Imperial College
London

Parallelizing Inductive Logic Programming in ASP

Nikolay Paleshnikov

June 24, 2016

Imperial College London

Example of induction in ASPAL

Solve the inductive learning task with

- $B = \{r(a).\ t(a).\ t(b).\}$
- $M_h = \{modeh(p(+t)).\}$ and
  $M_b = \{modeb(q(+t)).\ modeb(r(+t)).\}$
- $E^+ = \{p(a)\}$ and $E^- = \{p(b)\}$

Imperial College London

Example of induction in ASPAL

Solve the inductive learning task with

- $B = \{r(a).\ t(a).\ t(b).\}$
- $M_h = \{modeh(p(+t)).\}$ and $M_b = \{modeb(q(+t)).\ modeb(r(+t)).\}$
- $E^+ = \{p(a)\}$ and $E^- = \{p(b)\}$

The skeleton rules are the following:

$S_M = \{$
$p(A) \leftarrow t(A).$
$p(A) \leftarrow t(A), q(A).$
$p(A) \leftarrow t(A), r(A).$
$p(A) \leftarrow t(A), q(A), r(A).\}$

Example of induction in ASPAL

Solve the inductive learning task with

- $B = \{r(a).\ t(a).\ t(b).\}$
- $M_h = \{modeh(p(+t)).\}$ and
  $M_b = \{modeb(q(+t)).\ modeb(r(+t)).\}$
- $E^+ = \{p(a)\}$ and $E^- = \{p(b)\}$

The skeleton rules are the following:
$S_M = \{$
$p(A) \leftarrow t(A).$
$p(A) \leftarrow t(A), q(A).$
$p(A) \leftarrow t(A), r(A).$
$p(A) \leftarrow t(A), q(A), r(A).\}$
The only optimal solution H is

```
p(A) :- t(A), r(A).
```

such that $\{r(a), t(a), t(b), p(a)\} \in AS(B \cup H)$

Induction in ASP

An inductive learning task $< B, E, M >$ is defined through:

- background knowledge B
- positive examples E^+ and negative examples E^-
- head declarations M_h and body declarations M_b

Induction in ASP

An inductive learning task $< B, E, M >$ is defined through:

- background knowledge B
- positive examples E^+ and negative examples E^-
- head declarations M_h and body declarations M_b

An inductive solution H of $< B, E, M >$, called a hypothesis, must satisfy the following conditions:

- $H \subseteq S_M$
- $\exists a \in AS(B \cup H)\ s.t.\ \forall e \in E^+\ (e \in a) \wedge \forall e \in E^-\ (e \notin a)$

Imperial College London

Induction in ASP

An inductive learning task $< B, E, M >$ is defined through:

- background knowledge B
- positive examples E^+ and negative examples E^-
- head declarations M_h and body declarations M_b

An inductive solution H of $< B, E, M >$, called a hypothesis, must satisfy the following conditions:

- $H \subseteq S_M$
- $\exists a \in AS(B \cup H)\ s.t.\ \forall e \in E^+\ (e \in a) \wedge \forall e \in E^-\ (e \notin a)$

Main issue

Grounding of the meta task restricts scalability since it grows exponentially in respect to the:

- size of hypothesis space
- Herbrand domain of the program

Imperial College London

Example of a split

```
% mode declarations
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
modeh(lonely(+childless)).

modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
modeb(offspring(+parent,-child)).
```

```
% background knowledge
humanist(X) :- scientist(X).
child(annika).
child(tommy).
child(jack).
parent(john).
parent(clara).
childless(persephone).
offspring(clara, annika).
offspring(john, jack).
adventurous(tommy).
curious(annika).
```

```
% examples
example(humanist(tommy),-1).
example(humanist(annika),1).
example(explorer(tommy),1).
example(explorer(annika),-1).
example(proud(clara),1).
example(proud(john),-1).
example(lonely(persephone),1).
```

Example of a split

```
% mode declarations
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
modeh(lonely(+childless)).

modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
modeb(offspring(+parent,-child)).
```

We can split the head declarations into

```
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
```

and

```
modeh(lonely(+childless)).
```

```
% background knowledge
humanist(X) :- scientist(X).
child(annika).
child(tommy).
child(jack).
parent(john).
parent(clara).
childless(persephone).
offspring(clara, annika).
offspring(john, jack).
adventurous(tommy).
curious(annika).
```

```
% examples
example(humanist(tommy),-1).
example(humanist(annika),1).
example(explorer(tommy),1).
example(explorer(annika),-1).
example(proud(clara),1).
example(proud(john),-1).
example(lonely(persephone),1).
```

as well as the examples into

```
example(humanist(tommy),-1).
example(humanist(annika),1).
example(explorer(tommy),1).
example(explorer(annika),-1).
example(proud(clara),1).
example(proud(john),-1).
```

and

```
example(lonely(persephone),1).
```

Imperial College London

Example of a split

```
% mode declarations
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
modeh(lonely(+childless)).

modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
modeb(offspring(+parent,-child)).
```

```
% background knowledge
humanist(X) :- scientist(X).
child(annika).
child(tommy).
child(jack).
parent(john).
parent(clara).
childless(persephone).
offspring(clara, annika).
offspring(john, jack).
adventurous(tommy).
curious(annika).
```

```
% examples
example(humanist(tommy),-1).
example(humanist(annika),1).
example(explorer(tommy),1).
example(explorer(annika),-1).
example(proud(clara),1).
example(proud(john),-1).
example(lonely(persephone),1).
```

We can split further to obtain

```
modeh(scientist(+child)).
modeh(explorer(+child)).
```

as well as

```
example(humanist(tommy),-1).
example(humanist(annika),1).
example(explorer(tommy),1).
example(explorer(annika),-1).
```

```
modeh(proud(+parent)).
```

```
example(proud(clara),1).
example(proud(john),-1).
```

and

```
modeh(lonely(+childless)).
```

and

```
example(lonely(persephone),1).
```

Imperial College London

Example of a split

head declarations

```
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
```

solutions

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), scientist(B).
```

and

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), curious(B).
```

head declaration

```
modeh(lonely(+childless)).
```

solutions

```
lonely(A):- childless(A).
```

Imperial College London

Example of a split

head declarations

```
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
```

solutions

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), scientist(B).
```

and

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), curious(B).
```

head declaration

```
modeh(lonely(+childless)).
```

solutions

```
lonely(A):- childless(A).
```

solutions of the original task

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), scientist(B).
lonely(A):- childless(A).
```

and

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), curious(B).
lonely(A):- childless(A).
```

Imperial College London

Example of a split

head declarations

```
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
```

solutions

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), scientist(B).
```

and

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
proud(A):- parent(A), offspring(A,B),
   child(B), curious(B).
```

head declarations

```
modeh(scientist(+child)).
modeh(explorer(+child)).
```

solutions

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
```

rules from above plus head declaration

```
modeh(proud(+parent)).
```

solutions

```
proud(A):- parent(A), offspring(A,B),
   child(B), scientist(B).
```

and

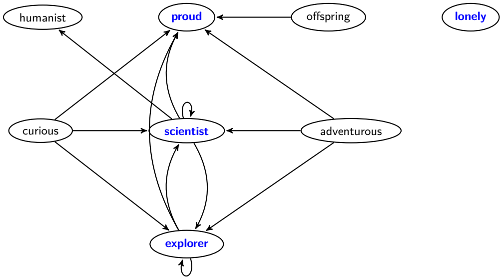
```
proud(A):- parent(A), offspring(A,B),
   child(B), curious(B).
```

Example of a split

```
% mode declarations
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
modeh(lonely(+childless)).

modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
modeb(offspring(+parent,-child)).
```

```
% background knowledge
humanist(X) :- scientist(X).
child(annika).
child(tommy).
child(jack).
parent(john).
parent(clara).
childless(persephone).
offspring(clara, annika).
offspring(john, jack).
adventurous(tommy).
curious(annika).
```

```
% examples
example(humanist(tommy),-1).
example(humanist(annika),1).
example(explorer(tommy),1).
example(explorer(annika),-1).
example(proud(clara),1).
example(proud(john),-1).
example(lonely(persephone),1).
```

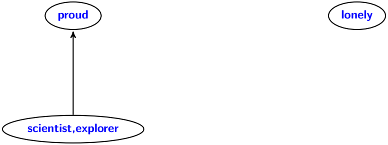


Example of a split

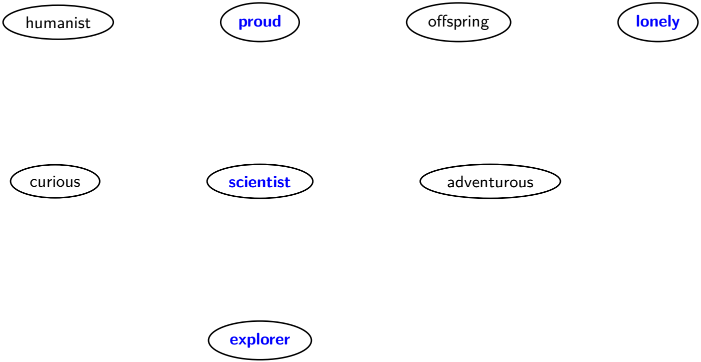
```
% mode declarations
modeh(scientist(+child)).
modeh(explorer(+child)).
modeh(proud(+parent)).
modeh(lonely(+childless)).

modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
modeb(offspring(+parent,−child)).
```

```
% background knowledge
humanist(X) :− scientist(X).
child(annika).
child(tommy).
child(jack).
parent(john).
parent(clara).
childless(persephone).
offspring(clara, annika).
offspring(john, jack).
adventurous(tommy).
curious(annika).
```

```
% examples
example(humanist(tommy),−1).
example(humanist(annika),1).
example(explorer(tommy),1).
example(explorer(annika),−1).
example(proud(clara),1).
example(proud(john),−1).
example(lonely(persephone),1).
```



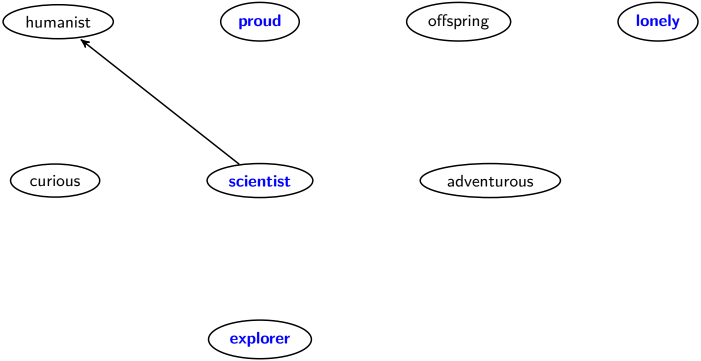
Example of the dependency graph construction




```
humanist(X) :- scientist(X).
```

Imperial College London

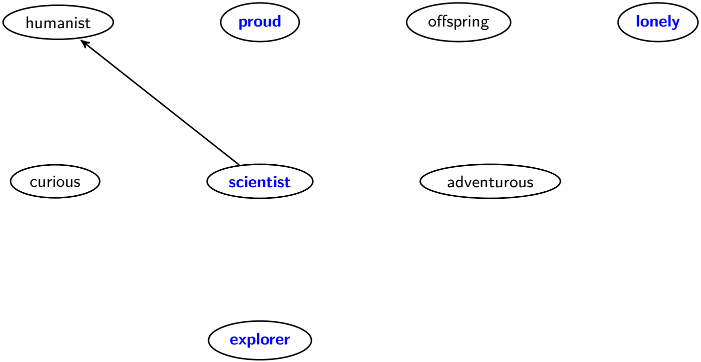
Example of the dependency graph construction




```
humanist(X) :- scientist(X).
```

Imperial College London

Example of the dependency graph construction

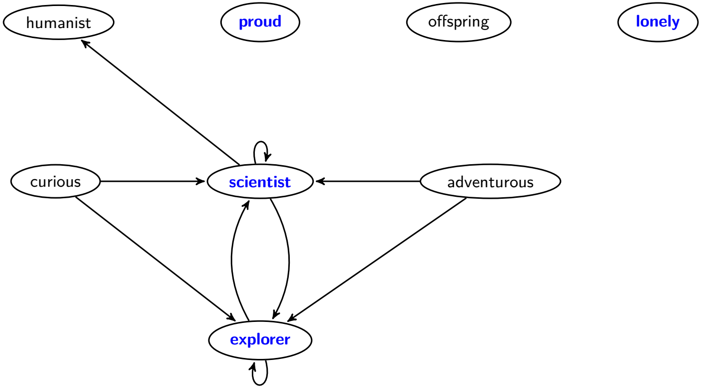



```
modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
```

```
modeh(scientist(+child)).
modeh(explorer(+child)).
```

Imperial College London

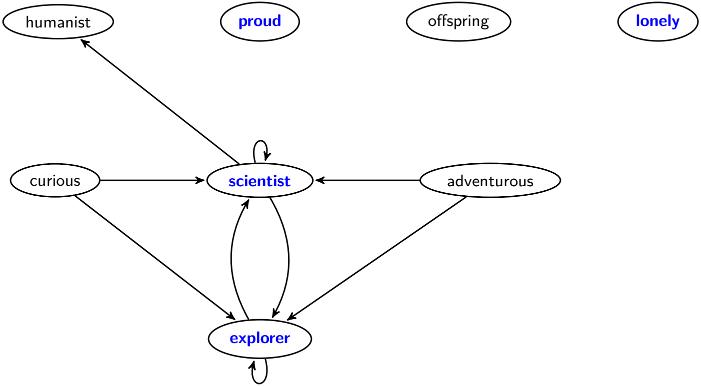
Example of the dependency graph construction




```
modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
```

```
modeh(scientist(+child)).
modeh(explorer(+child)).
```

Example of the dependency graph construction

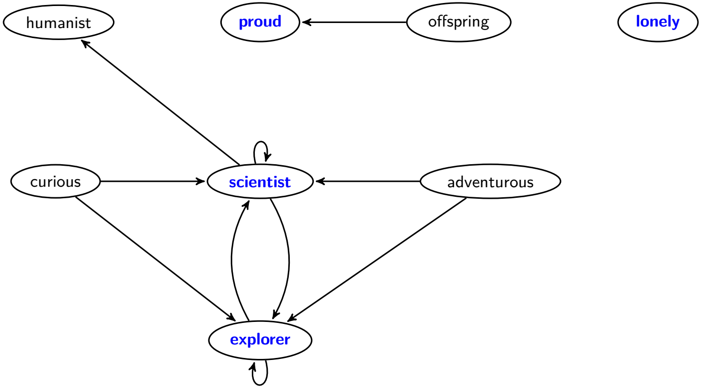



```
modeb(offspring(+parent,-child)).
```

```
modeh(proud(+parent)).
```

Imperial College London

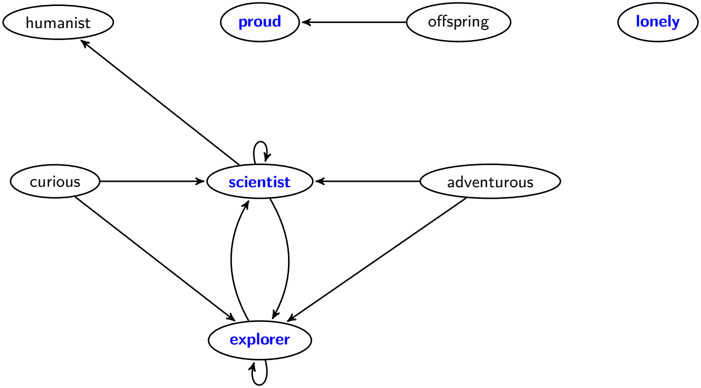
Example of the dependency graph construction




```
modeb(offspring(+parent,-child)).          modeh(proud(+parent)).
```

Imperial College
London

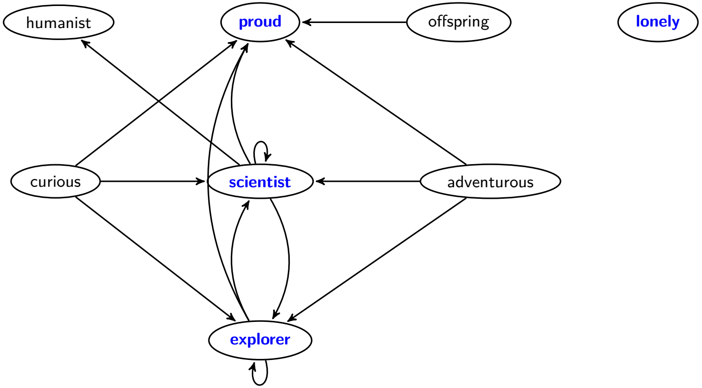
Example of the dependency graph construction




```
modeb(offspring(+parent,-child)).
modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
```

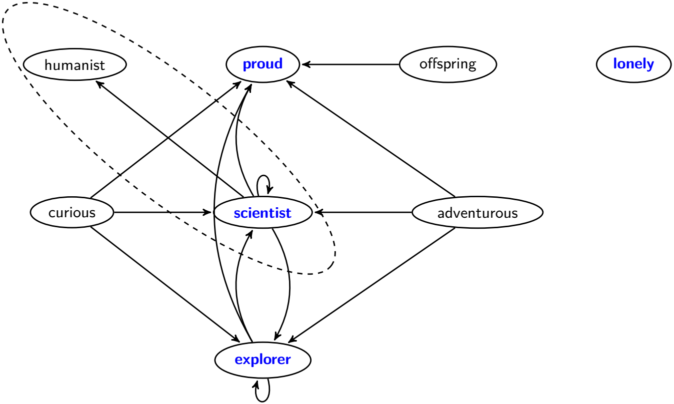
```
modeh(proud(+parent)).
```

Example of the dependency graph construction




```
modeb(offspring(+parent,-child)).       modeh(proud(+parent)).
modeb(scientist(+child)).
modeb(explorer(+child)).
modeb(adventurous(+child)).
modeb(curious(+child)).
```

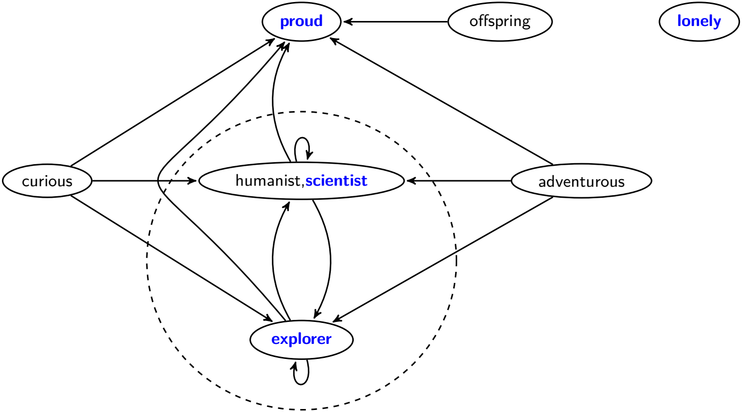
Example of reduction




```
humanist(X) :- scientist(X).
example(humanist(tommy),-1).
example(humanist(annika),1).
```

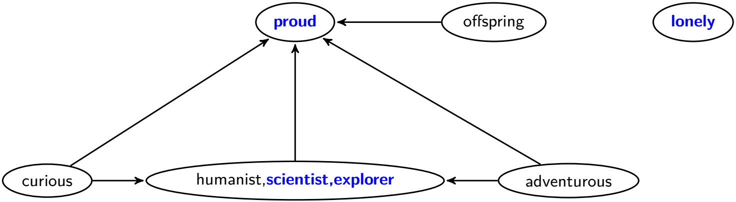
Imperial College London

Example of reduction

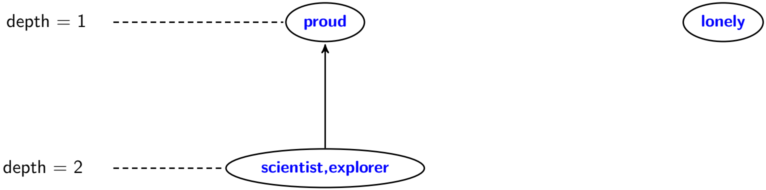


$\{(\{\text{humanist,scientist}\},\{\text{explorer}\}),\ (\{\text{explorer}\},\{\text{humanist,scientist}\})\} \subset E$

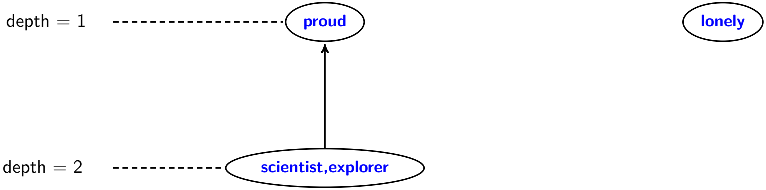
Example of reduction



Dependencies between head predicates



Dependencies between head predicates




```
proud(A):- parent(A), offspring(A,B),
   child(B), scientist(B).
```

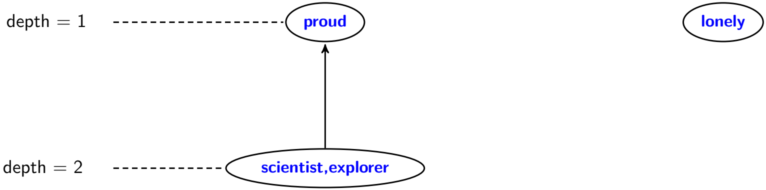
```
lonely(A):- childless(A).
```

and

```
proud(A):- parent(A), offspring(A,B),
   child(B), curious(B).
```

```
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
```

Dependencies between head predicates




```
proud(A):-parent(A),offspring(A,B),
   child(B),scientist(B).
scientist(A):-child(A),curious(A).
explorer(A):-child(A),adventurous(A).
```

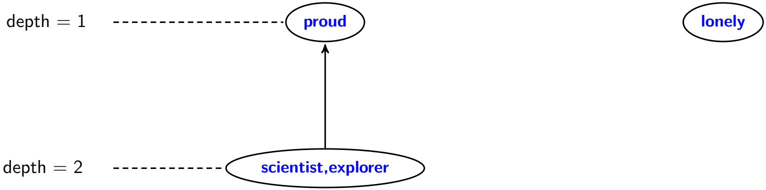
```
lonely(A):-childless(A).
```

and

```
proud(A):-parent(A),offspring(A,B),
   child(B),curious(B).
scientist(A):-child(A),curious(A).
explorer(A):-child(A),adventurous(A).
```

Imperial College
London

Dependencies between head predicates




```
lonely(A):- childless(A).
proud(A):- parent(A), offspring(A,B),
   child(B), scientist(B
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
```

and

```
lonely(A):- childless(A).
proud(A):- parent(A), offspring(A,B),
   child(B), curious(B).
scientist(A):- child(A), curious(A).
explorer(A):- child(A), adventurous(A).
```

Correctness of the horizontal split

Definition

We call two sets of head declarations M_1 and M_2 disconnected iff $pred(M_1)$ and $pred(M_2)$ are disjoint and there is no predicate in any of $pred(M_1)$ and $pred(M_2)$ dependent on a predicate in the another.

Correctness of the horizontal split

Definition

We call two sets of head declarations M_1 and M_2 disconnected iff $pred(M_1)$ and $pred(M_2)$ are disjoint and there is no predicate in any of $pred(M_1)$ and $pred(M_2)$ dependent on a predicate in the another.

Theorem

For an inductive learning task $< B, E, M >$ and head declaration sets $\{h_1, ..., h_n\}$ to be split along, which are pairwise disconnected and satisfy $\bigcup\{mode(h) \mid h \in h_1 \cup ... \cup h_n\} = M_h$,

$ASPAL^*(< B, E, M >) =$
$\bigtimes\{ASPAL^*(< B, E_h, < mode(h), M_b >>) \mid h \in \{h_1, ..., h_n\}\}$

Correctness of the vertical split

Definition

For an inductive learning task $< B, E, M >$ and
$CC \in connComp(redDepGr(B, M))$, define $CCRules(CC)$ as follows:
Base case: CC has a single node.
$CCRules(CC) = ASPAL^*(< B, E_{CC}, < M_{CC}, M_b >>)$
Inductive clause:
$CCRules(CC) = \{defComb \cup leafComb \mid$
$defComb \in \bigtimes\{CCRules(CSUBC) \mid$
$CSUBC \in connComp(remNs(CC, leaves(CC)))\} \wedge$
$leafComb \in \bigtimes\{ASPAL^*(< B \cup defComb, E_l,$
$< mode(l), M_b >>) \mid l \in leaves(CC)\}\}$

Imperial College London

Correctness of the vertical split

Definition

For an inductive learning task $< B, E, M >$ and
$CC \in connComp(redDepGr(B, M))$, define $CCRules(CC)$ as follows:
Base case: CC has a single node.
$CCRules(CC) = ASPAL^*(< B, E_{CC}, < M_{CC}, M_b >>)$
Inductive clause:
$CCRules(CC) = \{defComb \cup leafComb \mid$
$defComb \in \bigtimes\{CCRules(CSUBC) \mid$
$CSUBC \in connComp(remNs(CC, leaves(CC)))\} \wedge$
$leafComb \in \bigtimes\{ASPAL^*(< B \cup defComb, E_l,$
$< mode(l), M_b >>) \mid l \in leaves(CC)\}\}$

Theorem

For an ILP task $< B, E, M >$ and a connected component
$CC \in connComp(redDepGr(B, M))$,

$CCRules(CC) \subseteq ASPAL(< B, E_{CC}, < M_{CC}, M_b >>)$.

Algorithm summary

procedure PARALLELIZER(ASPAL input file)
    $< B, E, M > \leftarrow$ parseFile(ASPAL input file)
    $graph \leftarrow$ `RedDepGr`(B, M)
    $component\_rules\_set \leftarrow \emptyset$
    for $CC \in connComp(graph)$ **do**
        $component\_rules\_set.add($`CCRules(CC)`$)$
    end for
    return $\bigtimes component\_rules\_set$
end procedure

Comparison with PASPAL

PASPAL

- defines a split into subtasks with a single head declaration each if there are no interdependencies
- passes the task unamended to ASPAL otherwise

Comparison with PASPAL

PASPAL

- defines a split into subtasks with a single head declaration each if there are no interdependencies
- passes the task unamended to ASPAL otherwise

$\Rightarrow$ coincides with our horizontal split if there are no interdependencies

Evaluation

Table : Runtimes of Parallelizer, PASPAL and ASPAL on the original mobile task

	general solutions	optimal solutions
Parallelizer	22m34s	25m11s
PASPAL	51m36s	–
ASPAL	ran out of memory and crashed after 26m	ran out of memory and crashed after 30m50s

Evaluation

Table : Runtimes of Parallelizer, PASPAL and ASPAL on the original mobile task

	general solutions	optimal solutions
Parallelizer	22m34s	25m11s
PASPAL	51m36s	–
ASPAL	ran out of memory and crashed after 26m	ran out of memory and crashed after 30m50s

Table : Runtimes of Parallelizer and ASPAL on the enriched mobile task

	general solutions	optimal solutions
Parallelizer	10m17s	11m1s
ASPAL	ran out of memory and crashed after 17m34s	ran out of memory and crashed after 19m26s

Imperial College London

Future work

Main setback of our approach

For a given ILP task, the split within a connected component of its reduced dependency graph is incomplete under optimal solution settings. Furthermore, the computation of general solutions is often infeasible in practice.

Future work

Main setback of our approach

For a given ILP task, the split within a connected component of its reduced dependency graph is incomplete under optimal solution settings. Furthermore, the computation of general solutions is often infeasible in practice.

Possible solutions

- Split into subtasks, each containing a single head declaration and a set of abducibles for all head declarations in the connected component.

Imperial College London

Future work

Main setback of our approach

For a given ILP task, the split within a connected component of its reduced dependency graph is incomplete under optimal solution settings. Furthermore, the computation of general solutions is often infeasible in practice.

Possible solutions

- Split into subtasks, each containing a single head declaration and a set of abducibles for all head declarations in the connected component.
- Run RASPAL to obtain the solutions of each connected component.

Summary of contributions

- Development of a theoretical framework capturing dependencies between predicates of a given ILP task
- Definition of a split for an arbitrary ILP task into subtasks for pairwise disconnected sets of predicates; Proof of its soundness and completeness when either general or optimal solutions are considered
- Inductive definition of a split for the predicates of a connected component of the reduced dependency graph; Proof of its soundness in terms of optimal solutions; Proof of its soundness and completeness in terms of general solutions
- Construction of an algorithm encompassing both splits and its implementation in Python.
- Assessment of the performance of our Python script in comparison with other existing approaches at parallelizing ILP and ASPAL itself.