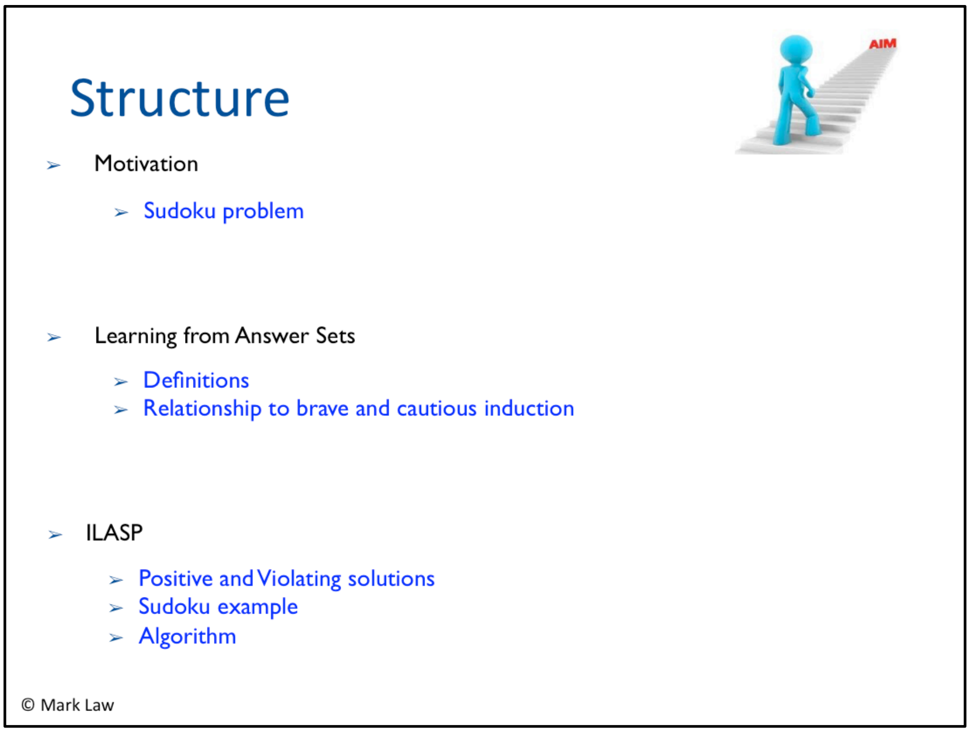
Learning from Answer Sets

Mark Law
mark.law09@imperial.ac.uk



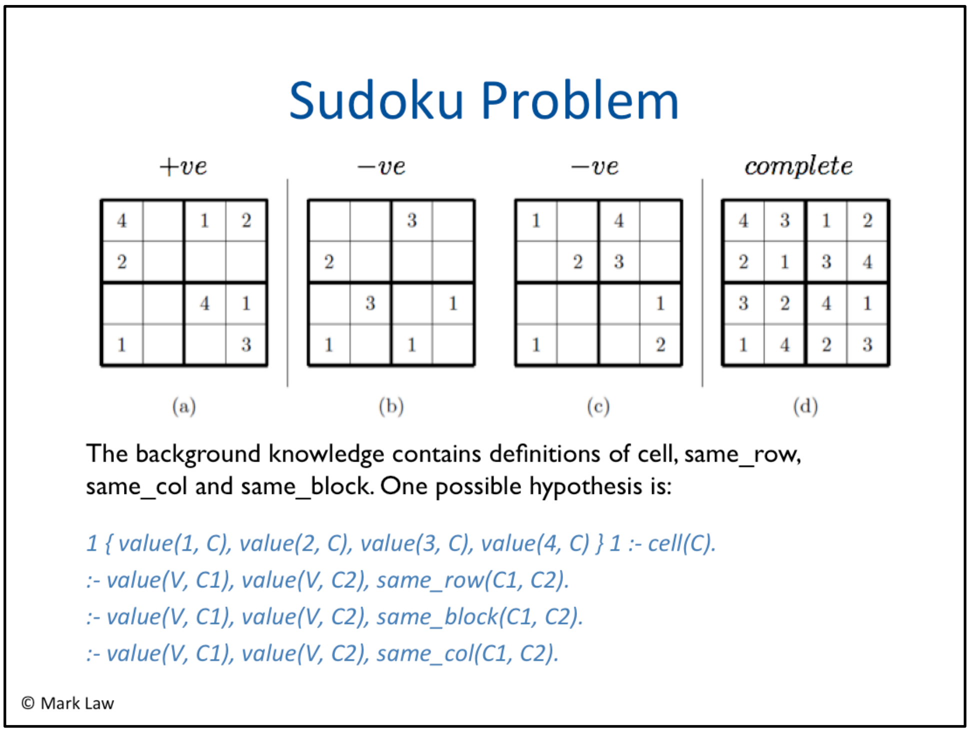
Structure

- Motivation
  - Sudoku problem
- Learning from Answer Sets
  - Definitions
  - Relationship to brave and cautious induction
- ILASP
  - Positive and Violating solutions
  - Sudoku example
  - Algorithm

In previous lectures, we have seen the limitations of approached using only brave or cautious induction when attempting to learn Answer Set Programs. In this lecture we look at a recent framework called Learning from Answer Sets *(Law, M; Russo, A; Broda, K. JELIA 2014)*. We will also cover its algorithm ILASP (Inductive Learning of Answer Set Programs) and see how it can be applied to learn the rules of sudoku as represented in ASP in an earlier lecture.

Motivation

In previous lectures, we have considered two different kinds of induction in ASP: *brave* and *cautious*. We have studied an algorithm, ASPAL, which is able to compute the solutions of a brave induction task and, although we haven't considered it in this course, has been applied to real applications such as learning user behaviours from mobile devices *(Markitanis, A; Corapi, D; Russo, A; Lupu, E. ILP 2011)* and Robot-driven rule learning in non-monotonic domains *(Corapi, D; Sykes, A; Inoue, K; Russo, A ICSE 2013)*.
These applications demonstrate that many problems can be solved with brave induction alone; however, as shown in previous lectures, there are programs which cannot be learned with solely brave or cautious induction. One such problem is to learn the rules of sudoku.



Sudoku Problem

$+ve$

4		1	2
2			
		4	1
1			3

(a)

$-ve$

		3	
2			
	3		1
1		1	

(b)

$-ve$

1		4	
	2	3	
			1
1			2

(c)

$complete$

4	3	1	2
2	1	3	4
3	2	4	1
1	4	2	3

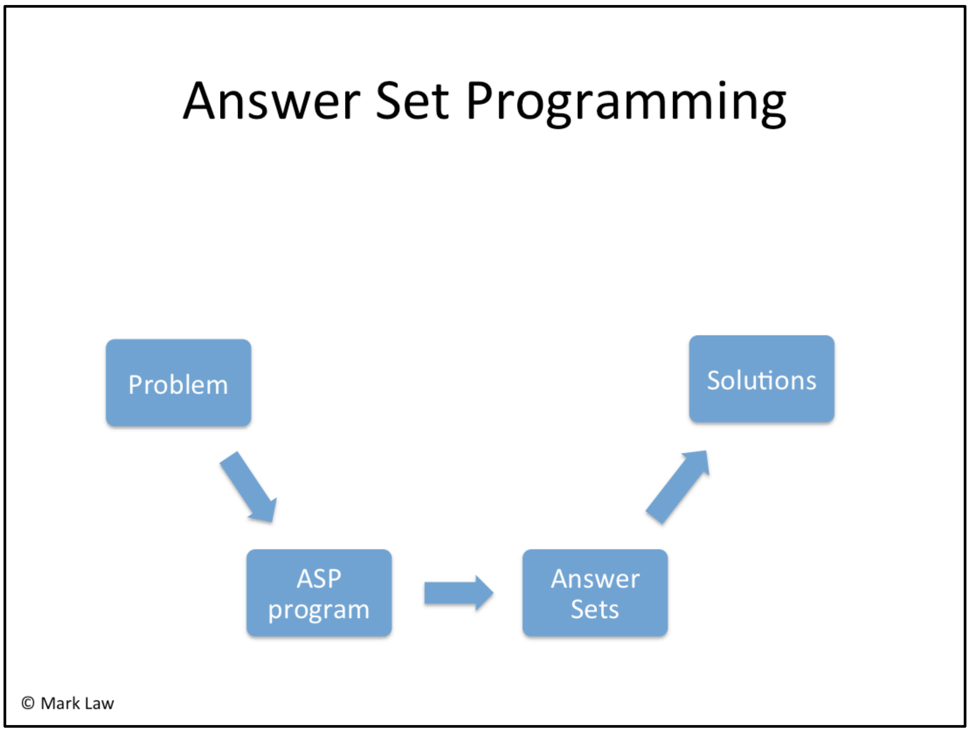
(d)

The background knowledge contains definitions of cell, same_row, same_col and same_block. One possible hypothesis is:

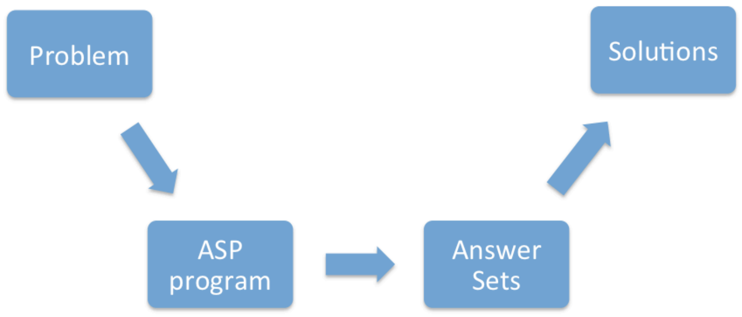
```
1 { value(1, C), value(2, C), value(3, C), value(4, C) } 1 :- cell(C).
:- value(V, C1), value(V, C2), same_row(C1, C2).
:- value(V, C1), value(V, C2), same_block(C1, C2).
:- value(V, C1), value(V, C2), same_col(C1, C2).
```

Given a background knowledge containing definitions of what it means for two different cells to be in the same row, column and block (and the definition of cell), the task is to learn an ASP program whose Answer Sets correspond exactly with the valid sudoku boards of a given size. The program on the slide is one possible solution.

If we were to try to learn this program using brave induction, we would not be able to learn the constraint which rule out Answer Sets (as discussed in previous lectures). If the hypothesis on the slide is an inductive solution of any given brave induction task, then so is the hypothesis containing only the first rule! Conversely, if we were to try to learn the hypothesis using only cautious induction, then we could only give as examples atoms which are either true in every Answer Set of the program, or false in every Answer Set. As there are many different sudoku board, there is no atom value(V, C) where V is a value between 1 and 4 and C is a cell, which has the same truth value in every Answer Set (as there is no cell which takes the same value in every possible sudoku board). We therefore cannot learn the choice rule in the hypothesis.

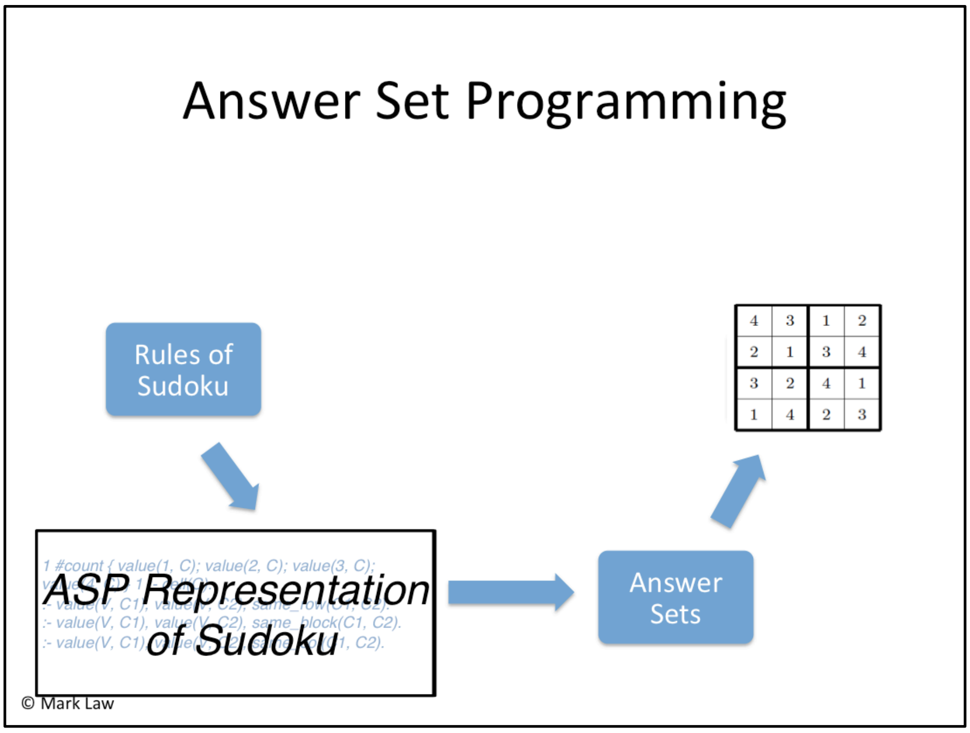


Answer Set Programming



The Answer Set Programming paradigm is to translate the problem we want to solve into an Answer Set Program such that when we solve the program for Answer Sets, these Answer Sets can then be translated back into solutions to the original problem.

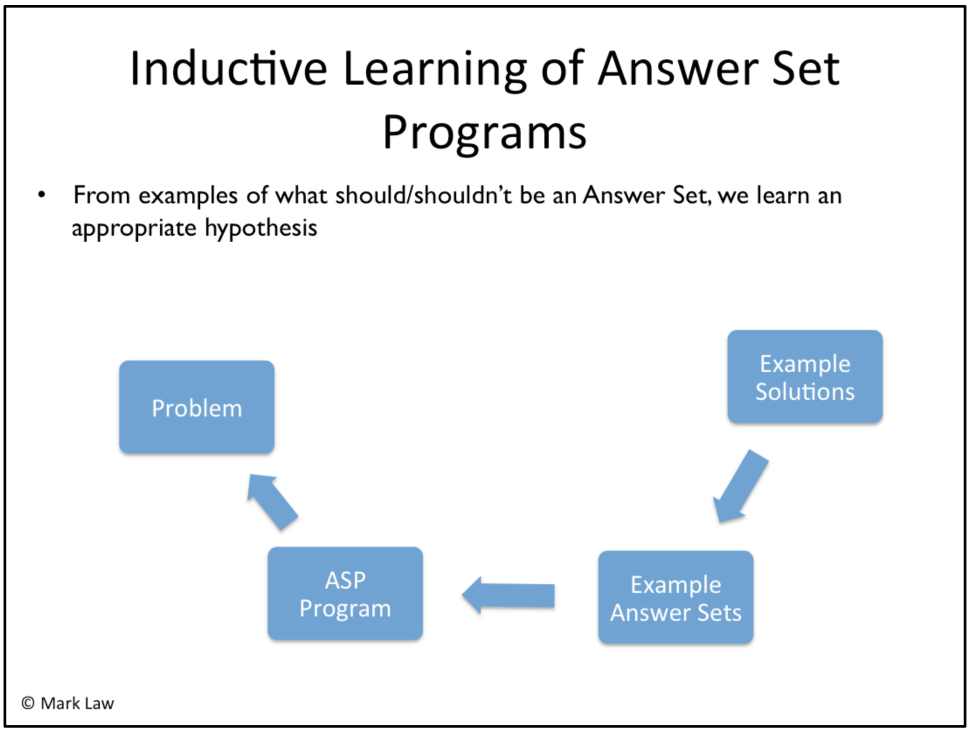
Answer Set Programming



```
1 #count { value(1, C); value(2, C); value(3, C);
value(4, C) } 1 :- cell(C).
:- value(V, C1), value(V, C2), same_row(C1, C2).
:- value(V, C1), value(V, C2), same_block(C1, C2).
:- value(V, C1), value(V, C2), same_col(C1, C2).
```

4	3	1	2
2	1	3	4
3	2	4	1
1	4	2	3

To solve Sudoku using ASP we translate the problem into an ASP program and solve for Answer Sets. These Answer Sets can then be mapped back into the valid Sudoku boards.



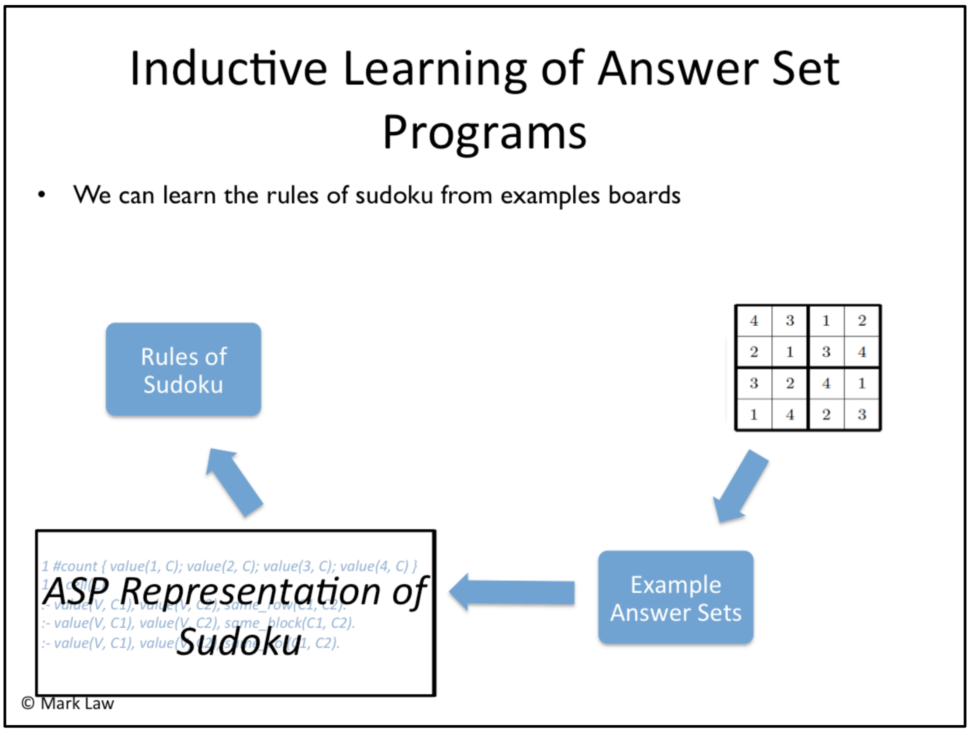
Inductive Learning of Answer Set Programs

- From examples of what should/shouldn't be an Answer Set, we learn an appropriate hypothesis

Problem
Example Solutions
ASP Program
Example Answer Sets

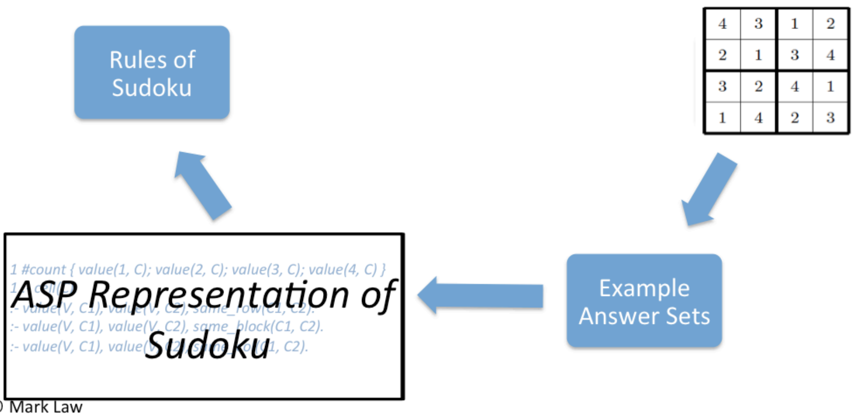
Atoms are usually given as examples in Inductive Logic Programming, as the task is to learn how to classify similar atoms as either true or false; however, when learning an ASP program, the task is not to learn the truth value of particular atoms, but to learn what should (or shouldn't) be an Answer Set of the program. It makes sense for our examples to be Answer Sets as these are the main objects for reasoning in ASP.

For this reason, the learning task we are going to see in this lecture, has as its examples (partial) Answer Sets rather than atoms. We shall see that this new learning task, *Learning from Answer Sets*, is in fact, capable of expressing both brave and cautious induction within a single learning task.



Inductive Learning of Answer Set Programs

- We can learn the rules of sudoku from examples boards

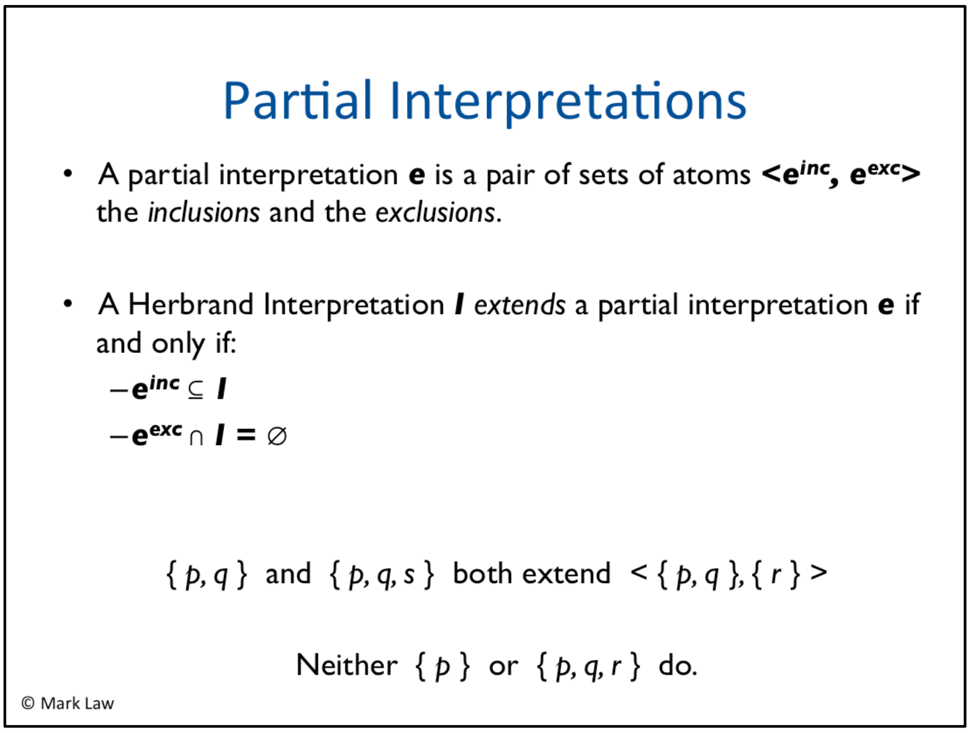


4	3	1	2
2	1	3	4
3	2	4	1
1	4	2	3

```
1 #count { value(1, C); value(2, C); value(3, C); value(4, C) }
1 :- cell(C).
:- value(V, C1), value(V, C2), same_row(C1, C2).
:- value(V, C1), value(V, C2), same_block(C1, C2).
:- value(V, C1), value(V, C2), same_col(C1, C2).
```

For example, we shouldn't try to learn the rules of Sudoku by examples of what is allowing in a particular cell as technically each of the numbers 1 to 4 could go in each cell in these boards. One cell on its own doesn't help us much. To learn the rules of Sudoku we need examples of valid and invalid boards (or at least partial boards). This corresponds to example partial Answer Sets. We can then learn an ASP program corresponding to the rules of Sudoku.

Learning from Answer Sets



Partial Interpretations

- A partial interpretation **e** is a pair of sets of atoms **<e^{inc}, e^{exc}>** the *inclusions* and the *exclusions*.

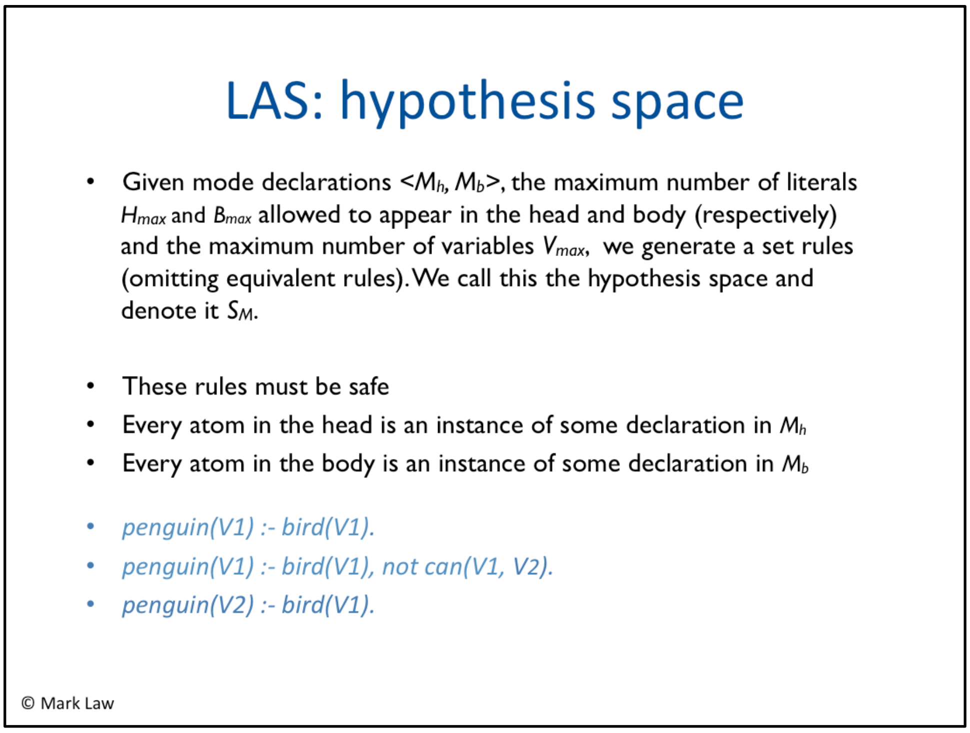
- A Herbrand Interpretation ***I*** *extends* a partial interpretation **e** if and only if:
  - $e^{inc} \subseteq I$
  - $e^{exc} \cap I = \varnothing$

$\{ p, q \}$ and $\{ p, q, s \}$ both extend $< \{ p, q \}, \{ r \} >$

Neither $\{ p \}$ or $\{ p, q, r \}$ do.

Examples in *Learning from Answer Sets* are partial interpretations. These are pairs of sets of atoms, the inclusions e^{inc} and the exclusions e^{exc}. A Herbrand Interpretation extends a partial interpretation if it includes all of the inclusions and none of the exclusions.

In Learning from Answer Sets, examples are covered if there exists and Answer Set of B union H which extends the example.



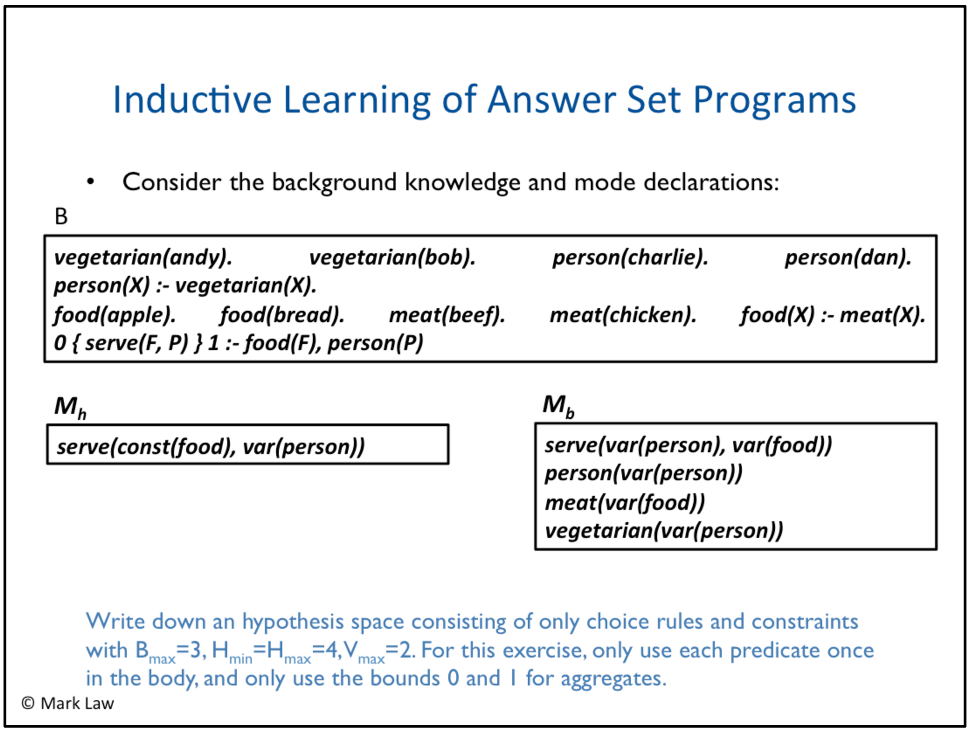
LAS: hypothesis space

- Given mode declarations $<M_h, M_b>$, the maximum number of literals H_{max} and B_{max} allowed to appear in the head and body (respectively) and the maximum number of variables V_{max}, we generate a set rules (omitting equivalent rules). We call this the hypothesis space and denote it S_M.

- These rules must be safe
- Every atom in the head is an instance of some declaration in M_h
- Every atom in the body is an instance of some declaration in M_b

- *penguin(V1) :- bird(V1).*
- *penguin(V1) :- bird(V1), not can(V1, V2).*
- *penguin(V2) :- bird(V1).*

Learning from Answer Sets is slightly less prescriptive about the rules in its hypothesis space than ASPAL (or other tasks). This is because, while other tasks targeted Prolog where floundering is a problem, floundering is not an issue in ASP. LAS allows rules so long as they are safe (unsafe rules result in infinite groundings if function symbols are present anywhere in the program and therefore are not permitted by modern ASP solvers).
Other than this, the normal rules generated by LAS are similar to ASPAL. For reasons of efficiency, using skeleton rules and pushing the task of grounding the constant variables to the ASP solver is not a good idea for the algorithm we use to solve LAS tasks (this is because the meta representation in LAS must be solved multiple times, so the ASP solver would be grounding the constants multiple times; whereas, in ASPAL the meta encoding is only solved once). LAS therefore has no concept of skeleton rule and instead, the hypothesis space contains rules with the constant symbols already ground. For this reason, when using ILASP, allowed constants (and their types must be specified in the task).
LAS also allows two previously unconsidered types of rule in its hypothesis space: choice rules, and constraints. Constraints are easily generated (just like normal rules, but with no head). Choice rules have an aggregate as their head; their body is constructed as usual, and the aggregate can contain Hmin – Hmax atoms. There is one instance of the rule for each set of appropriate bounds.
The lengths of normal rules and constraints is obvious, we just count the number of literals. For aggregates, it is less clear. We will not consider aggregate length in this course, but if you run ILASP and get a hypothesis which is less optimal (using the metric of counting literals) than you were expecting, this is why (all hypotheses are still correct inductive solutions).



Inductive Learning of Answer Set Programs

- Consider the background knowledge and mode declarations:

B

```
vegetarian(andy).     vegetarian(bob).     person(charlie).     person(dan).
person(X) :- vegetarian(X).
food(apple).    food(bread).    meat(beef).    meat(chicken).    food(X) :- meat(X).
0 { serve(F, P) } 1 :- food(F), person(P)
```

M_h

```
serve(const(food), var(person))
```

M_b

```
serve(var(person), var(food))
person(var(person))
meat(var(food))
vegetarian(var(person))
```

Write down an hypothesis space consisting of only choice rules and constraints with B_{max}=3, H_{min}=H_{max}=4, V_{max}=2. For this exercise, only use each predicate once in the body, and only use the bounds 0 and 1 for aggregates.

As ILASP does not make use of input and output variable in its mode declarations (not needing to prevent rules which would cause floundering in Prolog), it uses slightly different notation for its mode declarations. For a literal to be compatible with a mode declaration, it must replace all var(type) terms with variables and all const(type) with constants of that type. All instances of a variable within a rule must belong to the same type.

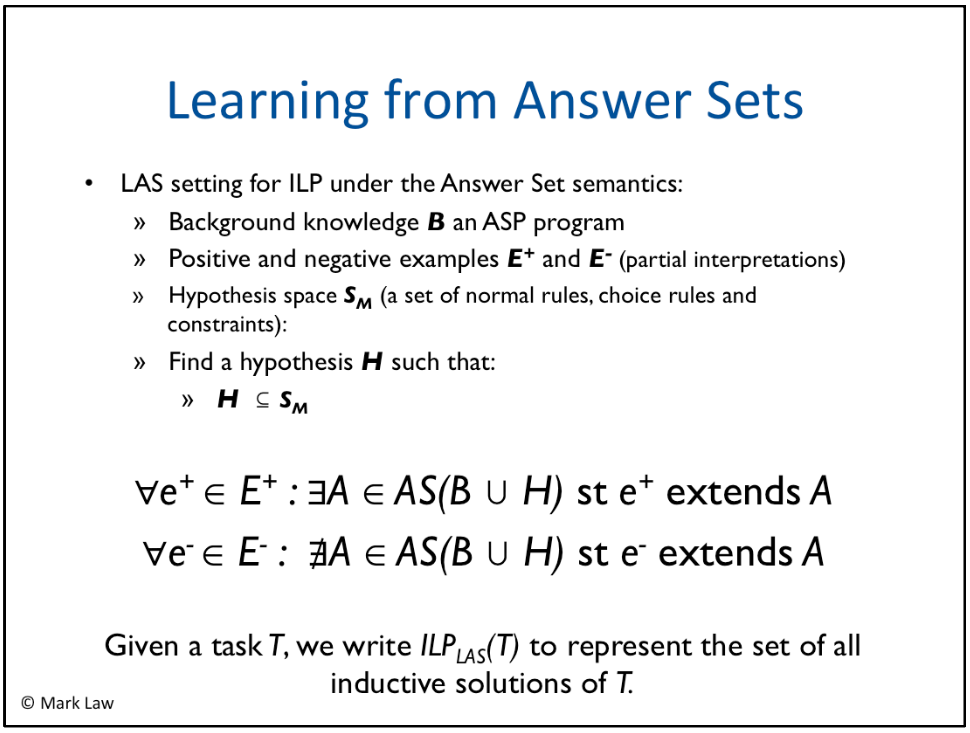
Here, to make computation simpler for you, we use $\boldsymbol{H_{min}}$ as the minimum number of atoms within an aggregate.

Inductive Learning of Answer Set Programs

Let H_1 = 0 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1

Let H_2 = 1 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1

S_M *contains:*



Learning from Answer Sets

- LAS setting for ILP under the Answer Set semantics:
  - » Background knowledge ***B*** an ASP program
  - » Positive and negative examples $\boldsymbol{E^+}$ and $\boldsymbol{E^-}$ (partial interpretations)
  - » Hypothesis space $\boldsymbol{S_M}$ (a set of normal rules, choice rules and constraints):
  - » Find a hypothesis ***H*** such that:
    - » $\boldsymbol{H} \subseteq \boldsymbol{S_M}$

$$\forall e^+ \in E^+ : \exists A \in AS(B \cup H) \text{ st } e^+ \text{ extends } A$$
$$\forall e^- \in E^- : \nexists A \in AS(B \cup H) \text{ st } e^- \text{ extends } A$$

Given a task *T*, we write $ILP_{LAS}(T)$ to represent the set of all inductive solutions of *T*.

© Mark Law

A *Learning from Answer Sets* task (LAS) is a tuple $\boldsymbol{<B, S_M, E^+, E^- >}$ where ***B*** is an Answer Set program called the background knowledge, $\boldsymbol{S_M}$ is the set of rules with which we are allowed to construct our hypotheses (these are normal rules, choice rules and constraints) and $\boldsymbol{E^+}$ and $\boldsymbol{E^-}$ are partial interpretations.

The hypothesis space $\boldsymbol{S_M}$ is usually constructed from a set of mode declarations ***M***. Unlike ASPAL and similar systems, as LAS is aimed at learning ASP rather than Prolog, LAS has no concept of input and output variables. The only restriction is that the rules in $\boldsymbol{S_M}$ are safe. The search space is therefore bigger in general. LAS also allows for constraints and choice rules (rules with a counting aggregate in the head). The heads of choice rules are allowed to be any aggregate with all atoms compatible with mode declarations in $\boldsymbol{M_h}$.

A hypothesis ***H*** is an inductive solution (written $\boldsymbol{H \in ILP_{LAS}<B, S_M, E^+, E^- >}$) if and only if it is constructed from the rules in $\boldsymbol{S_M}$, and each positive example is extended by at least one Answer Set of $\boldsymbol{B \cup H}$ (this can be a different Answer Set for each positive example) and no negative example is extended by any Answer Set of $\boldsymbol{B \cup H}$.

Learning from Answer Sets

Given S_M containing the rules with any aggregate of the form:

a {value(C, heads), value(C, tails) } b (for suitable *a* and *b*) at the head (or no head at all) and *value(C, heads)*, *value(C, tails)*, *coin(C)* or *biased_coin(C)* in the body.

Consider the task:

B

```
coin(C) :- biased_coin(C).
biased_coin(c1).
coin(c2).
```

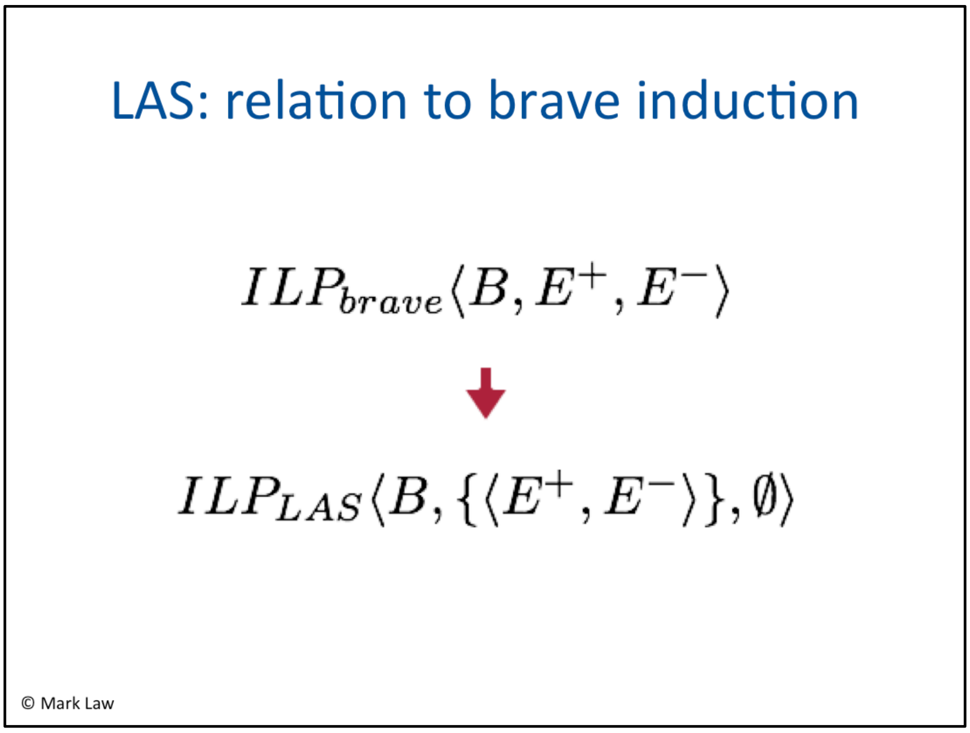
E$^+$

```
< { value(c2, tails), value(c1, heads) }, { } >
```

E$^-$

```
< { }, {value(c2, heads), value(c2, tails)} >
< {value(c2, heads), value(c2, tails)}, {} >
< { }, {value(c1, tails)} >
```

Find an hypothesis which covers the examples:



LAS: relation to brave induction

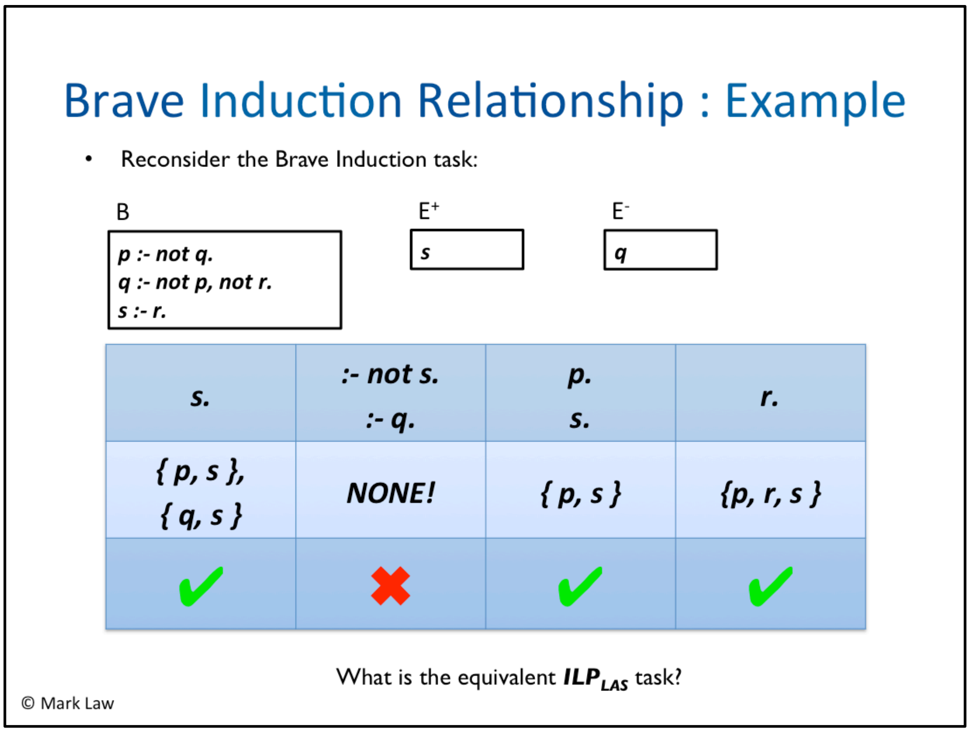
$$ILP_{brave}\langle B, E^{+}, E^{-}\rangle$$

$$\Downarrow$$

$$ILP_{LAS}\langle B, \{\langle E^{+}, E^{-}\rangle\}, \emptyset\rangle$$

A brave induction task is satisfied by a hypothesis ***H*** if and only if there is at least one Answer Set of ***B* ∪ *H*** which includes all of the positive examples ***E*$^{+}$** and none of the negative examples ***E*$^{-}$**. This is equivalent to there being an Answer Set of ***B* ∪ *H*** which extends the partial interpretation ***<E^{+}, E^{-}>***, which is, in turn, equivalent to ***H*** being an inductive solution of a *Learning from Answer Sets* task with the single (positive) example ***<E^{+}, E^{-}>***.

For this comparison, we ignore the hypothesis space of *Learning from Answer Sets*, as (because it had no concrete implementation) brave induction never had a fixed hypothesis space.



Brave Induction Relationship : Example

- Reconsider the Brave Induction task:

B

```
p :- not q.
q :- not p, not r.
s :- r.
```

E^+

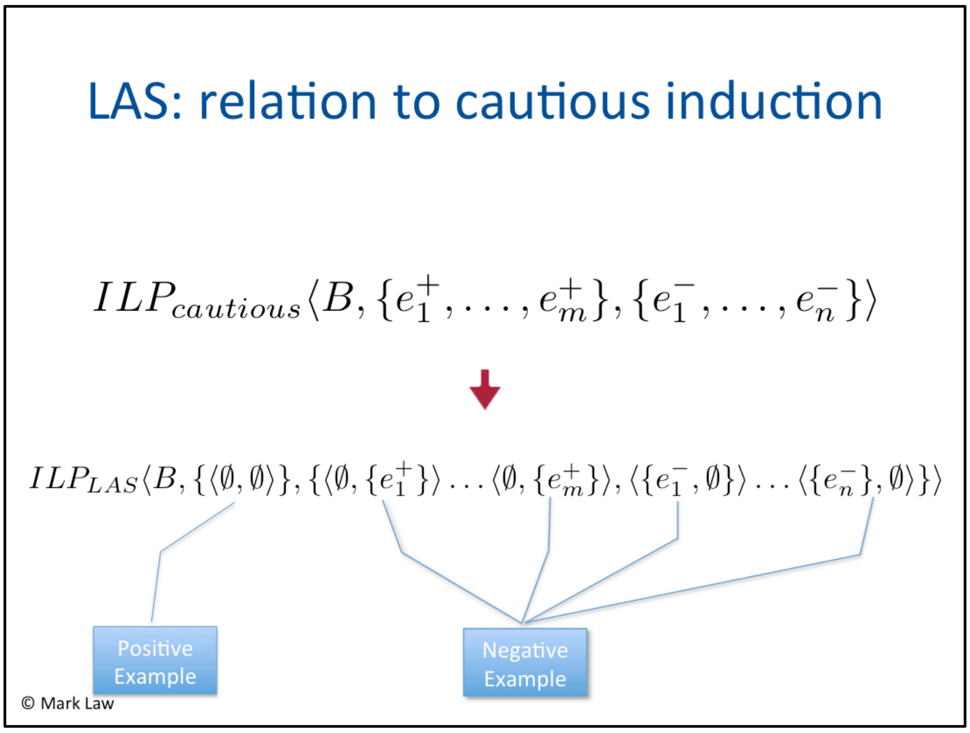
```
s
```

E^-

```
q
```

s.	*:- not s.* *:- q.*	*p.* *s.*	*r.*
{ p, s }, { q, s }	*NONE!*	*{ p, s }*	*{p, r, s }*
✓	✗	✓	✓

What is the equivalent $\boldsymbol{ILP_{LAS}}$ task?



LAS: relation to cautious induction

$$ILP_{cautious}\langle B, \{e_1^+, \ldots, e_m^+\}, \{e_1^-, \ldots, e_n^-\}\rangle$$

$$\Downarrow$$

$$ILP_{LAS}\langle B, \{\langle\emptyset,\emptyset\rangle\}, \{\langle\emptyset,\{e_1^+\}\rangle \ldots \langle\emptyset,\{e_m^+\}\rangle, \langle\{e_1^-,\emptyset\}\rangle \ldots \langle\{e_n^-\},\emptyset\rangle\}\rangle$$

Positive Example
Negative Example

A cautious induction task is satisfied by a hypothesis ***H*** if and only if every Answer Set of ***B* ∪ *H*** includes all of the positive examples ***E***$^+$ and none of the negative examples ***E***$^-$ (and ***B* ∪ *H*** has at least one Answer Set). This is equivalent to there being no Answer Set of ***B* ∪ *H*** which contains any negative example, and for each positive example, no Answer Set which does not contain that example (and ***B* ∪ *H*** having at least one Answer Set). This is equivalent to ***B* ∪ *H*** having no Answer Set which extends any of the partial interpretations $\langle\emptyset, e^+_1\rangle, \ldots, \langle\emptyset, e^+_m\rangle, \langle e^-_1, \emptyset\rangle, \ldots, \langle e^-_n, \emptyset\rangle$ (and at least one Answer Set which extends $\langle\emptyset, \emptyset\rangle$) which is, in turn, equivalent to ***H*** being an inductive solution of the *Learning from Answer Sets* task shown on the slide.

For this comparison, we ignore the hypothesis space of *Learning from Answer Sets*, as (because it had no concrete implementation) cautious induction never had a fixed hypothesis space.

Cautious Induction Relationship: Example

- Reconsider the Cautious Induction task:

B

```
p :- not q.
q :- not p, not r.
s :- r.
```

E^+

```
s
```

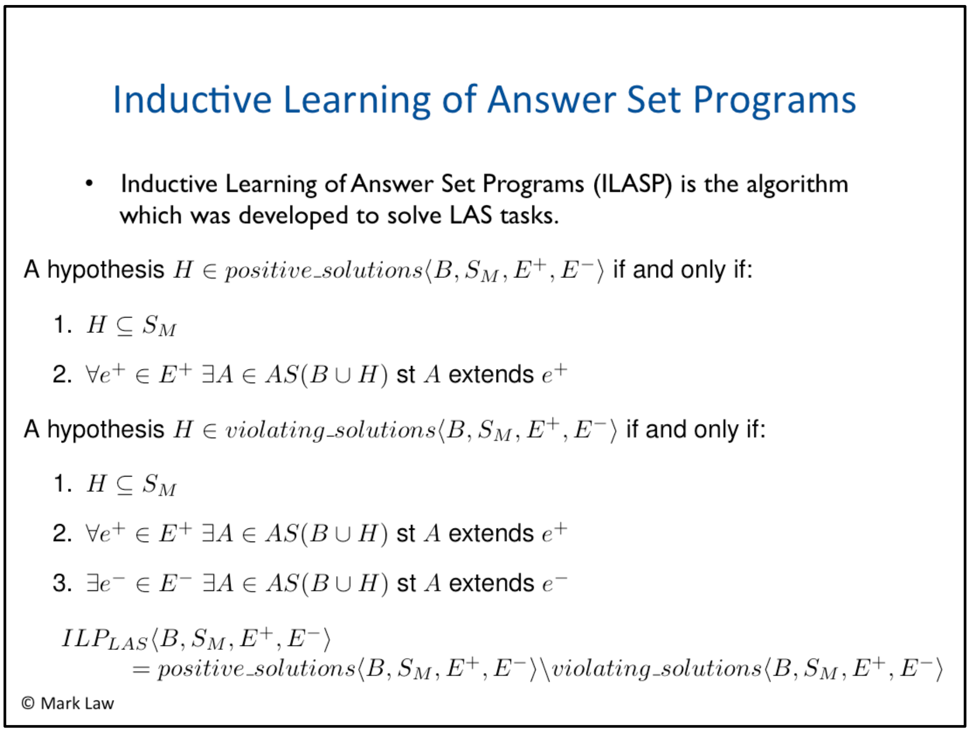
E^-

```
q
```

s.	*:- not s.* *:- q.*	*p.* *s.*	*r.*
{ p, s }, *{ q, s }*	*NONE!*	*{ p, s }*	*{p, r, s }*
✗	✗	✓	✓

What is the equivalent ***ILP***$_{LAS}$ task?

ILASP



Inductive Learning of Answer Set Programs

- Inductive Learning of Answer Set Programs (ILASP) is the algorithm which was developed to solve LAS tasks.

A hypothesis $H \in positive\_solutions\langle B, S_M, E^+, E^- \rangle$ if and only if:

1. $H \subseteq S_M$
2. $\forall e^+ \in E^+ \; \exists A \in AS(B \cup H)$ st A extends e^+

A hypothesis $H \in violating\_solutions\langle B, S_M, E^+, E^- \rangle$ if and only if:

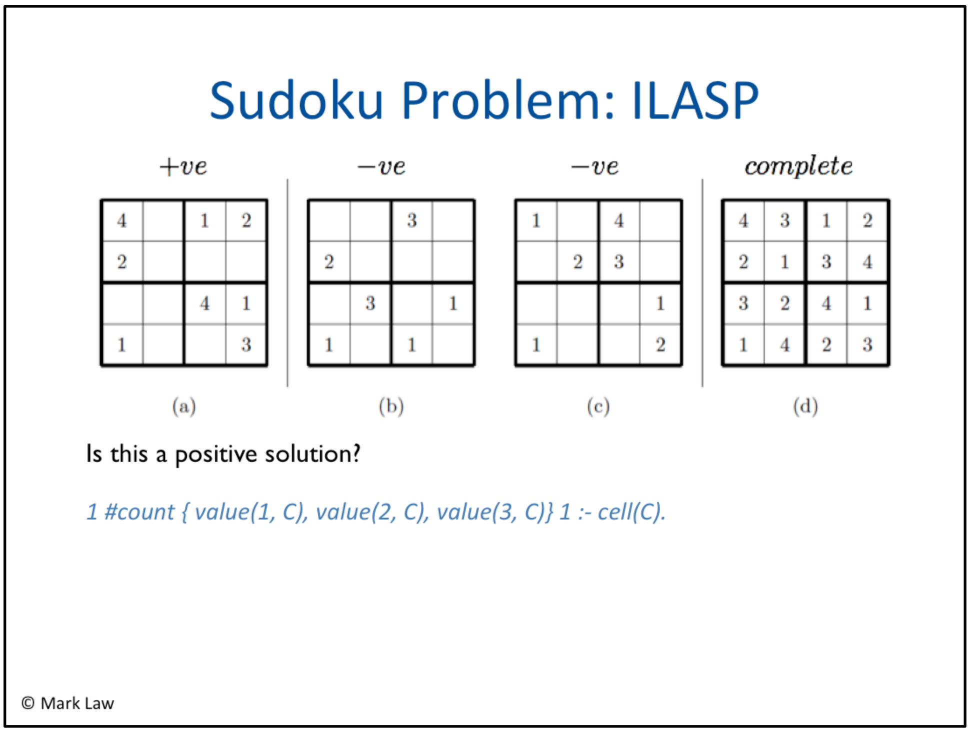
1. $H \subseteq S_M$
2. $\forall e^+ \in E^+ \; \exists A \in AS(B \cup H)$ st A extends e^+
3. $\exists e^- \in E^- \; \exists A \in AS(B \cup H)$ st A extends e^-

$$ILP_{LAS}\langle B, S_M, E^+, E^- \rangle = positive\_solutions\langle B, S_M, E^+, E^- \rangle \backslash violating\_solutions\langle B, S_M, E^+, E^- \rangle$$

Inductive Learning of Answer Set Programs (ILASP) is a sound and complete algorithm for finding the optimal inductive solutions of a give LAS task. It relies on two fundamental concepts: *positive solutions* and *violating solutions*.

Positive solutions are hypotheses which cover all of the positive examples (for each of the positive examples, there is an Answer Set of BUH which extends that positive example). The violating solutions are exactly those positive solutions which are not inductive solutions (this means that they have at least one Answer Set which extends a negative example).

The inductive solutions of any particular length *n* are equal to the positive solutions of length *n* which are not violating solutions. This is the fundamental principle which ILASP works on. It first constructs the violating solutions of length *n* and then uses these to constrain the search for positive solutions.



Sudoku Problem: ILASP

$+ve$

4		1	2
2			
		4	1
1			3

(a)

$-ve$

		3	
2			
	3		1
1		1	

(b)

$-ve$

1		4	
	2	3	
			1
1			2

(c)

$complete$

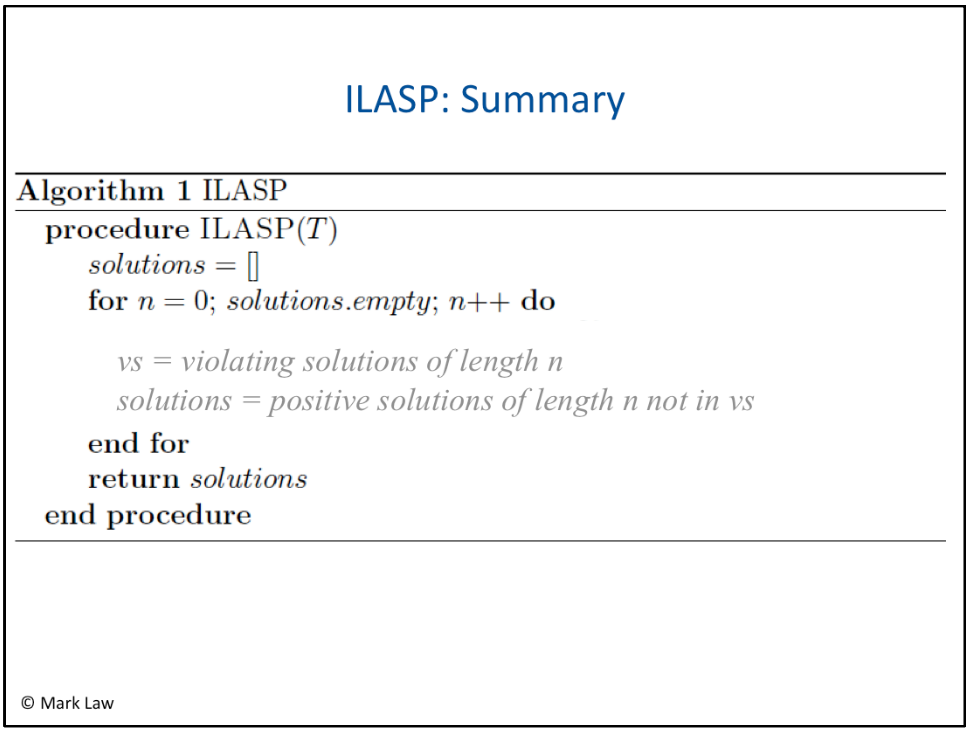
4	3	1	2
2	1	3	4
3	2	4	1
1	4	2	3

(d)

Is this a positive solution?

1 #count { value(1, C), value(2, C), value(3, C)} 1 :- cell(C).

I will upload the full version of these slides (with solutions) after the lecture.



ILASP: Summary

Algorithm 1 ILASP

```
procedure ILASP($T$)
    $solutions = []$
    for $n = 0$; $solutions.empty$; $n{+}{+}$ do
        $vs$ = violating solutions of length $n$
        $solutions$ = positive solutions of length $n$ not in $vs$
    end for
    return $solutions$
end procedure
```

Given an ILP$_{LAS}$ task and an integer *n*, ILASP is able to calculate the set of inductive solutions of *T* of length *n*. It does this by mapping *T* into a meta level representation in ASP called the *task program* T^n_{meta}. We will not consider how to construct the task program in this course, but its properties are that:

1) The Answer Sets of T^n_{meta} can be mapped to the positive solutions of the task.
2) There may be many Answer Sets of T^n_{meta} which correspond to each positive solution, but if the solution is violating then at least one of these Answer Sets will contain the atom ***violating***.
3) Therefore the Answer Sets of T^n_{meta} ***U { :-** not violating***}** can be mapped to the violating solutions of the task.

ILASP constructs the inductive solutions of length ***n*** by solving the program T^n_{meta} ***U { :-** not violating***}** and mapping the Answer Sets into the set of violating solutions of length ***n*** (***VS***). These are then converted into constraints which, when added back into the task program, rule out all of the violating solutions. Therefore when ILASP solves the task program a second time (with these new constraints and without the constraint ***:-** not violating*) the Answer Sets can be mapped into the set of positive solutions which it didn't find the first time.
These are exactly the set of inductive solutions of length ***n***. ILASP starts with length ***n*** as 0 and increments ***n*** by 1 until it finds a non-empty set of inductive solutions. The first non-empty set is the set of all optimal inductive solutions of the task ***T***.
This algorithm (like ASPAL), is sound and complete with respect to the optimal inductive solutions of its tasks. This is something of a rarity in ILP systems, which seldom complete.
For the purposes of this course, you need only know how the algorithm works (i.e. the pseudo code on the slide) and to be able to compute the posItive, violating and inductive solutions yourself; you do not need to learn the properties of the task program.
For more details see the paper *Inductive Learning of Answer Set Programs (Law, M; Russo, A; Broda, K. JELIA 2014).*

Inductive Learning of Answer Set Programs

- Consider the learning task:

B

```
vegetarian(andy).        vegetarian(bob).        person(charlie).        person(dan).
person(X) :- vegetarian(X).
food(apple).     food(bread).     meat(beef).     meat(chicken).     food(X) :- meat(X).
0 { serve(F, P) } 1 :- food(F), person(P)
```

E^+

```
<{serve(apple, andy), serve(bread, bob), serve(chicken, charlie), serve(beef, dan)}, { }>
```

E^-

```
<{serve(chicken, andy), serve(bread, bob), serve(chicken, charlie), serve(beef, dan)}, { }>.
 <{serve(bread, andy), serve(beef, bob), serve(chicken, charlie), serve(beef, dan)}, { }>.
 <{serve(bread, andy), serve(apple, andy), { }>
```

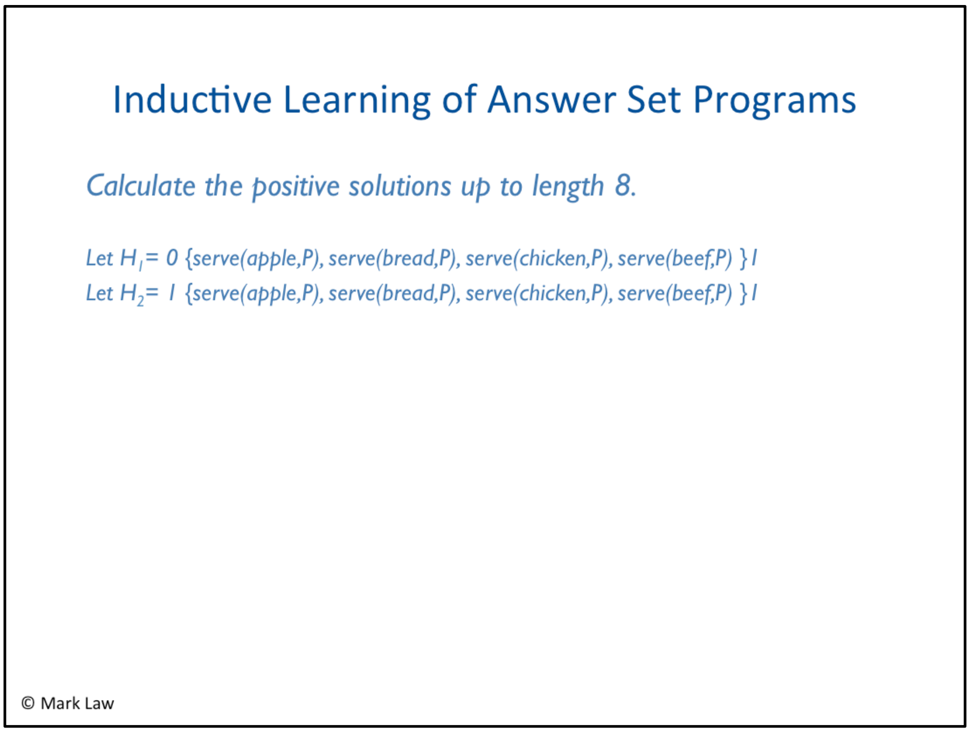
Inductive Learning of Answer Set Programs

Reconsider the hypothesis space S_M

Let $H_1 = 0 \{serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P)\} 1$

Let $H_2 = 1 \{serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P)\} 1$

S_M *contains:*

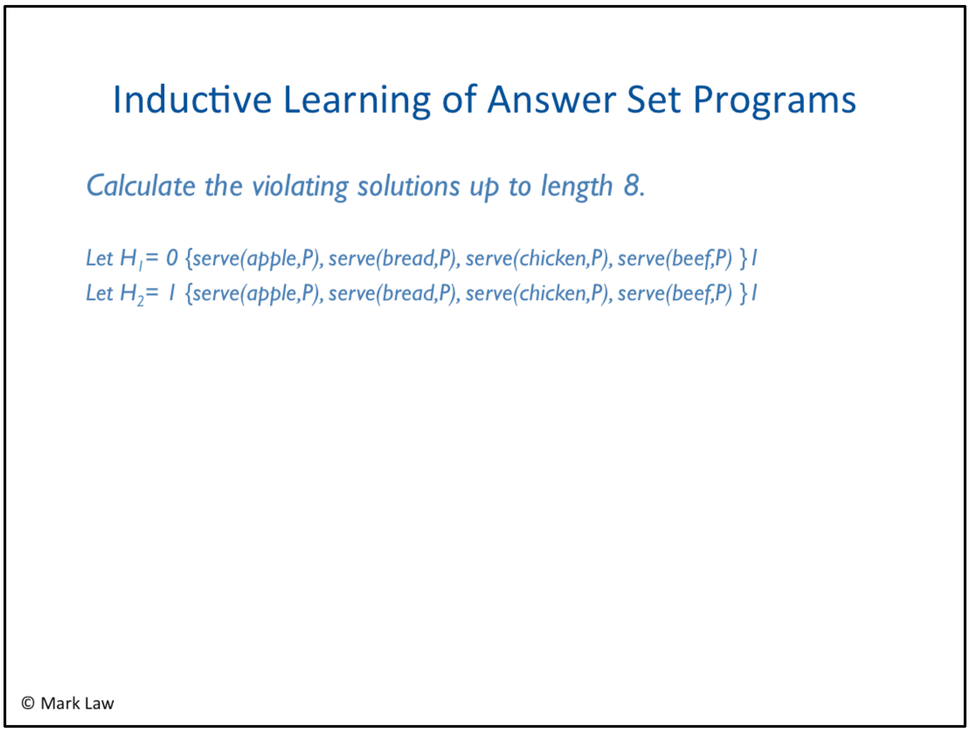


Inductive Learning of Answer Set Programs

Calculate the positive solutions up to length 8.

Let H_1= 0 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1
Let H_2= 1 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1

The slide shows the positive solutions up to length 8 (there are 8 of them). Any inductive solution must be a positive solution, but some positive solutions are violating (not inductive). We must next compute the violating solutions.

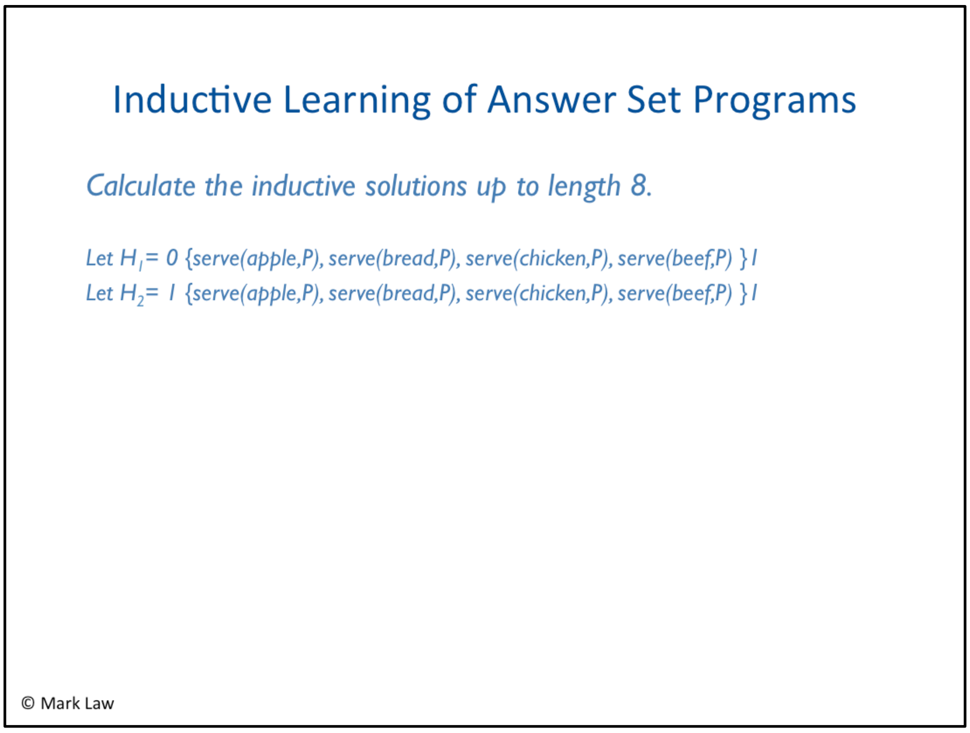


Inductive Learning of Answer Set Programs

Calculate the violating solutions up to length 8.

Let H_1= 0 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1
Let H_2= 1 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1

This slide shows the violating solutions of up to length 8. These are the positive solutions which are not inductive solutions.



Inductive Learning of Answer Set Programs

Calculate the inductive solutions up to length 8.

Let H_1= 0 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1

Let H_2= 1 {serve(apple,P), serve(bread,P), serve(chicken,P), serve(beef,P) } 1

Having computed both the positive and the violating solutions we can not compute the inductive solutions be simply finding those positive solutions which are not violating; these are shown on the slide.

Summary

- Motivated the need for a learning task incorporating both Brave and Cautious reasoning

- Defined
  - Partial Interpretations
  - Learning from Answer Sets
  - Positive Solutions
  - Violating Solutions

- We saw an algorithm (ILASP) to generate the optimal solutions of any Learning from Answer Sets task