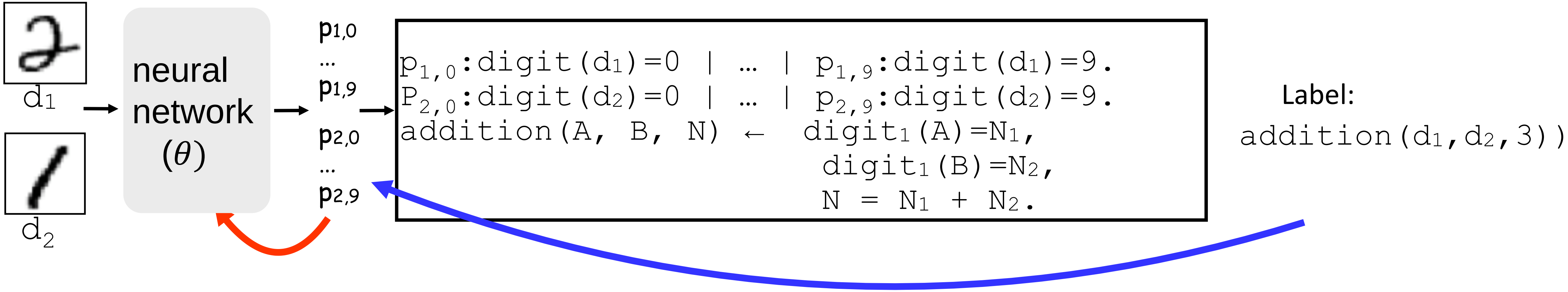


For solving offset sudoku: add

:- a(R1,C1,N), a(R2,C2,N), R1\3 = R2\3, C1\3 = C2\3, R1 != R2, C1 != C2.

NeurASP: Learning

- Given the sum as the label, learn a digit classifier.



Learning is to find the weights of neural network that maximizes the probability of the observation:

$$\hat{\boldsymbol{\theta}} \in \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{O \in \mathbf{O}} log(P_{\Pi(\boldsymbol{\theta})}(O)).$$

$$\frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(addition(d_1,d_2,3))}{\partial \boldsymbol{\theta}} = \sum_{\substack{i\in\{1,2\}\\ j\in\{0,\dots,9\}}} \boxed{\frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(addition(d_1,d_2,3))}{\partial p_{i,j}}} \times \boxed{\frac{\partial p_{i,j}}{\partial \boldsymbol{\theta}}}$$

Gradients Computation

Proposition 1 *Let* $\Pi(\boldsymbol{\theta})$ *be a* NeurASP *program and let* O *be an observation such that* $P_{\Pi(\boldsymbol{\theta})}(O) > 0$. *Let* p *denote the probability of an atom* $c = v$ *in* σ^{nn}, *i.e.,* p *denotes* $P_{\Pi(\boldsymbol{\theta})}(c = v)$. *We have that*[4]

$$\frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial p} = \frac{\sum\limits_{\substack{I:\ I \models O \\ I \models c=v}} \frac{P_{\Pi(\boldsymbol{\theta})}(I)}{P_{\Pi(\boldsymbol{\theta})}(c=v)} - \sum\limits_{\substack{I, v':\ I \models O \\ I \models c=v', v \neq v'}} \frac{P_{\Pi(\boldsymbol{\theta})}(I)}{P_{\Pi(\boldsymbol{\theta})}(c=v')}}{\sum\limits_{I:\ I \models O} P_{\Pi(\boldsymbol{\theta})}(I)}$$

	prob.	atom
neural network	p	c=v
	p'	c=v'
	…	…

Consider a simpler case that there is only one stable model I satisfying O.

$$\frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial p} = \begin{cases} \frac{1}{p} & \text{if } I \models c = v; \\ -\frac{1}{p'} & \text{if } I \models c = v' \text{ and } v' \neq v. \end{cases}$$

NeurASP Example: Sudoku

Task: given an image of Sudoku board and a pre-trained neural network to identify the value in each cell, predict the solution.

- Use NN `identify` to identify the digits in each of the 81 grid cells.

```
nn(identify(81,img), [empty,1,2,3,4,5,6,7,8,9]).
```

- Assign one number to each cell `i` for `i` ∈ {1, ..., 81}.

$$a(R,C,N) \leftarrow identify_i(img)=N,\ R=i/9,\ C=i \backslash 9,\ N \neq empty.$$

$$\{a(R,C,1);\ \ldots;\ a(R,C,9)\}=1 \leftarrow identify_i(img)=empty,\ R=i/9,\ C=i \backslash 9.$$

- No number repeats in the same row, column, and 3×3 box.

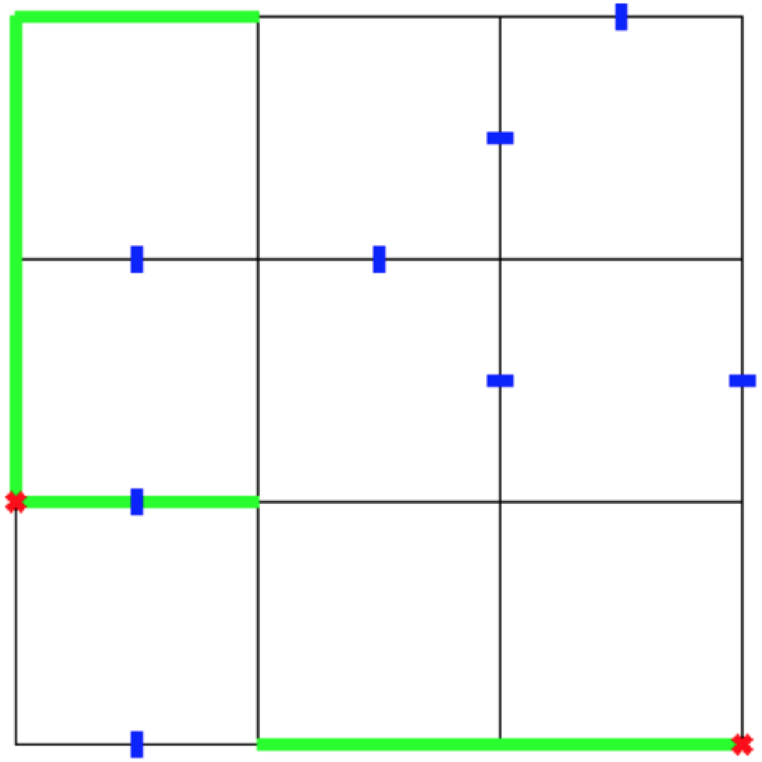
$$\leftarrow a(R,C_1,N),\ a(R,C_2,N),\ C_1 \neq C_2.$$

$$\leftarrow a(R_1,C,N),\ a(R_2,C,N),\ R_1 \neq R_2.$$

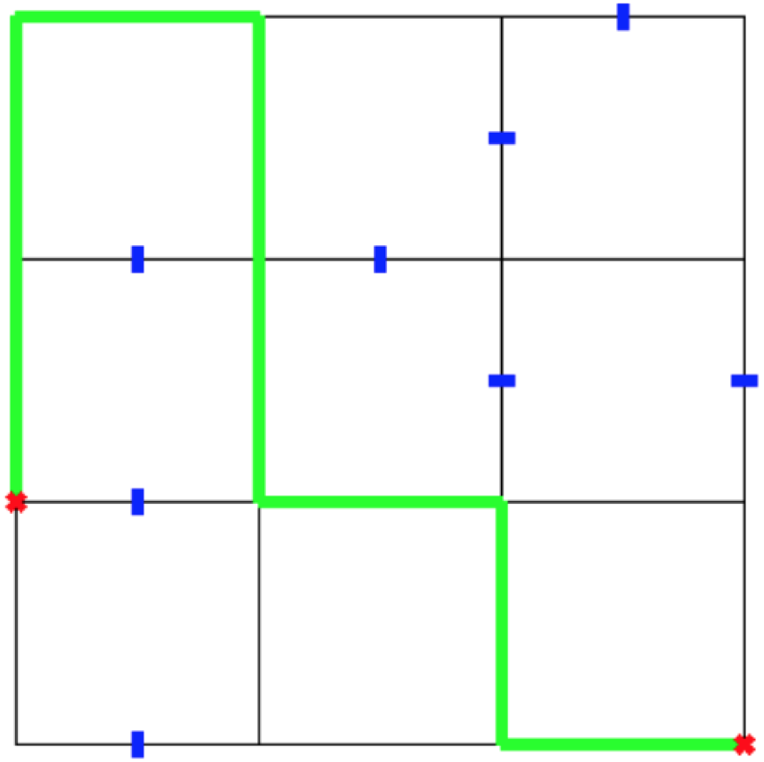
$$\leftarrow a(R_1,C_1,N),\ a(R_2,C_2,N),\ R_1 \neq R_2,\ C_1 \neq C_2,\ ((R_1/3)\times 3+C_1/3) = ((R_2/3)\times 3+C_2/3).$$

NeurASP Advantages (Learning)

4. NeurASP can be used to inject constraints into neural networks



MLP with Cross-entropy loss



MLP trained with NeurASP

NeurASP Advantages (Learning)

4. NeurASP can be used to inject constraints into neural networks

	Predictions satisfying Path constraints	Predictions satisfying Shortest Path constraints constraints
MLP trained with cross-entropy	28.3%	23.0%
MLP trained with NeurASP Using rules for Path constraints	96.6%	33.2%
MLP trained with NeurASP Using rules for Shortest Path constraints	100%	45.7%

```
← X=0..15, #count{Y: sp(X,Y)} = 1.
← X=0..15, #count{Y: sp(X,Y)} ≥ 3.
reachable(X,Y) :- sp(X,Y).
reachable(X,Y) :- reachable(X,Z), sp(Z,Y).
:- sp(X,A), sp(Y,B), not reachable(X,Y).
:~ sp(X,g,true). [1, X]
```

Path

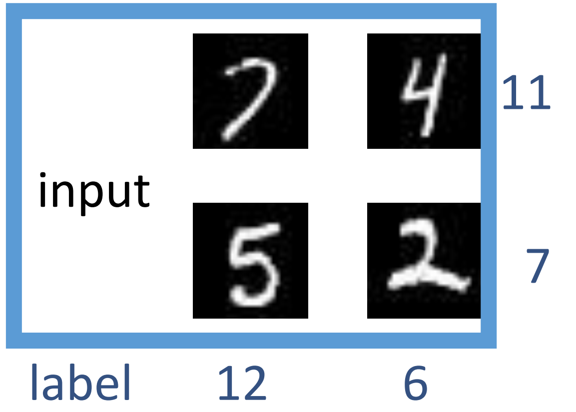
Shortest Path

NeurASP Advantages (Learning)

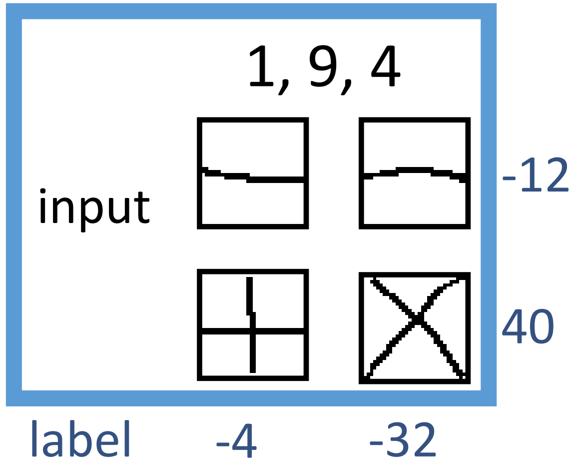
5. NeurASP allows one to train a NN under weak supervision.

	add2x2	apply2x2	member(3)	member(5)
accuracy(%)				
DeepProbLog	88.4±0.7	100±0	96.3±0.3	timeout
NeuroLog	97.5±0.4	100±0	94.5±1.5	93.9±1.5
NeurASP	97.6±0.2	100±0	93.5±0.9	timeout
time(s)				
DeepProbLog	1035±71	586±9	2218±211	timeout
NeuroLog	2400±46	2428±29	427±12	682±40
NeurASP	142±2	47±1	253±1	timeout

add2x2



apply2x2



Outline

1. Introduction
2. Review of Stable Model Semantics
3. Syntax and Semantics of LPMLN
4. Relation to Other Languages
5. Inference in LPMLN
6. Learning in LPMLN
7. Extension to Embrace Neural Network Components
8. **Other Related works**

Papers Related to LP^{MLN}

- Language LP^{MLN} proposed [AAAI 2015, KR 2016, ICLP 2015, Commonsense 2016]
- LP^{MLN} inference & LP^{MLN} solver [TPLP 2017]
- Splitting theorem for LP^{MLN} [Wang et al. AAAI 2018]
- Parallel LP^{MLN} solver [Wu et al. ICTAI 2018]
- Relationship between LP^{MLN} and P-Log [Gelfond and Balai IJCAI 2017; AAAI 2017]
- Using LP^{MLN} for hybrid classification with contextual knowledge [Eiter & Kaminski, JELIA 2016]

Papers Related to LP^{MLN}

- Weight learning in LP^{MLN} [KR 2018]
- Probabilistic action language pBC+ based on LP^{MLN} [TPLP 2018]
- Decision-theoretic LP^{MLN} [LPNMR 2019]
- Extension of pBC+ for elaboration tolerant representation of (PO)MDP [LPNMR 2019]
- Strong equivalence for LP^{MLN} [ICLP 2019]
- Explainable fact checking LP^{MLN} [TTO 2019]
- NeurASP [IJCAI 2020]
- PLINGO [Hahn et al., 2022]

Thank you!