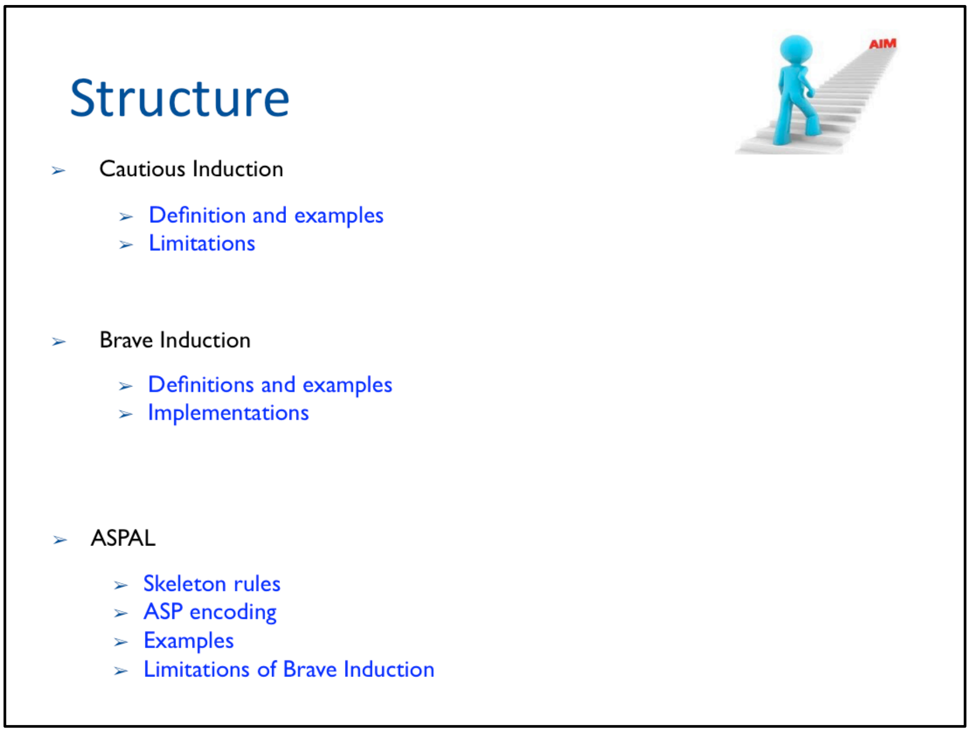
Brave and Cautious Induction

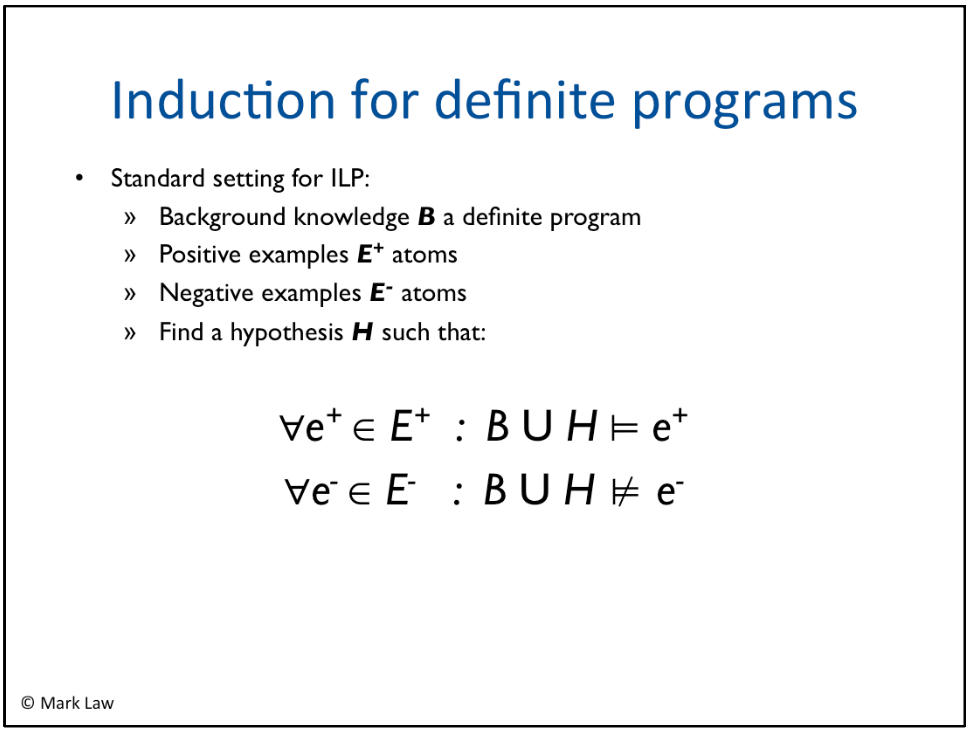
Mark Law
mark.law09@imperial.ac.uk



Structure

- Cautious Induction
  - Definition and examples
  - Limitations
- Brave Induction
  - Definitions and examples
  - Implementations
- ASPAL
  - Skeleton rules
  - ASP encoding
  - Examples
  - Limitations of Brave Induction

In this part of the course, we will consider the non-monotonic learning under the Answer Set/Stable Model Semantics. In this lecture we look at early approaches to learning under the Answer Set Semantics - brave and cautious induction - and study the algorithm ASPAL which maps its ILP task to an ASP program which can then be solved for optimal hypotheses.



Induction for definite programs

- Standard setting for ILP:
  - » Background knowledge ***B*** a definite program
  - » Positive examples ***E⁺*** atoms
  - » Negative examples ***E⁻*** atoms
  - » Find a hypothesis ***H*** such that:

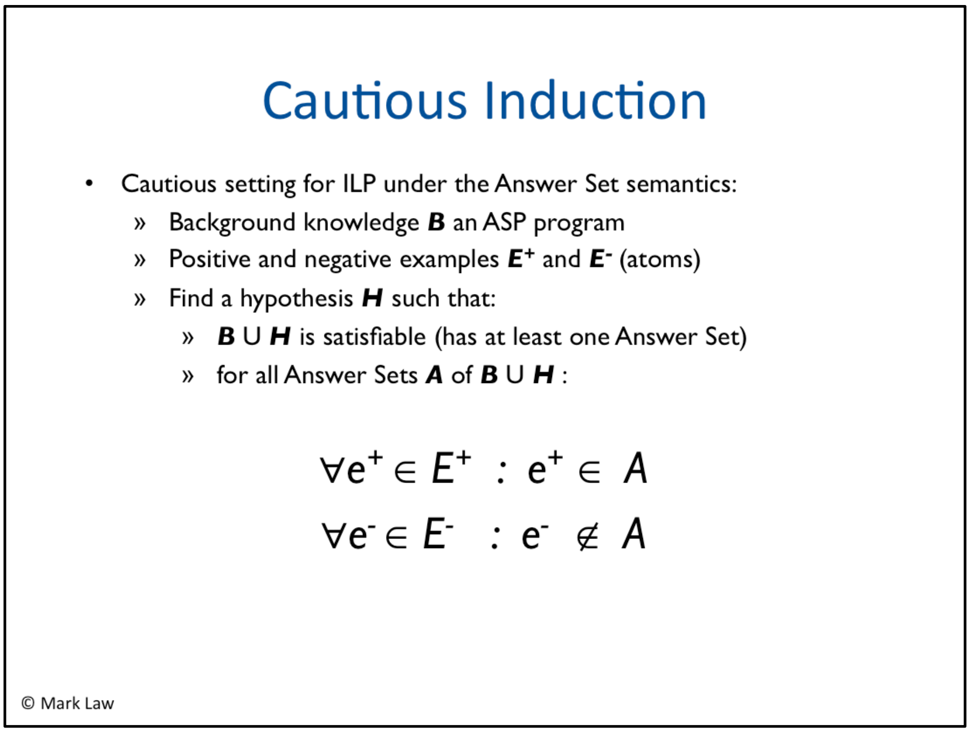
$$\forall e^{+} \in E^{+} \ : \ B \cup H \models e^{+}$$
$$\forall e^{-} \in E^{-} \ : \ B \cup H \not\models e^{-}$$

For a definite program *P*, as there is always a unique least Herbrand model (written ***M(P)***), entailment is defined in terms of this least Herbrand model. The task of ILP therefore is to find a hypothesis ***H*** such that ***M(B ∪ H)*** contains all of a set of positive examples and none of a set of negative examples.

When considering Answer Set programs, as we saw in previous lectures, there are two different kinds of entailment: **brave** and **cautious**. In this lecture, we will see the two ILP tasks based on these different types of entailment.

In subsequent lectures, we will then see a new ILP task which is capable of expressing both brave and cautious induction.

Cautious Induction



Cautious Induction

- Cautious setting for ILP under the Answer Set semantics:
  - » Background knowledge ***B*** an ASP program
  - » Positive and negative examples ***E***$^+$ and ***E***$^-$ (atoms)
  - » Find a hypothesis ***H*** such that:
    - » ***B*** ∪ ***H*** is satisfiable (has at least one Answer Set)
    - » for all Answer Sets ***A*** of ***B*** ∪ ***H*** :

$$\forall e^+ \in E^+ \; : \; e^+ \in A$$
$$\forall e^- \in E^- \; : \; e^- \notin A$$

Cautious induction is based on the cautious semantics of ASP. As in the usual setting for ILP we have sets of positive and negative examples called atoms (***E***$^+$ and ***E***$^-$). We also have a background knowledge ***B*** which is an ASP program. We search for a hypothesis ***H*** such that ***B*** ∪ ***H*** is satisfiable (has at least one Answer Set) and every Answer Set of ***B*** ∪ ***H*** contains all of the positive examples and contains none of the negative examples. We denote such an ILP task as the tuple ***<B, E***$^+$***, E***$^-$ ***>***.

In other words, we search for a hypothesis such that ***B*** ∪ ***H*** cautiously entails the positive examples and the negation of each of the negative examples (***B*** ∪***H*** $\models_c$ **(e**$_1^+$ ∧ **…** ∧ **e**$_n^+$ ∧ **(not e**$_1^-$**)** ∧ **…** ∧ **(not e**$_m^-$**))**).

We write *ILP*$_c$***<B, E***$^+$***, E***$^-$ ***>*** to denote the set of hypotheses which are cautious inductive solutions of the task ***<B, E***$^+$***, E***$^-$ ***>.***

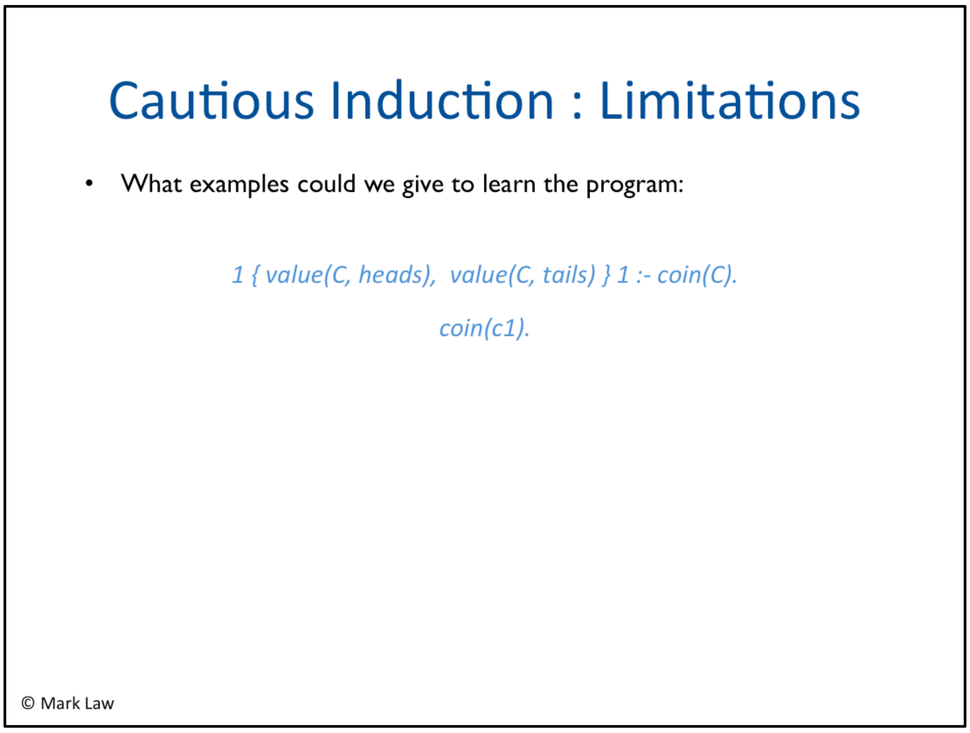
It should be noted that the original definition of Cautious Induction (Sakama 2008) had no concept of negative examples. Here, we have presented a more general definition.

Cautious Induction : Example

B	E^+	E^-
p :- not q. ***q :- not p, not r.*** ***s :- r.***	***s***	***q***

- Which of the following hypotheses are cautious inductive solutions?

s.	***:- not s.*** ***:- q.***	***p.*** ***s.***	***r.***



Cautious Induction : Limitations

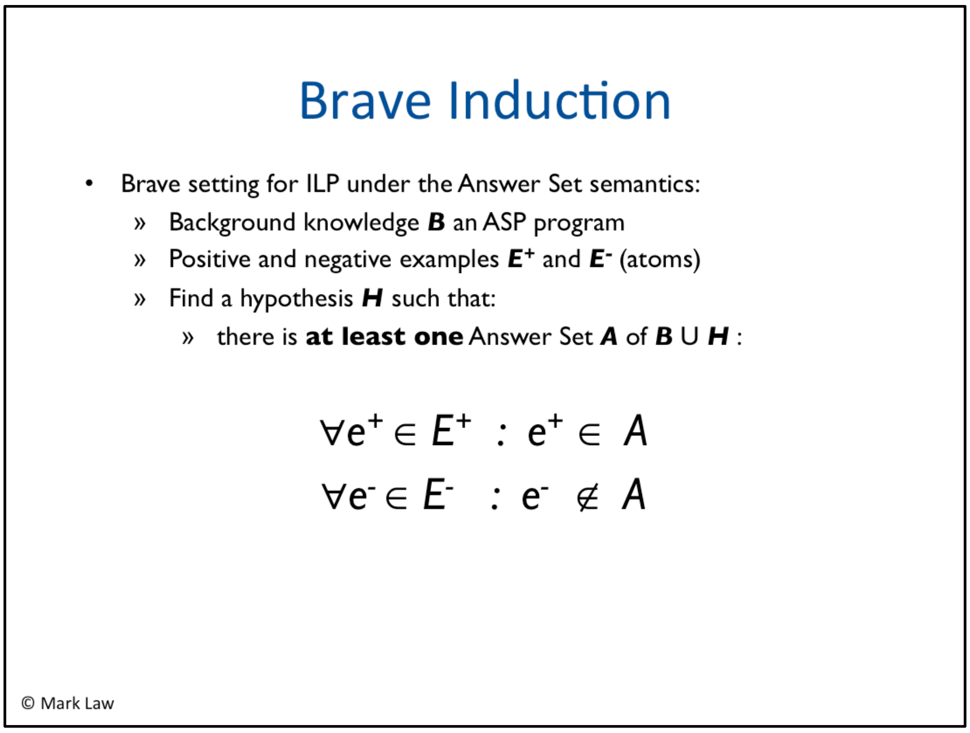
- What examples could we give to learn the program:

1 { value(C, heads), value(C, tails) } 1 :- coin(C).

coin(c1).

Consider an empty background knowledge. We cannot construct a set of examples such that any of the shortest hypotheses are: ***{ 1 #count { value(C, heads); value(C, tails) } 1 :- coin(C).   coin(c1). }***. This is because the only atom which is true in all Answer Sets of the program we are trying to learn is **coin(c1)**, and neither of the atoms ***value(c1, heads)*** or ***value(c1, tails)*** is false in all Answer Sets. The only relevant example we can give is therefore the positive example ***coin(c1).*** This would cause us to learn ***coin(c1).*** as our hypothesis which has the single Answer Set ***coin(c1)***. This is not what we are aiming for at all; we want to learn a program with two distinct Answer Sets corresponding to the coin being *heads* or *tails*. Cautious entailment of all examples in this case is too strong a requirement. We need to be able to give examples of what is true in *some* Answer Sets but *not all* Answer Sets of the learned program.

Brave Induction



Brave Induction

- Brave setting for ILP under the Answer Set semantics:
  - » Background knowledge ***B*** an ASP program
  - » Positive and negative examples ***E***$^+$ and ***E***$^-$ (atoms)
  - » Find a hypothesis ***H*** such that:
    - » there is **at least one** Answer Set ***A*** of **B** ∪ ***H*** :

$$\forall e^+ \in E^+ \; : \; e^+ \in A$$
$$\forall e^- \in E^- \; : \; e^- \notin A$$

Brave induction is based on the brave semantics of ASP. We again have sets of atoms called the positive and negative examples (E^+ and E^-) and a background knowledge ***B*** which is an ASP program. We search for a hypothesis ***H*** such that ***B*** ∪***H*** has at least one Answer Set which contains all of the positive examples and contains none of the negative examples. We denote such an ILP task as the tuple ***<B, E***$^+$***, E***$^-$ ***>***.

In other words, we search for a hypothesis such that ***B*** ∪***H*** bravely entails the positive examples and the negation of each of the negative examples ($B \cup H \models_b (e_1^+ \wedge \ldots \wedge e_n^+ \wedge (\text{not } e_1^-) \wedge \ldots \wedge (\text{not } e_m^-))$).

We write $ILP_b\langle B, E^+, E^- \rangle$ to denote the set of hypotheses which are brave inductive solutions of the task $\langle B, E^+, E^- \rangle$.

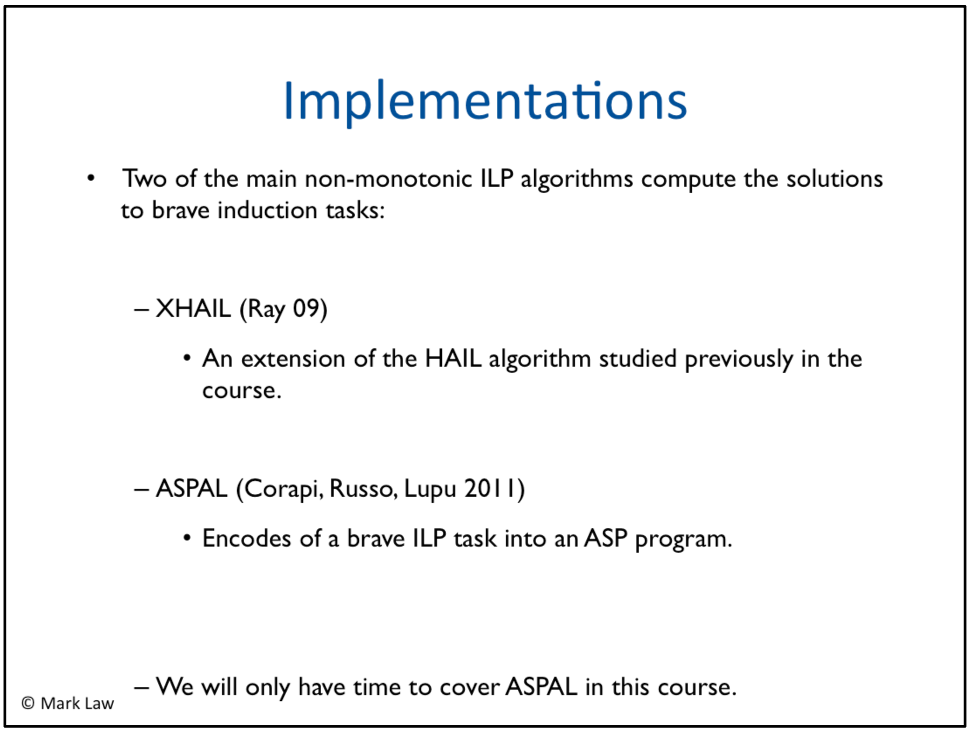
Similarly to cautious induction, it should be noted that the original definitions of Brave Induction (Sakama 2008) had no concept of negative examples. We present the more general definition here as it coincides with the task used by the algorithm we will study.

Brave Induction : Example

B	E^+	E^-
p :- not q. *q :- not p, not r.* *s :- r.*	*s*	*q*

- Which of the following hypotheses are cautious inductive solutions?

s.	***:- not s.*** ***:- q.***	***p.*** ***s.***	***r.***



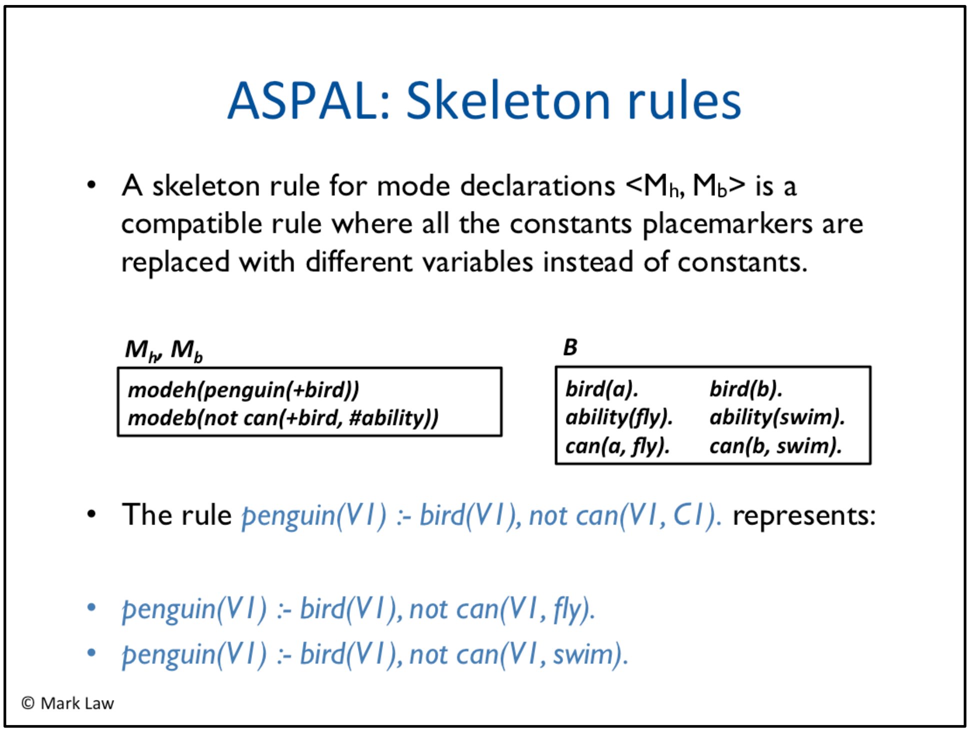
Implementations

- Two of the main non-monotonic ILP algorithms compute the solutions to brave induction tasks:
  - XHAIL (Ray 09)
    - An extension of the HAIL algorithm studied previously in the course.
  - ASPAL (Corapi, Russo, Lupu 2011)
    - Encodes of a brave ILP task into an ASP program.
  - We will only have time to cover ASPAL in this course.

Two of the main non-monotonic ILP systems which have been implemented, both compute the solutions to a brave induction task. These are XHAIL - an extension of the HAIL system studied earlier in the course - and ASPAL. We will study ASPAL in this course.

ASPAL solves the ILP task by encoding the search as an ASP program.

ASPAL



ASPAL: Skeleton rules

- A skeleton rule for mode declarations <M_h, M_b> is a compatible rule where all the constants placemarkers are replaced with different variables instead of constants.

M_h, M_b

```
modeh(penguin(+bird))
modeb(not can(+bird, #ability))
```

B

```
bird(a).        bird(b).
ability(fly).   ability(swim).
can(a, fly).    can(b, swim).
```

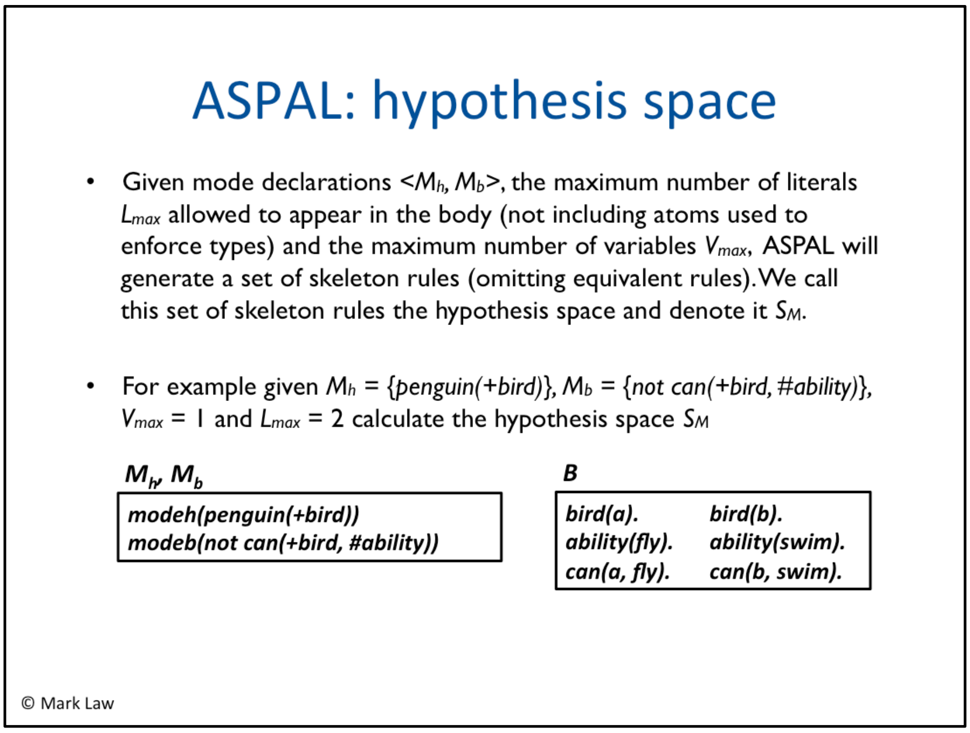
- The rule *penguin(V1) :- bird(V1), not can(V1, C1).* represents:

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

© Mark Law

For every variable, ASPAL will add the type of the variable to the body of the rule (as on the slide with ***bird(V1)***). This enforces that the rules only apply to terms with these types and also means that ASPAL does not have to worry about the safety of rules (all variables will certainly appear in at least one positive literal in the body). These additional atoms do not count towards the length of the rules.

By using skeleton rules where the constant symbols are not yet ground, ASPAL shifts the burden of finding the ground rules to the ASP solver. Each skeleton rule represents a set of rules where the variables representing constants have been replaced with constants of the correct types.



ASPAL: hypothesis space

- Given mode declarations <M_h, M_b>, the maximum number of literals L_{max} allowed to appear in the body (not including atoms used to enforce types) and the maximum number of variables V_{max}, ASPAL will generate a set of skeleton rules (omitting equivalent rules). We call this set of skeleton rules the hypothesis space and denote it S_M.
- For example given M_h = {*penguin(+bird)*}, M_b = {*not can(+bird, #ability)*}, V_{max} = 1 and L_{max} = 2 calculate the hypothesis space S_M

M_h, M_b

```
modeh(penguin(+bird))
modeb(not can(+bird, #ability))
```

B

```
bird(a).        bird(b).
ability(fly).   ability(swim).
can(a, fly).    can(b, swim).
```

Given a set of mode declarations <Mh, Mb>, and positive integers Lmax and Vmax specifying the number of literals allowed to appear in the body of the rule, and the number of unique variables allowed in the rule, ASPAL constructs a set of skeleton rules SM. We call this set the hypothesis space.

We say that a set of skeleton rules SM is maximal given <Mh, Mb>, Lmax and Vmax, if any rule which can be constructed respecting <Mh, Mb>, Lmax and Vmax is equivalent to a rule already in SM.

The hypothesis space constructed by ASPAL is maximal. We will not cover ASPAL's algorithm for generating a maximal hypothesis space, but you will be expected to construct a maximal hypothesis space for given mode declarations (this does not have to be the same as the one generated by ASPAL).

ASPAL: hypothesis space

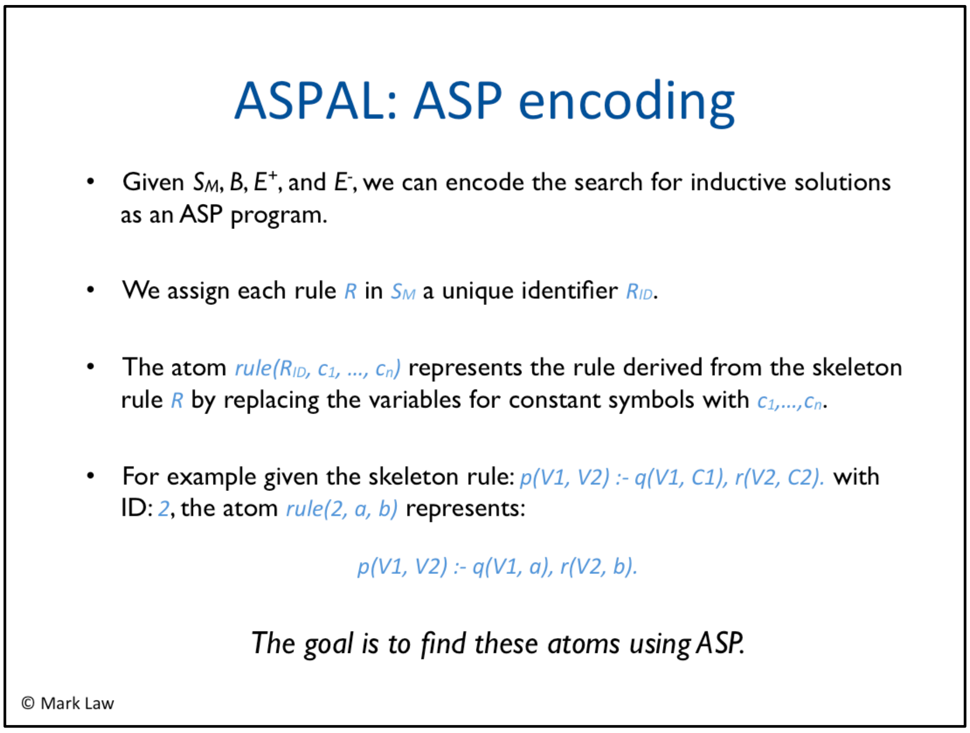
S_M

```
penguin(V1) :- bird(V1).
penguin(V1) :- bird(V1), not can(V1, C1).
penguin(V1) :- bird(V1), not can(V1, C1),
                         not can(V1, C2).
```

B

```
bird(a).         bird(b).
ability(fly).    ability(swim).
can(a, fly).     can(b, swim).
```

- Write down all possible rules represented by these skeleton rules:



ASPAL: ASP encoding

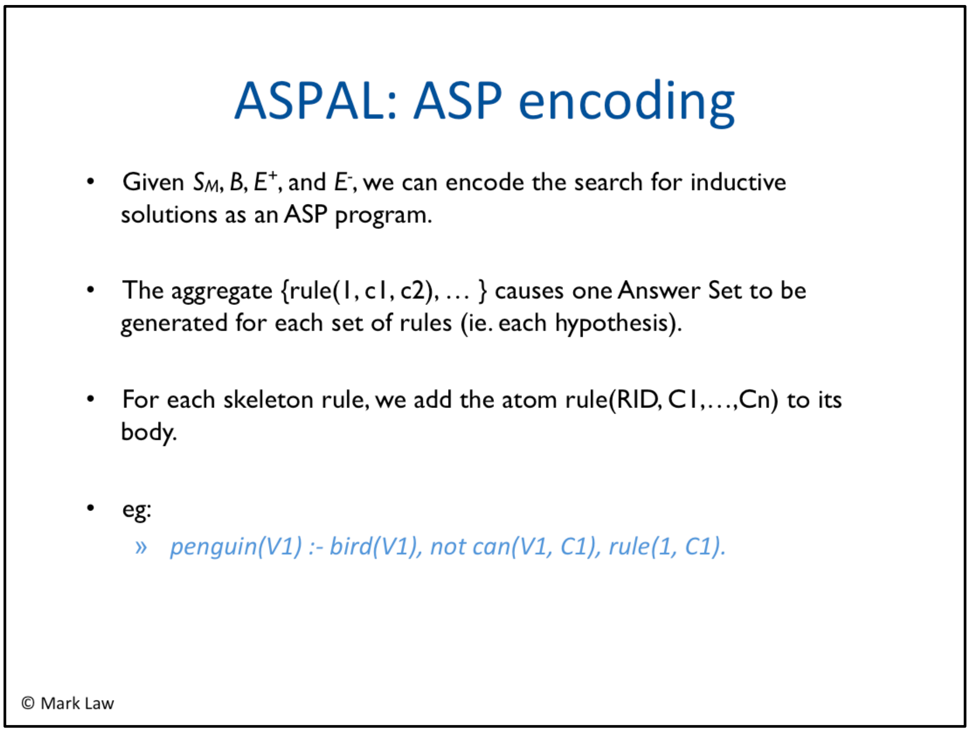
- Given S_M, B, E^+, and E^-, we can encode the search for inductive solutions as an ASP program.
- We assign each rule *R* in S_M a unique identifier R_{ID}.
- The atom *rule*(R_{ID}, c_1, …, c_n) represents the rule derived from the skeleton rule *R* by replacing the variables for constant symbols with c_1,…,c_n.
- For example given the skeleton rule: *p(V1, V2) :- q(V1, C1), r(V2, C2).* with ID: *2*, the atom *rule(2, a, b)* represents:

p(V1, V2) :- q(V1, a), r(V2, b).

The goal is to find these atoms using ASP.

ASPAL encodes an ILP task as a meta level ASP program. The Answer Sets contain atoms representing each of the rules in the hypothesis. This means that we can map each Answer Set of the meta level program back to an inductive solution of the task.

Each skeleton rule *R* in the hypothesis space S_M we assign a unique identifier R_{ID}. The atom *rule*(R_{ID}, c_1, …, c_n) represents the skeleton rule *R* with each of the constant variables replaced with the constants c_1, …, c_n.

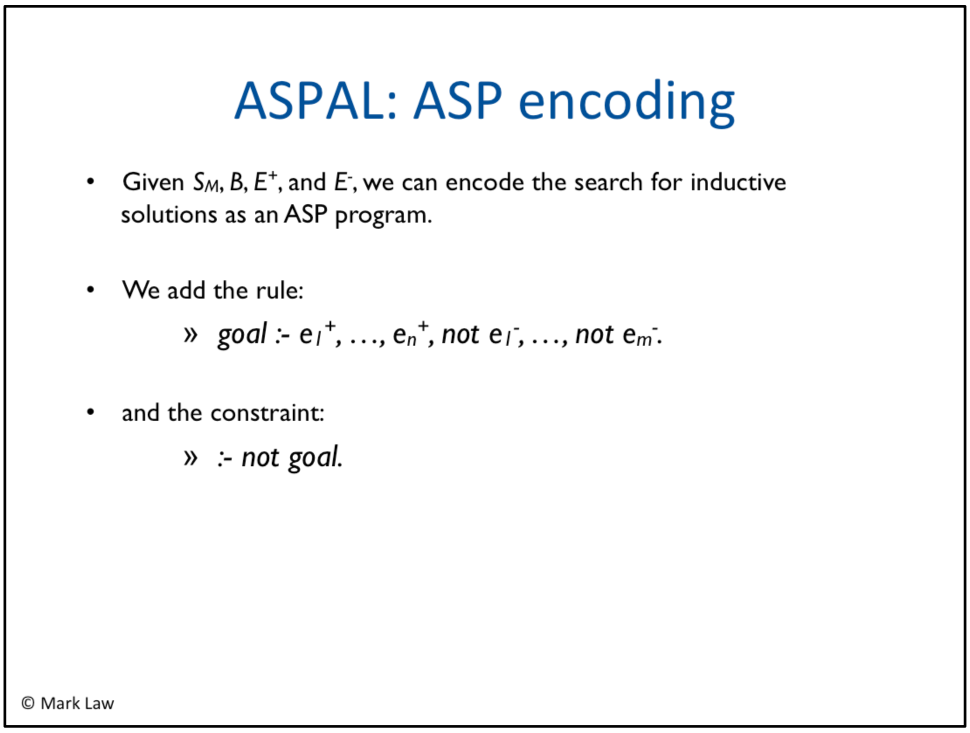


ASPAL: ASP encoding

- Given S_M, B, E^+, and E^-, we can encode the search for inductive solutions as an ASP program.
- The aggregate {rule(1, c1, c2), … } causes one Answer Set to be generated for each set of rules (ie. each hypothesis).
- For each skeleton rule, we add the atom rule(RID, C1,…,Cn) to its body.
- eg:
  - » *penguin(V1) :- bird(V1), not can(V1, C1), rule(1, C1).*

The aggregate, *{rule(1, c1, c2); … }*, causes one Answer Set to be generated for each hypothesis which could be constructed from the skeleton rules.

By adding *rule*(R_{ID}, C_1,…,C_n) to each rule R, we cause only the rules in the hypothesis represented by a particular Answer Set of the meta program to be effective. For any hypothesis H, for each Answer Set A of $B \cup H$, there is an Answer Set A' of the meta program which contains all of the atoms of A in addition to the atoms *rule*(R_{ID}, C_1,…,C_n) which represent the hypothesis H.



ASPAL: ASP encoding

- Given S_M, B, E^+, and E^-, we can encode the search for inductive solutions as an ASP program.
- We add the rule:
  - » *goal* :- $e_1^+, \ldots, e_n^+$, *not* $e_1^-, \ldots,$ *not* e_m^-.
- and the constraint:
  - » *:- not goal.*

By adding the rules:

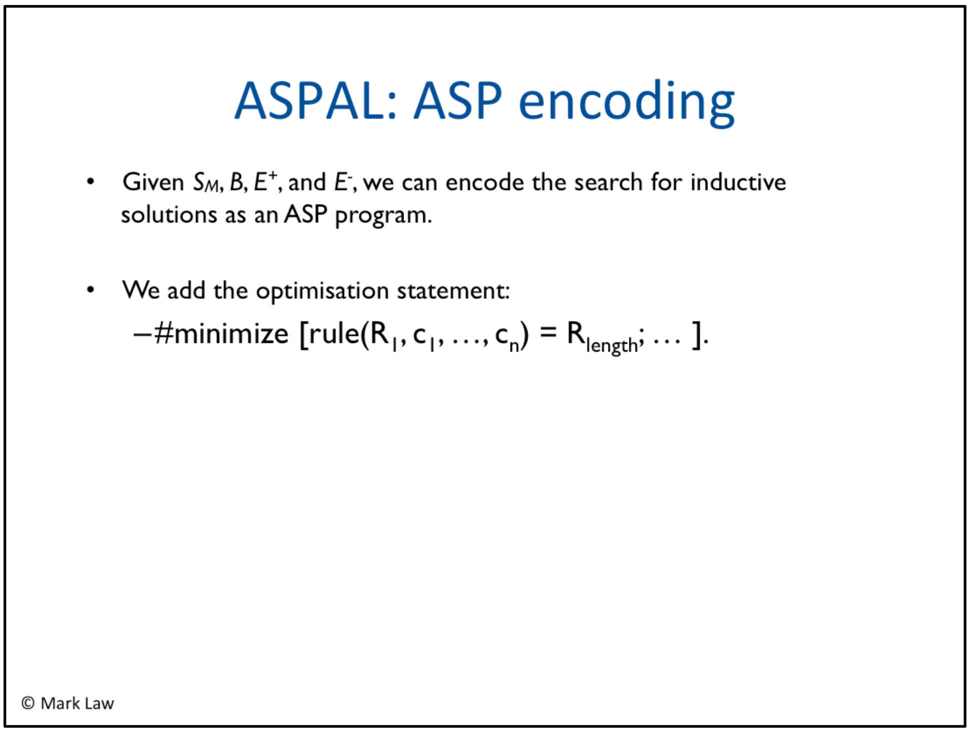
goal :- ***e₁⁺, …, eₙ⁺, not e₁⁻, …, not eₘ⁻***.

:- not goal.

we rule out any Answer Set of the meta program which corresponds to an Answer Set of BUH which does not contain all of the positive examples or contains a negative example.

This means that if there is an Answer Set of the meta program corresponding to a particular hypothesis *H*, then *BUH* must have at least one Answer Set which contains all of the positive examples or none of the negative examples (otherwise, all Answer Sets of the meta program which correspond to *H* would have been rules out by the goal constraint).

The result of this is that the Answer Sets of the meta program correspond exactly to the inductive solutions of the task. This is exactly how ASPAL computes these solutions.



ASPAL: ASP encoding

- Given S_M, B, E^+, and E^-, we can encode the search for inductive solutions as an ASP program.
- We add the optimisation statement:
  - #minimize [rule(R_1, c_1, …, c_n) = R_{length}; …].

ASPAL uses an optimisation statement such that the optimal Answer Sets of the meta level program correspond exactly to the optimal inductive solutions of the task. It does this by weighting each of the rules by its length.

ASPAL: ASP encoding example

B

bird(a).	***bird(b).***
ability(fly).	***ability(swim).***
can(a, fly).	***can(b, swim).***

E^+

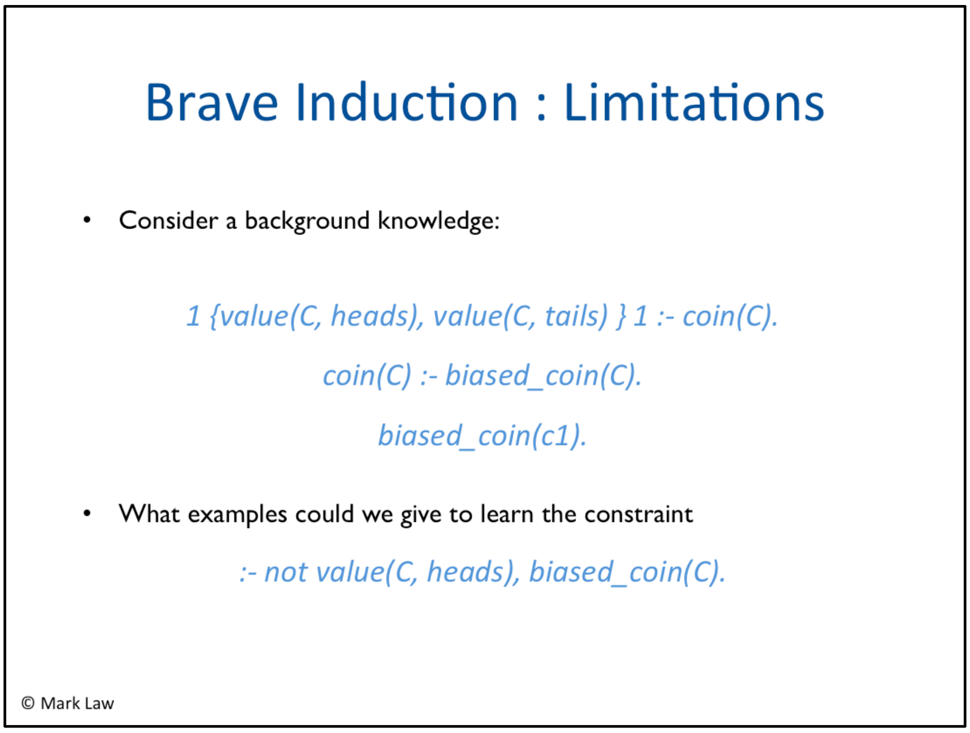
penguin(b)

E^-

penguin(a)

```
% Background
bird(a). bird(b).
ability(fly). ability(swim).
can(a, fly). can(b, swim).
% Skeleton Rules
penguin(V1) :- bird(V1), rule(1).
penguin(V1) :- bird(V1), not can(V1, C1), rule(2, C1).
penguin(V1) :- bird(V1), not can(V1, C1), not can(V1, C2), rule(3, C1, C2).
% Generate Hypotheses
{rule(1),  rule(2, fly),  rule(2, swim), rule(3, fly, swim) }.
#minimise [ rule(1)=1,  rule(2, fly)=2,  rule(2, swim)=2, rule(3, fly, swim)=3 ].
```

```
% Examples
goal :- penguin(b), not penguin(a).
:- not goal.
```



Brave Induction : Limitations

- Consider a background knowledge:

1 {value(C, heads), value(C, tails) } 1 :- coin(C).

coin(C) :- biased_coin(C).

biased_coin(c1).

- What examples could we give to learn the constraint

:- not value(C, heads), biased_coin(C).

Some hypotheses cannot be learned using brave induction alone; for example, brave induction cannot be used to learn any hypothesis which only rules out Answer Sets and does not generate anything new.

In particular, a brave induction task will never have as an optimal solution, a hypothesis containing a constraint. This is because, if the hypothesis is a solution for the brave induction task, then it must have an Answer Set which contains all of the positive and none of the negative examples. As constraints only rule out Answer Sets, this Answer Set will still be an Answer Set of the program without the constraint. In fact, one of the advanced questions on your tutorial sheet is to prove this property.

Summary

- Defined
  - Cautious Induction
  - Brave Induction
- Seen an algorithm (ASPAL) which realises Brave Induction by encoding the search as a meta level ASP program
- Seen that not all ASP programs can be learned using either Brave or Cautious Induction.

- Next time: we will see a framework which is capable of learning *any* ASP program consisting of normal rules, choice rules and constraints.