Imperial College London

Inductive Learning of Answer Set Programs

Mark Law, Alessandra Russo and Krysia Broda

Imperial College London

Overview

- Introduction, motivating example and state of the art
- Our learning task (Learning from Answer Sets)
- Our learning algorithm (ILASP)
- Comparison to related work
- Application to learning while planning
- Current and future work

Inductive Logic Programming

The task of Inductive Logic Programming (ILP) is to find a hypothesis *H* which, with respect to a background knowledge *B*, entails a set of positive examples E^+ and does not entail any negative examples E^-.

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

Most of the previous work on ILP has addressed the learning of monotonic (definite) logic programs where entailment is defined in terms of the least Herbrand model.

Inductive Logic Programming

Under the Answer Set (or stable model) semantics a set of normal clauses can have one, many or even no Answer Sets.

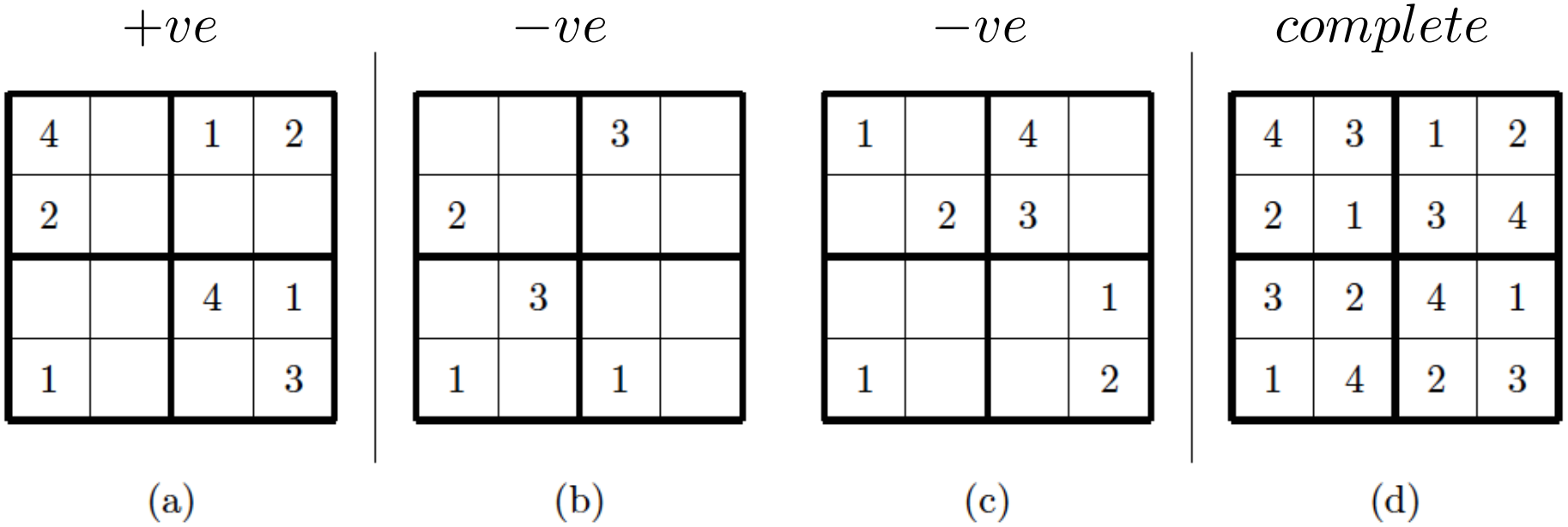
We say that a formula is **bravely entailed** by a program *P* if it is true in **at least one** Answer Set of *P*. Conversely a formula is **cautiously entailed** by a program *P* if it is true in **all** Answer Sets of *P*.

Previous work on nonmonotonic ILP under the Answer Set/Stable Model semantics has mostly been restricted to **either** brave **or** cautious reasoning.

Our new learning task, *Learning from Answer Sets*, incorporates **both** brave **and** cautious reasoning with the aim of learning Answer Set Programs containing normal rules, choice rules and constraints.

Imperial College London

Sudoku Example



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).
```

Imperial College London

Comparison with related works under the Answer Set semantics

Learning Task	Normal Rules	Choice Rules	Constraints	Classical negation	Brave	Cautious	Algorithm for optimal solutions
Brave Induction [Sakama, Inoue 2009]	✔	✔	✖	✔	✔	✖	✖
Cautious Induction [Sakama, Inoue 2009]	✔	✔	✖	✔	✖	✔	✖
XHAIL [Ray 2009] & ASPAL [Corapi, Russo, Lupu 2011]	✔	✖	✖	✖	✔	✖	✔
Induction of Stable Models [Otero 2001]	✔	✖	✖	✖	✔	✖	✖
Induction from Answer Sets [Sakama 2005]	✔	✖	✔	✔	✔	✔	✖
LAS	✔	✔	✔	✖	✔	✔	✔

Learning from Answer Sets

A partial interpretation E is a pair of sets of atoms $\langle E^{inc}, E^{exc} \rangle$ called the *inclusions* and *exclusions* respectively.

An Answer Set A *extends* $\langle E^{inc}, E^{exc} \rangle$ if and only if: $E^{inc} \subseteq A$ and $E^{exc} \cap A = \emptyset$.

A *Learning from Answer Sets task* is a tuple $T = \langle B, S_M, E^+, E^- \rangle$ where B is an ASP program, S_M is the search space defined by a language bias M, E^+ and E^- are sets of partial interpretations.

A hypothesis $H \in ILP_{LAS} \langle B, S_M, E^+, E^- \rangle$ 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. $\forall e^- \in E^- \; \nexists A \in AS(B \cup H)$ st A extends e^-

What is an Optimal Hypothesis?

Previous algorithms for ILP have searched for *optimal* hypotheses which are inductive solutions of the task.

This is usually (but not always) defined in terms of the length of the hypothesis.

For normal rules, this is obvious: we just count the number of literals in the hypothesis.

$$H = \left\{ \begin{array}{l} p \leftarrow q, \text{not } s. \\ q \leftarrow r. \end{array} \right.$$

The length of *H* is 5.

What is an Optimal Hypothesis?

But what about the hypotheses:

$$H_1 = 0\{p, q\}2. \qquad H_2 = 1\{p, q\}1.$$

Both have 2 literals, but the first generates twice as many Answer Sets as the second.

To calculate the length of a hypothesis with choice rules, we first convert the aggregates into disjunctive normal form:

$$H_1 = (p \wedge q) \vee (p \wedge \text{not } q) \vee (\text{not } p \wedge q) \vee (\text{not } p \wedge \text{not } q).$$

$$H_2 = (p \wedge \text{not } q) \vee (\text{not } p \wedge q).$$

This gives the first hypothesis length 8 and the second length 4.

Inductive Learning of Answer Set Programs

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^+

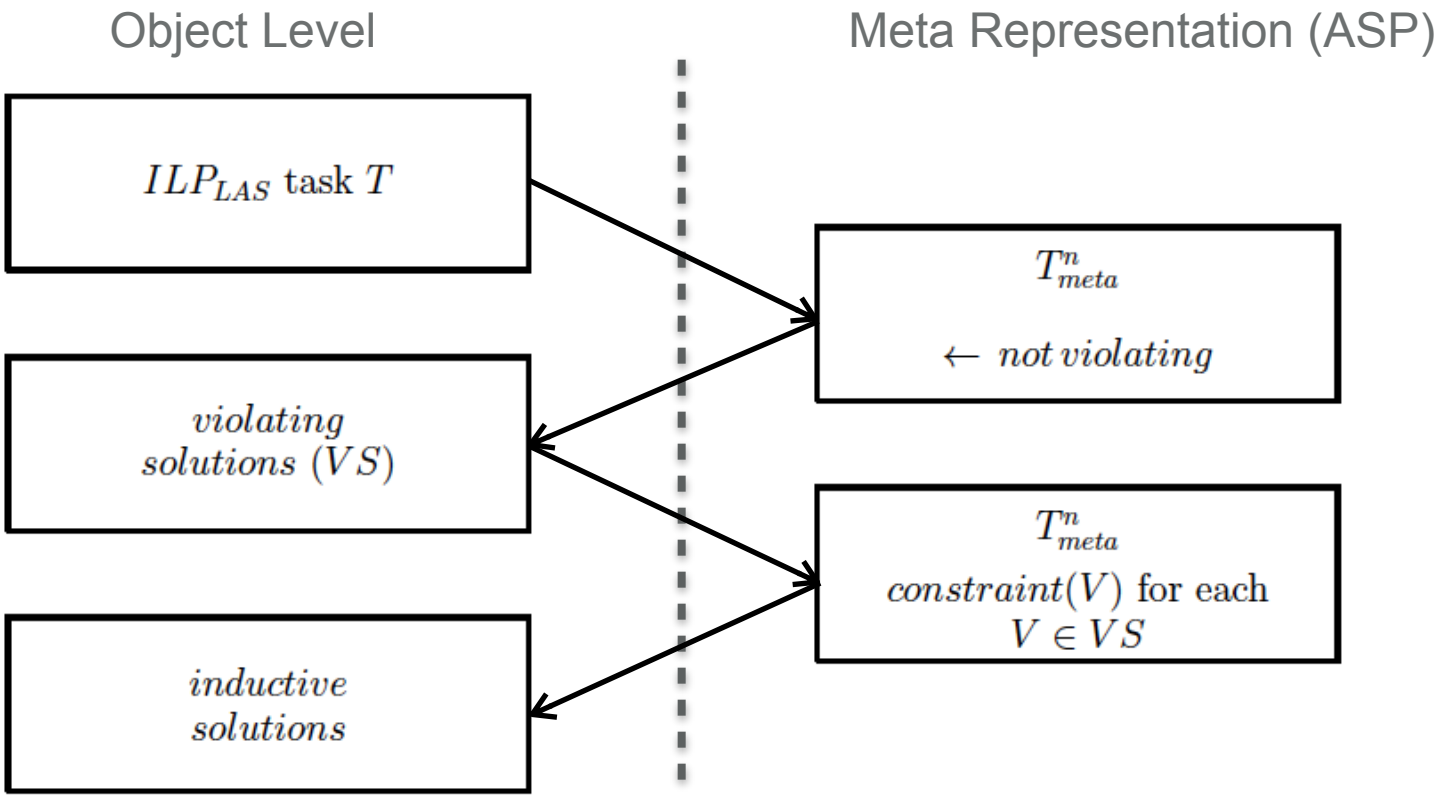
A hypothesis $H \in violating\_solutions\langle B, S_M, E^+, E^- \rangle$ if and only if:

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^-

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

Imperial College London

Inductive Learning of Answer Set Programs



n a given hypothesis length

T^n_{meta}: ASP task program (a meta representation of the task T)

Constructing the task program T_{meta}

$$B = p \leftarrow \text{not } q$$

$$S_M = \left\{ \begin{array}{l} q \leftarrow \\ q \leftarrow \text{not } p \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right.$$

$$E^- = \left\{ \begin{array}{l} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{array} \right.$$

Constructing the task program T_{meta}

$$B = p \leftarrow \text{not } q$$

$$S_M = \left\{ \begin{array}{l} q \leftarrow active(1) \\ q \leftarrow \text{not } p, active(2) \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right.$$

$$E^- = \left\{ \begin{array}{l} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{array} \right.$$

$$n\{active(R) = X : length(R, X)\}n; \qquad length(1, 1); \qquad length(2, 2)$$

Each rule R is given a unique identifier R_{id}.
$active(R_{id})$ means that R is in our hypothesis.

Constructing the task program T_{meta}

$$B = e(p, X) \leftarrow ex(X), \text{not } e(q, X)$$

$$S_M = \left\{ \begin{array}{l} e(q, X) \leftarrow ex(X), active(1) \\ e(q, X) \leftarrow ex(X), \text{not } e(p, X), active(2) \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{lll} ex(1); & covered(1) \leftarrow e(p, 1), \text{not } e(q, 1); & \leftarrow \text{not } covered(1) \\ ex(2); & covered(2) \leftarrow e(q, 2), \text{not } e(p, 2); & \leftarrow \text{not } covered(2) \end{array} \right.$$

$$E^- = \left\{ \begin{array}{ll} ex(negative); & violating \leftarrow \text{not } e(p, negative), \text{not } e(q, negative) \\ & violating \leftarrow e(p, negative), e(q, negative) \end{array} \right.$$

$$n\{active(R) = X : length(R, X)\}n; \qquad length(1, 1); \qquad length(2, 2)$$

Each Answer Set of T^n_{meta} has many $ex(X)$ atoms. Each X is "assigned" an Answer Set of $B \cup H$. $e(A, X)$ means A is in the Answer Set "assigned" to X.

Constructing the task program T_{meta}

$$B = p \leftarrow \text{not } q$$

$$S_M = \left\{ \begin{array}{l} q \leftarrow \\ q \leftarrow \text{not } p \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right. \qquad E^- = \left\{ \begin{array}{l} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{array} \right.$$

When $n = 1$, we must choose the first rule as our hypothesis H.
$B \cup H$ has one Answer Set: $\{q\}$.

Constructing the task program T_{meta}

The only Answer Set of $B \cup H$ is $\{q\}$.

$$E^+ = \left\{ \begin{array}{lll} ex(1); & covered(1) \leftarrow e(p,1), \text{not } e(q,1); & \leftarrow \text{not } covered(1) \\ ex(2); & covered(2) \leftarrow e(q,2), \text{not } e(p,2); & \leftarrow \text{not } covered(2) \end{array} \right.$$

$$E^- = \left\{ \begin{array}{ll} ex(negative); & violating \leftarrow \text{not } e(p,negative), \text{not } e(q,negative) \\ & violating \leftarrow e(p,negative), e(q,negative) \end{array} \right.$$

We must "assign" all of $ex(1)$, $ex(2)$ and $ex(negative)$ to $\{q\}$.

Constructing the task program T_{meta}

$$B = p \leftarrow \text{not } q$$

$$S_M = \left\{ \begin{array}{l} q \leftarrow \\ q \leftarrow \text{not } p \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right. \qquad E^- = \left\{ \begin{array}{l} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{array} \right.$$

When $n = 1$, we must choose the first rule as our hypothesis H.
$B \cup H$ has one Answer Set: $\{q\}$.

There are no Answer Sets of T^n_{meta} as H is not a positive solution.

Constructing the task program T_{meta}

$$B = p \leftarrow \text{not } q$$

$$S_M = \left\{ \begin{array}{l} q \leftarrow \\ q \leftarrow \text{not } p \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right. \qquad E^- = \left\{ \begin{array}{l} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{array} \right.$$

When $n = 2$, we must choose the second rule as our hypothesis H.
$B \cup H$ now has two Answer Sets: $\{p\}$ and $\{q\}$.

Constructing the task program T_{meta}

The Answer Sets of $B \cup H$ are $\{p\}$ and $\{q\}$.

$$E^+ = \left\{ \begin{array}{lll} ex(1); & covered(1) \leftarrow e(p,1), \text{not } e(q,1); & \leftarrow \text{not } covered(1) \\ ex(2); & covered(2) \leftarrow e(q,2), \text{not } e(p,2); & \leftarrow \text{not } covered(2) \end{array} \right.$$

$$E^- = \left\{ \begin{array}{ll} ex(negative); & violating \leftarrow \text{not } e(p,negative), \text{not } e(q,negative) \\ & violating \leftarrow e(p,negative), e(q,negative) \end{array} \right.$$

We must "assign" $ex(1)$ to $\{p\}$ and $ex(2)$ to $\{q\}$.

There are two different Answer Sets of T^n_{meta} where we "assign" $ex(negative)$ to each of the Answer Sets $\{p\}$ and $\{q\}$.

Constructing the task program T_{meta}

$$B = p \leftarrow \text{not } q$$

$$S_M = \left\{ \begin{array}{l} q \leftarrow \\ q \leftarrow \text{not } p \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right. \qquad E^- = \left\{ \begin{array}{l} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{array} \right.$$

When $n = 2$, we must choose the second rule as our hypothesis H.
$B \cup H$ now has two Answer Sets: $\{p\}$ and $\{q\}$.

There are two Answer Sets of T^n_{meta}:

$\{e(p,1), e(q,2), active(2), covered(1), covered(2), e(p, negative) \ldots\}$
$\{e(p,1), e(q,2), active(2), covered(1), covered(2), e(q, negative) \ldots\}$

which both correspond to the hypothesis: $q \leftarrow not\ p$.

Constructing the task program T_{meta}

$$B = \emptyset$$

$$S_M = \left\{ \begin{array}{l} q \leftarrow \\ q \leftarrow \text{not } p \\ 1\{p, q\}2 \end{array} \right.$$

$$E^+ = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right. \qquad E^- = \left\{ \begin{array}{l} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{array} \right.$$

Consider the new (incorrect) hypothesis (of length 6): $1\{p, q\}2$

$B \cup H$ has three Answer Sets: $\{p\}$, $\{q\}$ and $\{p, q\}$. The Answer Set $\{p, q\}$ extends a negative example.

Constructing the task program T_{meta}

The Answer Sets of $B \cup H$ are $\{p\}$, $\{q\}$ and $\{p, q\}$.

$$E^+ = \left\{ \begin{array}{lll} ex(1); & covered(1) \leftarrow e(p,1), \text{not } e(q,1); & \leftarrow \text{not } covered(1) \\ ex(2); & covered(2) \leftarrow e(q,2), \text{not } e(p,2); & \leftarrow \text{not } covered(2) \end{array} \right.$$

$$E^- = \left\{ \begin{array}{ll} ex(negative); & violating \leftarrow \text{not } e(p, negative), \text{not } e(q, negative) \\ & violating \leftarrow e(p, negative), e(q, negative) \end{array} \right.$$

We must "assign" $ex(1)$ to $\{p\}$ and $ex(2)$ to $\{q\}$.

There are three different Answer Sets of T^n_{meta} where we "assign" $ex(negative)$ to each of the three Answer Sets of $B \cup H$. The third one, where we "assign" $ex(negative)$ to $\{p, q\}$, contains the atom $violating$.

Constructing the task program T_{meta}

$$B = \emptyset$$

$$S_M = \begin{cases} q \leftarrow \\ q \leftarrow \text{not } p \\ 1\{p, q\}2 \end{cases}$$

$$E^+ = \begin{cases} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{cases} \qquad E^- = \begin{cases} \langle \emptyset, \{p, q\} \rangle \\ \langle \{p, q\}, \emptyset \rangle \end{cases}$$

Consider the new (incorrect) hypothesis (of length 6): $1\{p, q\}2$

$B \cup H$ has three Answer Sets: $\{p\}$, $\{q\}$ and $\{p, q\}$. The Answer Set $\{p, q\}$ extends a negative example.
When $n = 6$, the Answer Sets of T^n_{meta} are:

$\{e(p,1), e(q,2), active(3), covered(1), covered(2), e(p, negative) \ldots\}$
$\{e(p,1), e(q,2), active(3), covered(1), covered(2), e(q, negative) \ldots\}$
$\{e(p,1), e(q,2), active(3), covered(1), covered(2), e(p, negative),$
$e(q, negative), violating \ldots\}$

Inductive Learning of Answer Set Programs

Algorithm 1 ILASP

procedure ILASP(T)
  $solutions = []$
  for $n = 0$; $solutions.empty$; $n{+}{+}$ **do**
    $vs = AS(T^n_{meta} \cup \{\leftarrow \text{ not } violating\})$
    $ps = AS(T^n_{meta} \cup \{constraint(meta^{-1}(V)) : V \in vs\})$
    $solutions = \{meta^{-1}(A) : A \in ps\}$
  end for
  return *solutions*
end procedure

T^n_{meta}: ASP task program (a meta representation of the task T)
vs: violating solutions
ps: positive solutions

Inductive Learning of Answer Set Programs

Let $ILP^*_{LAS}(T)$ be the optimal inductive solutions of the task T.

Theorem 2. *Let T be any ILP_{LAS} learning task such that there is at least one inductive solution.*

Then $ILASP(T) = ILP^*_{LAS}(T)$.

Imperial College London

Comparison with related works

$$ILP_{brave}\langle B, E\rangle$$

$$ILP_{ASPAL/XHAIL}\langle B, \langle E, \emptyset\rangle\rangle$$

$$ILP_{ASPAL/XHAIL}\langle B, \langle E^+, E^-\rangle\rangle$$

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

$$ILP_{stable\_models}\langle B, \{\langle E_1^+, E_1^-\rangle \ldots \langle E_n^+, E_n^-\rangle\}\rangle$$

$$ILP_{LAS}\langle B, \{\langle E_1^+, E_1^-\rangle \ldots \langle E_n^+, E_n^-\rangle\}, \emptyset\rangle$$

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

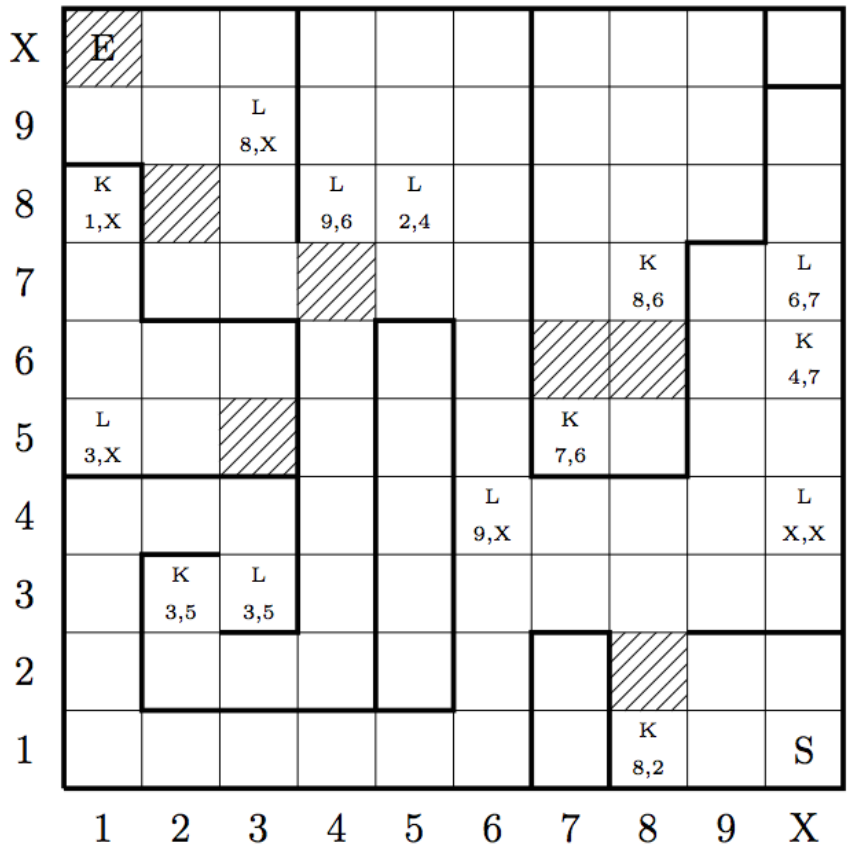
Comparison with related works

$$ILP_{cautious}\langle B, \{e_1, \ldots, e_n\}\rangle$$

$$ILP_{LAS}\langle B, \emptyset, \{\langle \emptyset, \{e_1\}\rangle \ldots \langle \emptyset, \{e_n\}\rangle\}\rangle$$

Imperial College London

Application to a planning problem



S — The agent *starts* here

E — The agent is aiming to *end* here

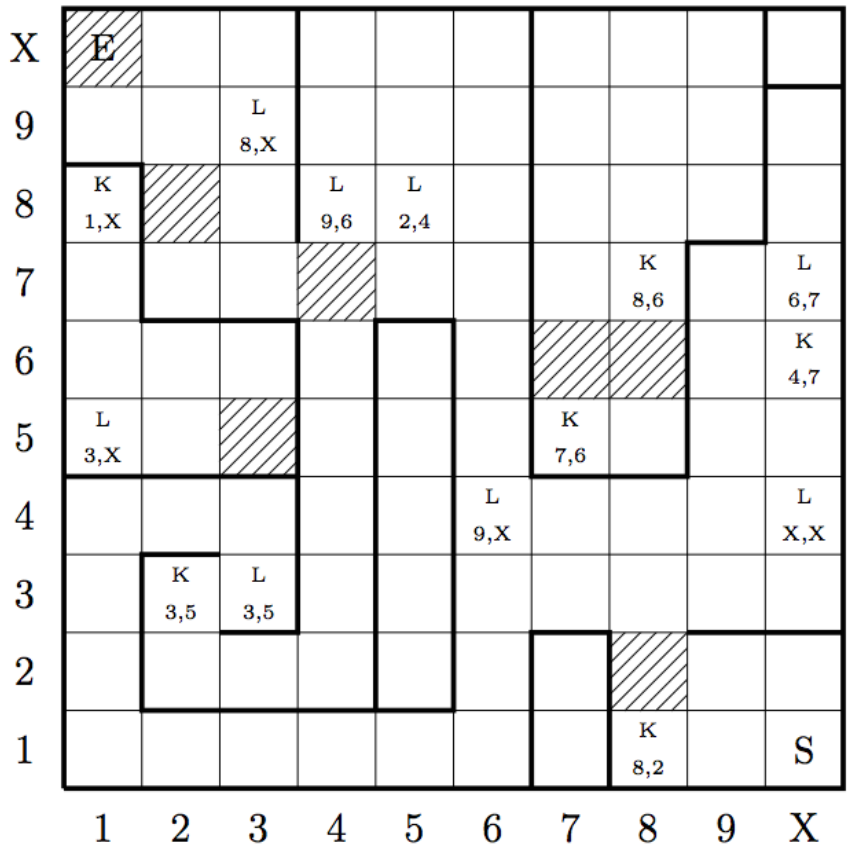
This cell starts off as being *locked* Cell

K 4,7 — This is the *key* for the locked cell $(4, 7)$

L 8,X — This is a *link* cell which allows the agent to move to $(8, X)$

Imperial College London

Application to a planning problem

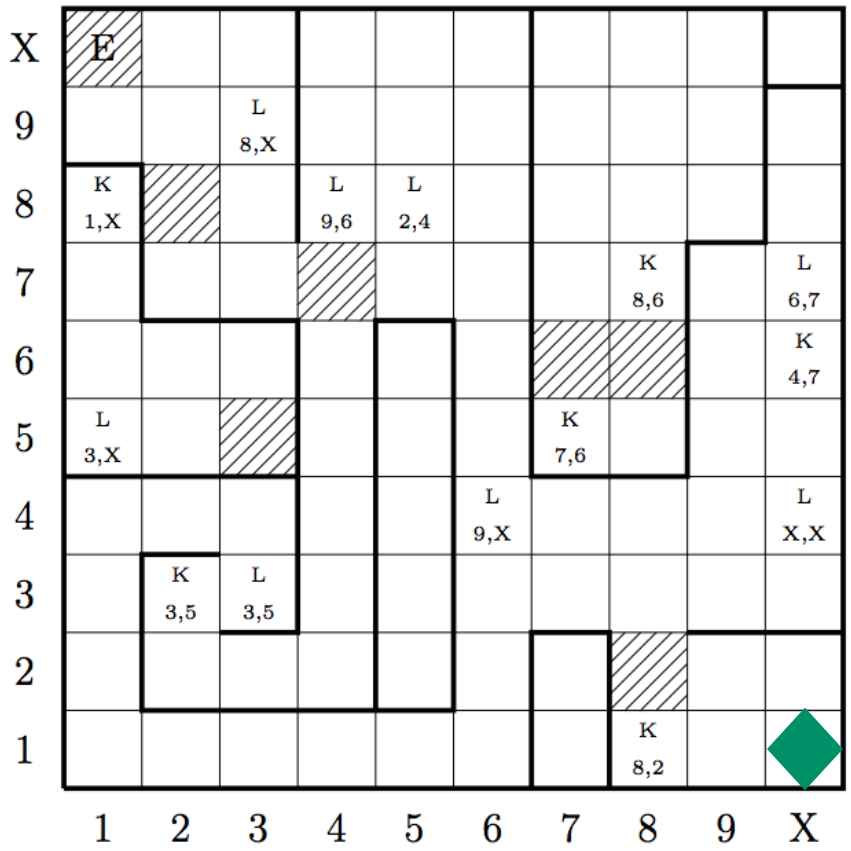



```
unlocked(C, T) :- not locked(C),
                  cell(C),
                  time(T).

unlocked(C, T) :- visited_cell(Key, T),
                  key(Key, C).
```

Imperial College London

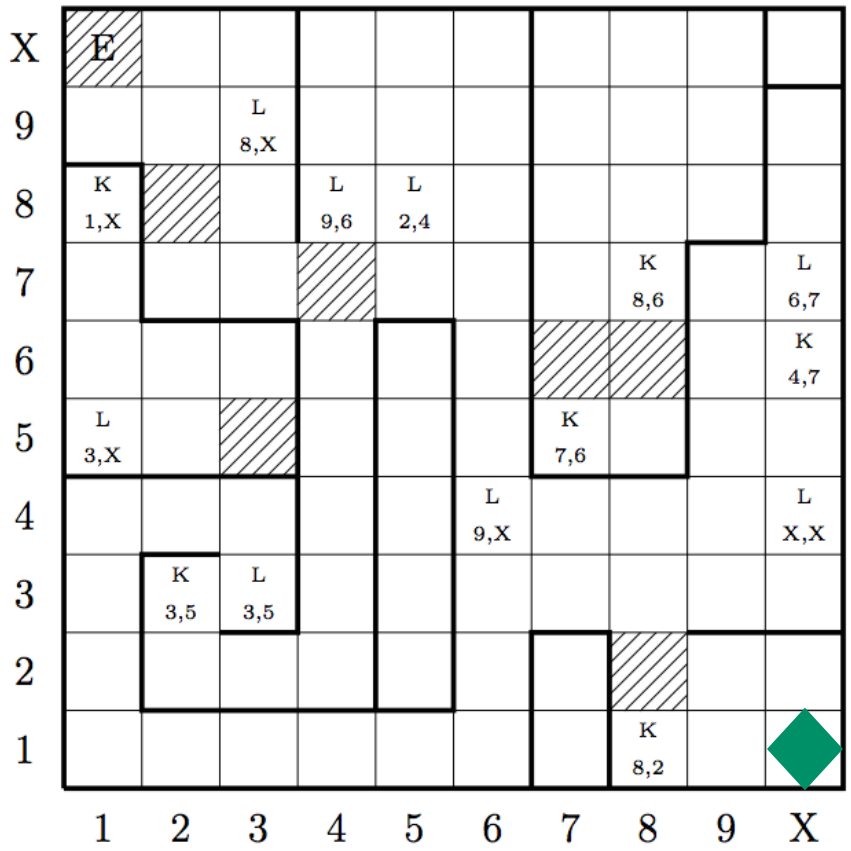
Application to a planning problem



valid_moves(1) = {(9,1), (10, 2)}

Imperial College London

Application to a planning problem

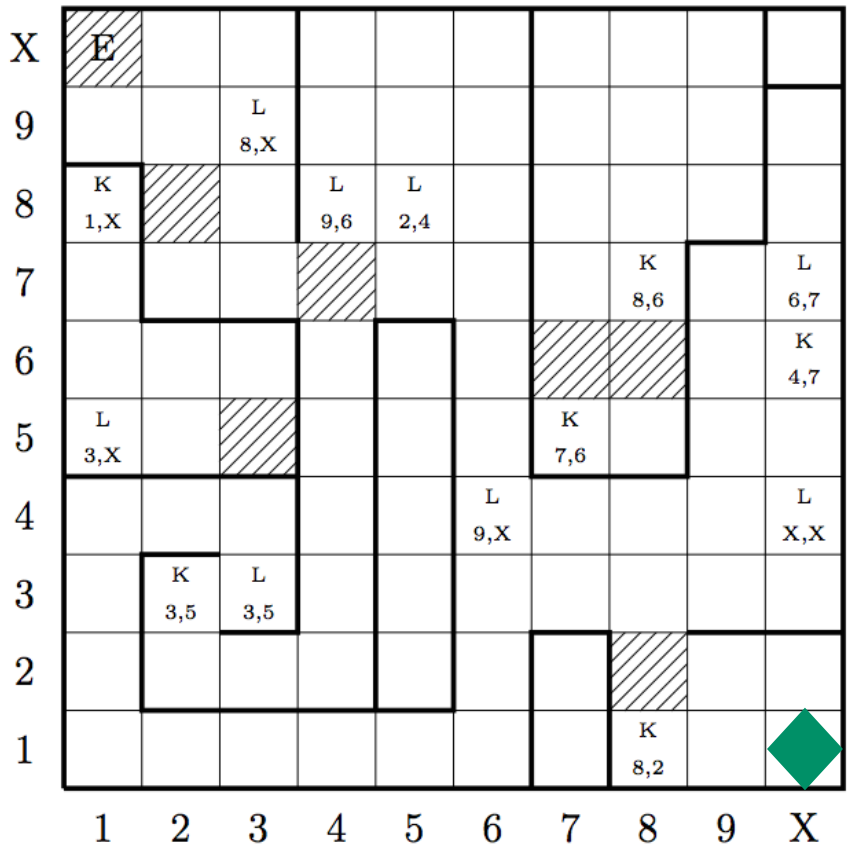


valid_moves(1) = {(9,1), (10, 2)}

$E^+(1)$ = <{valid_move(cell(9,1), 1),
valid_move(cell(10, 2),1)},{}>

Imperial College London

Application to a planning problem

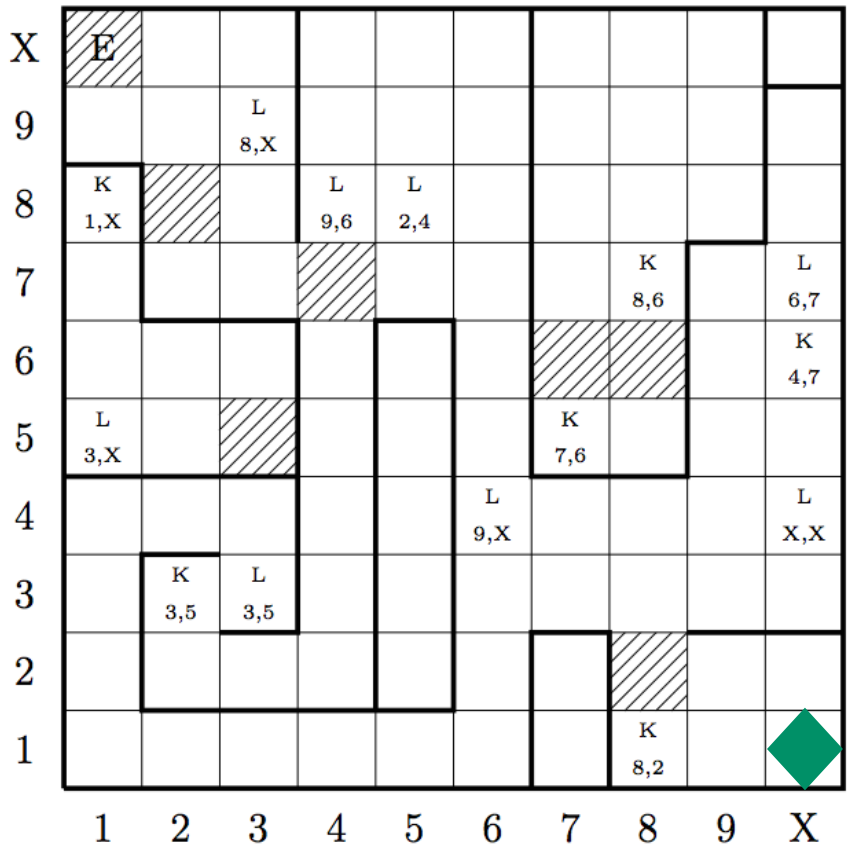


valid_moves(1) = {(9,1), (10, 2)}

$E^{+}(1)$ = <{valid_move(cell(9,1), 1),
valid_move(cell(10, 2),1)},{}>

valid_move(C, T) :- unlocked(C, T).

Application to a planning problem



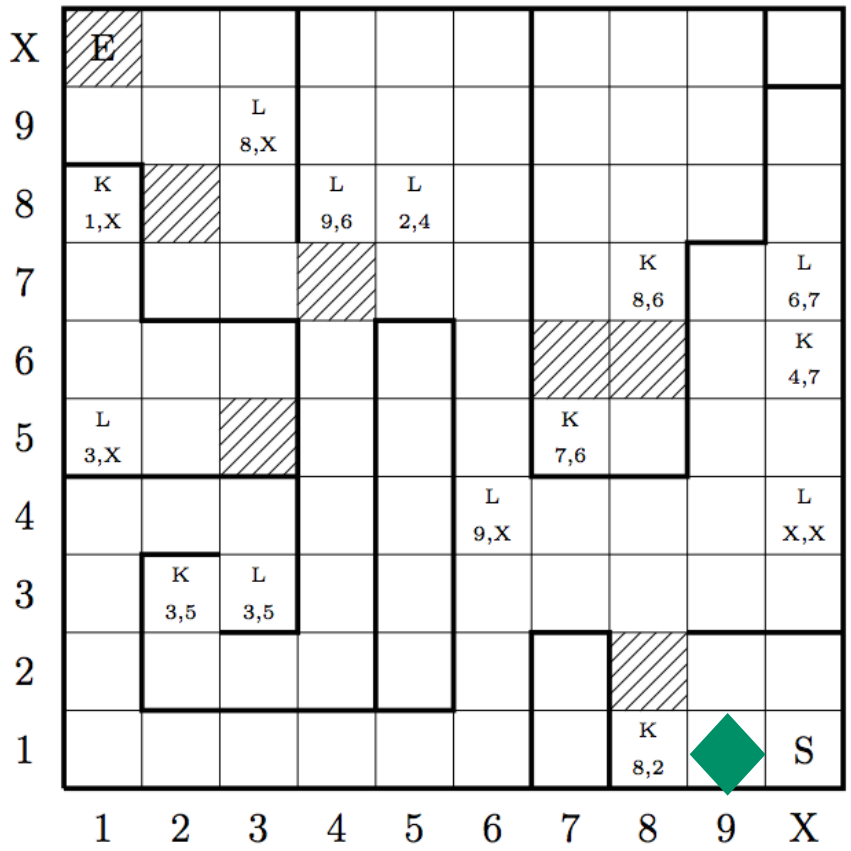
valid_moves(1) = {(9,1), (10, 2)}

$E^{+}(1)$ = <{valid_move(cell(9,1), 1),
valid_move(cell(10, 2),1)},{}>

E^{-} = {
<{valid_move(cell(9,2),1)},{}>,
<{valid_move(cell(8,1),1)},{}>,
…
}

valid_move(C1, T) :-
agent_at(C2, T),
adjacent(C1, C2).

Application to a planning problem

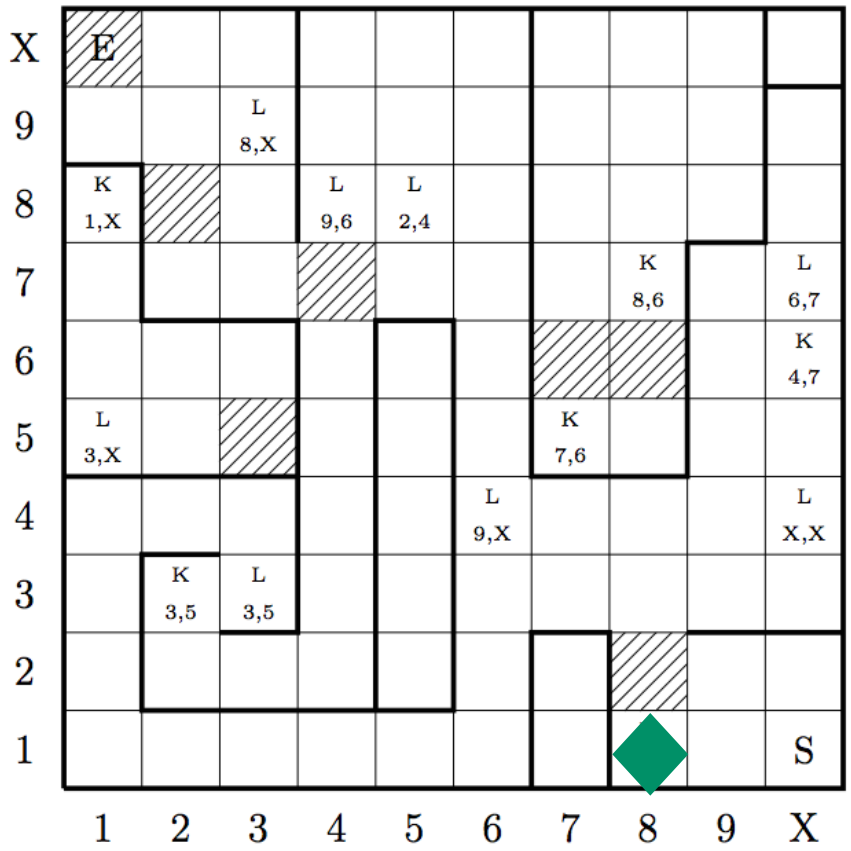


valid_moves(1) = {(8, 1), (10, 1), (9, 2)}

```
valid_move(C1, T) :-
        agent_at(C2, T),
        adjacent(C1, C2).
```

Imperial College London

Application to a planning problem

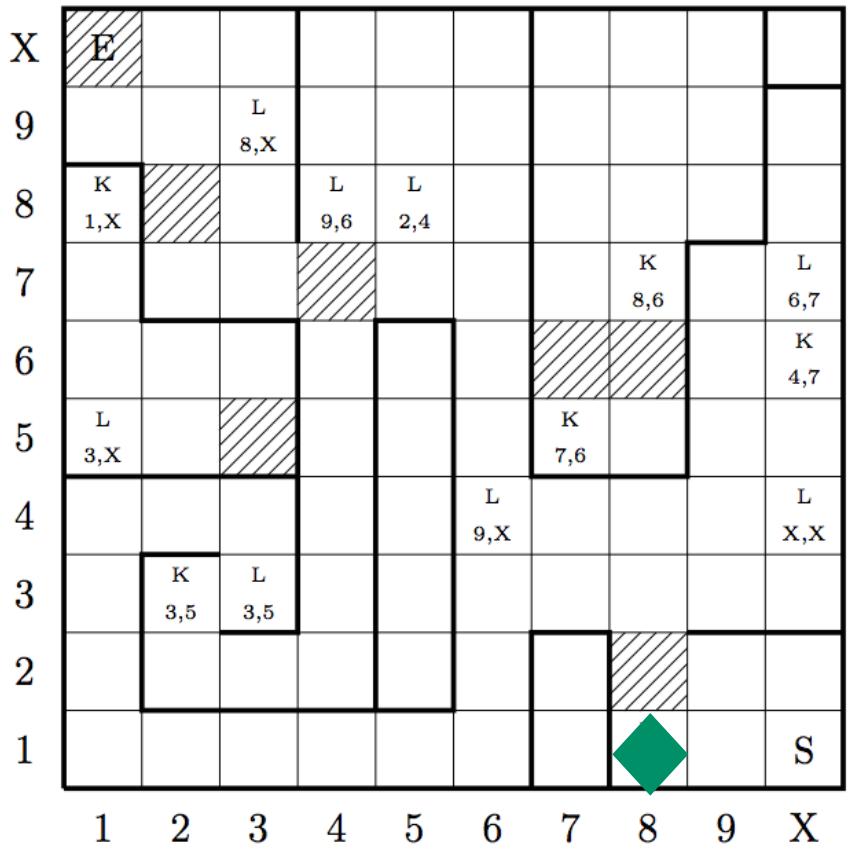


valid_moves(1) = {(9, 1), (8, 2)}

```
valid_move(C1, T) :-
        agent_at(C2, T),
        adjacent(C1, C2).
```

Imperial College London

Application to a planning problem

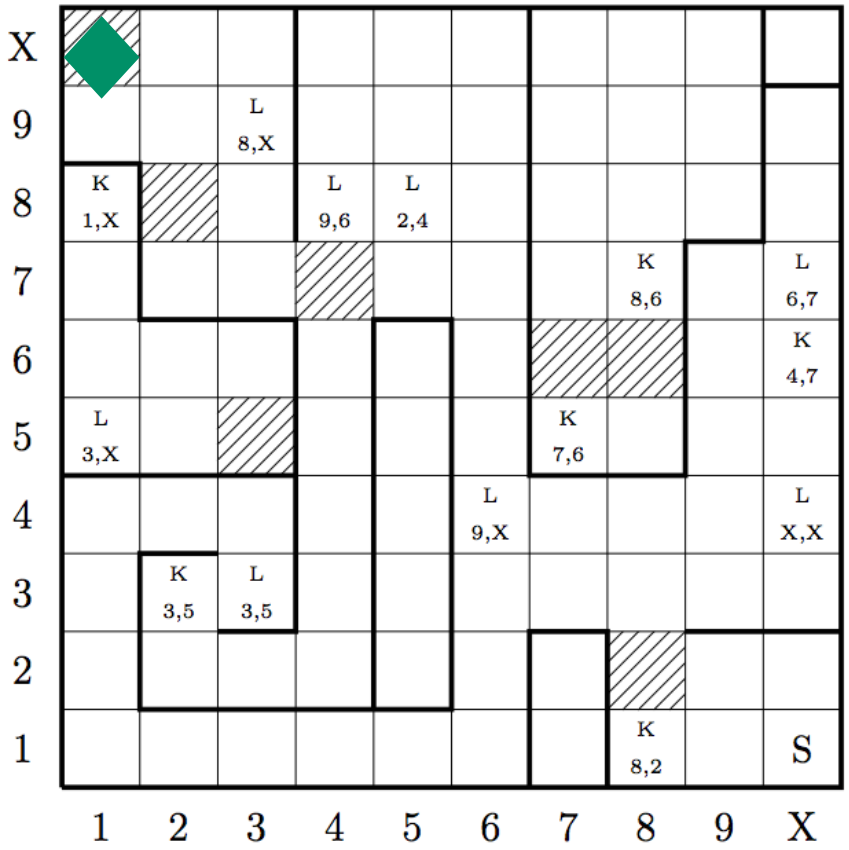


valid_moves(1) = {(9, 1), (8, 2)}

```
valid_move(C1, T) :-
        agent_at(C2, T),
        not wall(C1, C2),
        adjacent(C1, C2).
```

Imperial College London

Application to a planning problem




```
valid_move(C1, T) :-
        unlocked(C1, T),
        agent_at(C2, T),
        not wall(C1, C2),
        adjacent(C1, C2).

valid_move(C1, T) :-
        unlocked(C1, T),
        agent_at(C2, T),
        link(C1, C2).
```

Predicate Invention

We took away the rule for unlocked from the background knowledge:

```
unlocked(C, T) :- not locked(C), cell(C), time(T).
unlocked(C, T) :- visited_cell(Key, T), key(Key, C).
```

We expected the agent to relearn this definition, along with the previous definition for valid move. In fact the agent learned the shorter hypothesis:

```
extra(C,T) :- agent_at(C1, V1), link(C1,C).
extra(C,T) :- adjacent(C,C1), agent_at(C1, T), not wall(C,C1).

valid_move(C, T) :- extra(C,T), not locked(C).
valid_move(C, T) :- extra(C,T), key(C1,C), visited_cell(C1,T).
```

Non-deterministic rules

Up until now, we have been learning programs with one Answer Set. We now change the moves given by the oracle:

- A cell with a link to (X,Y) will now either allow the agent to go to (X, Y), or it will allow the agent to go to (Y, X).

This forces the agent to learn the rules:

```
1 {valid_move(C, T); valid_move(FC, T)} 1 :-  unlocked(C, T),
                                              link(C2, C, FC),
                                              agent_at(C2, T).

valid_move(C, T) :- unlocked(C, T),
                    joined(C, C2),
                    agent_at(C2, T).
```

Current work: modification of ILASP

- For some classes of problem there could be many violating solutions before we find an inductive solution.
- The sudoku example is one such problem, with 413044 before the first inductive solution it takes over 14 minutes to solve with ILASP.
- In fact, many of these are violating for the same reason (they share Answer Sets which extend negative examples).
- With our new system based on ruling out classes of hypothesis, we need only 7 classes and the problem is solved in less than 1 second.

Other current work

- Expand the subset of ASP that we can learn
  - conditions, weighted aggregates etc.
  - weak constraints/optimisation statements

- Real applications
  - Ideally not achievable by other ILP tasks
  - Will motivate the work from a practical point of view
  - Measure the accuracy of the learning task

Questions?

How does this compare to the learning of Prolog programs?

- The main advantage of learning ASP rather than Prolog is that it is purely declarative. The sudoku problem can be written in Prolog as follows:

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

board(B) :- partial(B, 0).

partial([], 16).
partial([V|Prev], C-1) :- value(V), partial(Prev, C), not violation(V, Prev, C-1).

violation(V, [V, Prev] Index, Cell) :- same_row(Index, Cell).
violation(V, [V, Prev] Index, Cell) :- same_col(Index, Cell).
violation(V, [V, Prev] Index, Cell) :- same_block(Index, Cell).
violation(V, [V2, Prev], Index, Cell) :- violation(V, Prev, Index - 1, Cell).
```

- In fact, this also relies on negation as failure, which not many Prolog ILP systems allow for!

Imperial College London

Learning from Interpretation Transitions

Take the ILP_{LFIT} task:

$$B = \emptyset, \qquad E = \left\{ \begin{array}{l} \langle \{p\}, \{q\} \rangle \\ \langle \{q\}, \{p\} \rangle \end{array} \right. , \qquad S_M = \left\{ \begin{array}{l} p \leftarrow q \\ q \leftarrow p \end{array} \right.$$

We could map this to the ILP_{LAS} task:

$$B = \left\{ \begin{array}{l} 1\{example1, example2\}1 \\ p \leftarrow example1 \\ q \leftarrow example2 \end{array} \right. , \qquad E^+ = \left\{ \begin{array}{l} \langle \{example1, next(q)\}, \{next(p)\} \rangle \\ \langle \{example2, next(p)\}, \{next(q)\} \rangle \end{array} \right. ,$$

$$E^- = \emptyset, \qquad S_M = \left\{ \begin{array}{l} next(p) \leftarrow q \\ next(q) \leftarrow p \end{array} \right.$$

This would have the hypothesis: $\{next(p) \leftarrow q;\ \ next(q) \leftarrow p\}$.
This could then be mapped back to the hypothesis: $\{p \leftarrow q;\ \ q \leftarrow p\}$

How does this compare to the learning of Prolog programs?

- Most ILP systems which learn Prolog are restricted to learning monotonic logic programs.
- If we restrict ourselves to only the shared syntax of ASP and Prolog (no lists, aggregates, constraints or optimisation statements), then the “usual” monotonic ILP task maps trivially to a Learning from Answer Sets task with one positive example.

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

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