Learning with Knowledge

Viktor Schlegel

What about knowledge?

So far, we have regarded learning in a pure inductive setting. We pursued the following question:

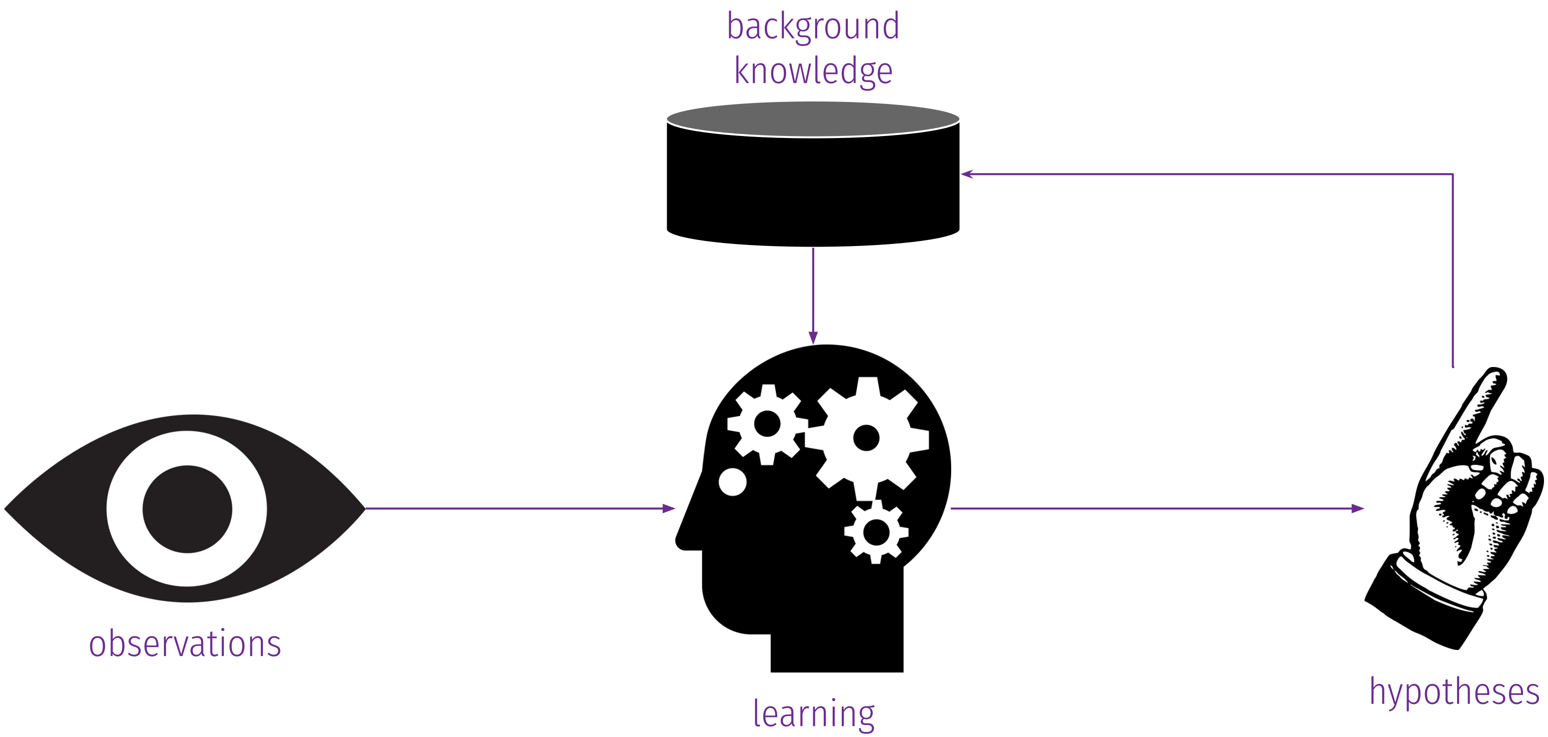
“Assuming nothing and looking at these examples, what general rules can we learn?”

This is not how we (humans) learn!

When learning new things, we already possess some prior knowledge, which we can make use of in order to learn better and faster.

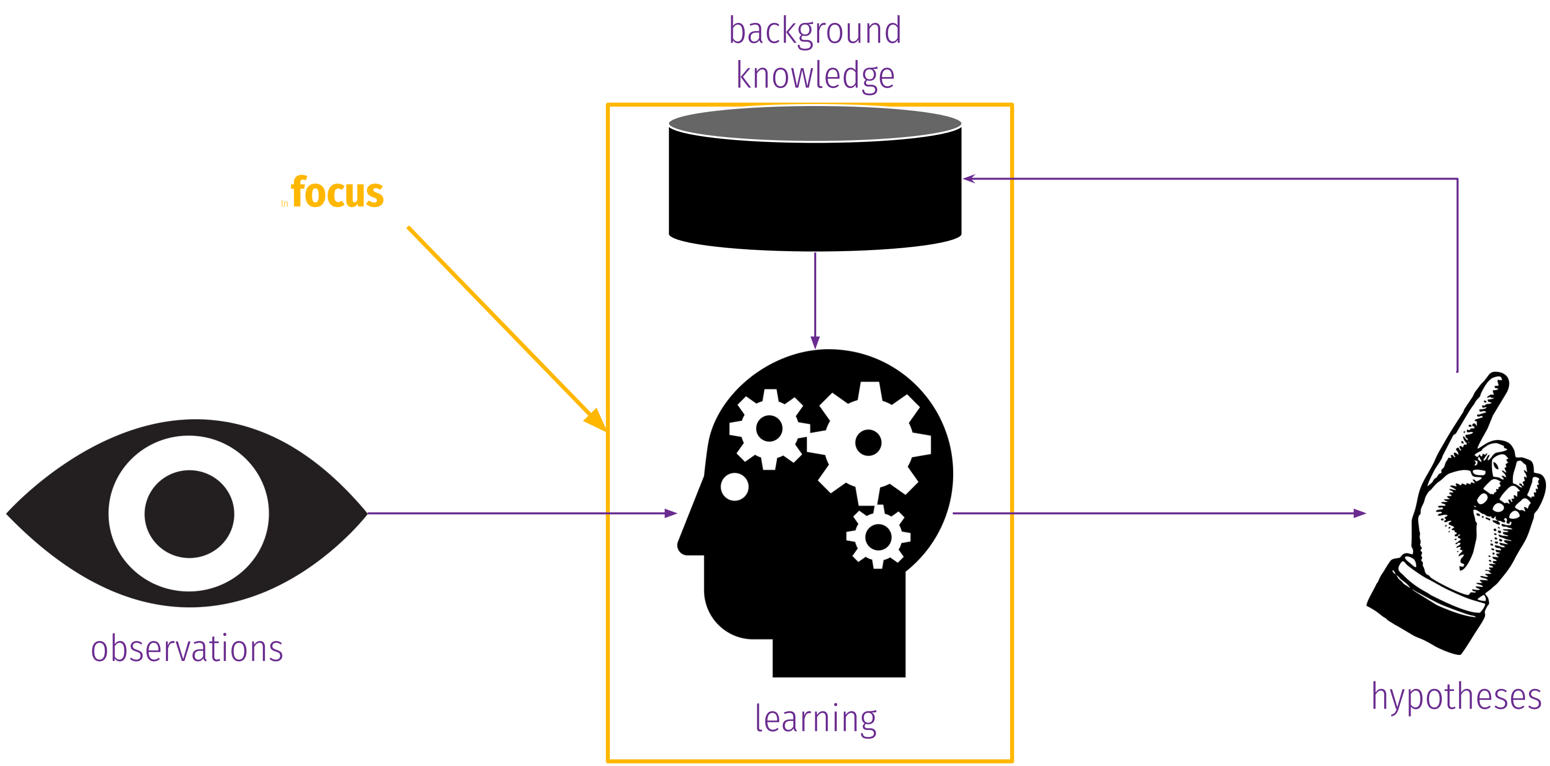
Learning, idealised

An AI agent should be able to use prior knowledge as well.



Learning, idealised

An AI agent should be able to use prior knowledge as well.



Using knowledge

When learning how to solve quadratic equations in high school, you might have encountered an example like this:

$$\begin{aligned} & 2x^2 + 8x + 16 \\ = \; & 2(x^2 + 4x + 8) \\ = \; & 2((x+2)^2 + 4) \end{aligned}$$

Which, when substituting the numbers for variables results in

$$ax^2 + bx + c = a(x + \tfrac{b}{2a})^2 + c - \tfrac{b^2}{4a}$$

And finding x in f(x) = 0 results in

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 + 4ac}}{2a}$$

Using knowledge

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 + 4ac}}{2a}$$

From there on, to solve for x you would use this formula, that you learned from the example, without deriving it every time.

You didn't learn anything "new" in the process, you just deduced a shortcut using your existing knowledge (of arithmetics).

Explanation based learning

We use the term "explanation" quite liberally here, referring to a proof. We will look at explanations more formally later.

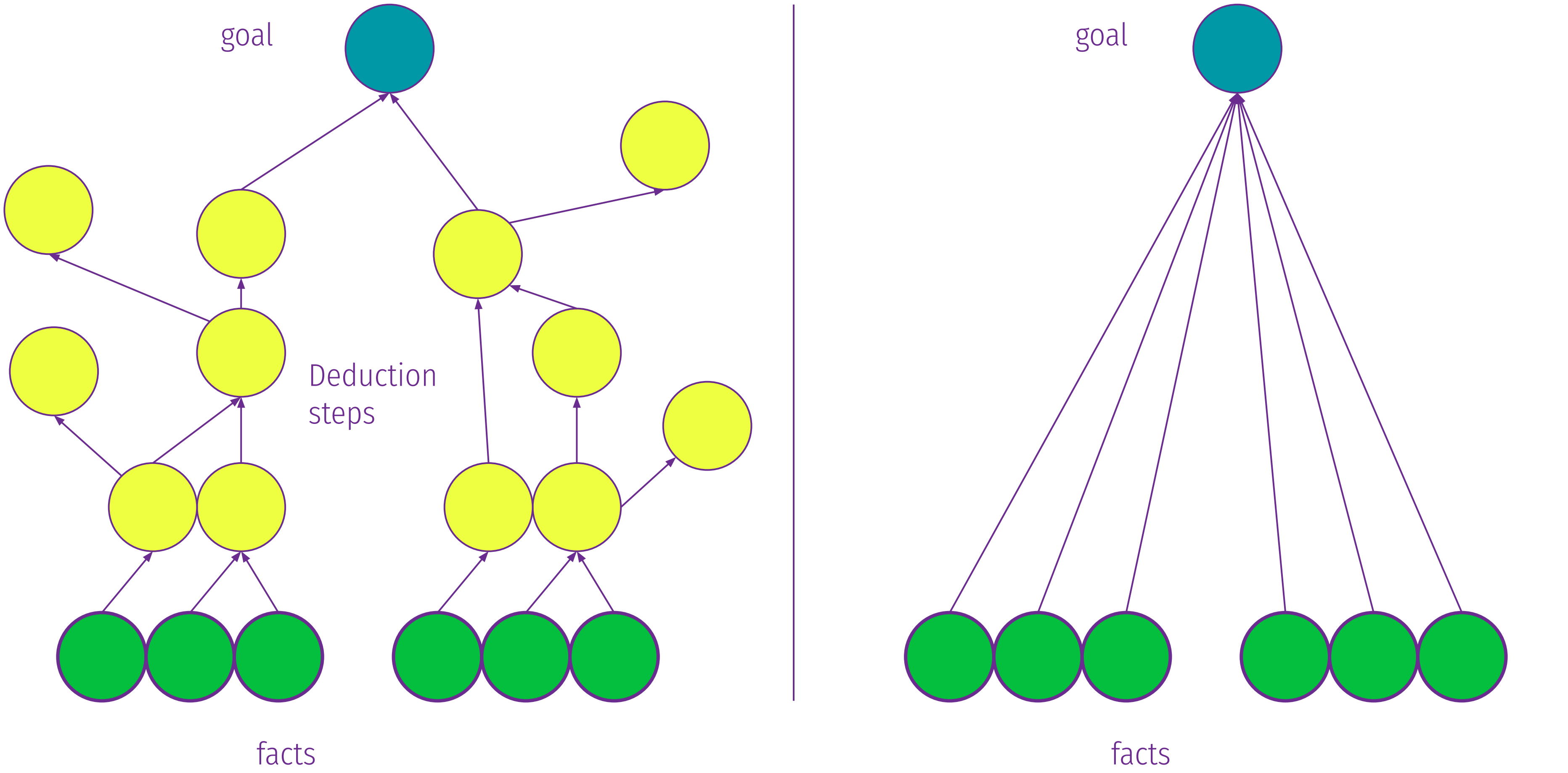
The AI equivalent for this process is called **Explanation based learning** (EBL).

EBL aims to extract **general rules** from detailed examples, in order to apply them to a class of similar problems.

These general rules speed up the search for a proof for a problem where those rules apply.

$$Hypothesis \wedge Descriptions \models Classifications$$
$$Background \models Hypothesis$$

Explanation based learning



The University of Manchester

EBL example

Consider the following prolog KB (excerpt) that describes some simple arithmetic rules.

We will derive the proof for the simplification of

$$1 \times (0 + y)$$

```
simplify(U,W) :- rewrite(U,V) ,simplify(V,W).

simplify(U,U) :- primitive(U).

primitive(U) :- variable(U).
primitive(U) :- nr(U).

rewrite(1 x U, U).
rewrite(0 p U, U).
....
variable(y).
....
```

Infix for: x(1, U)

Proof SLD

```
[trace]  ?- simplify(1 x (0 p y), X).
   Call: (10) simplify(1 x (0 p y), _29874) ? creep
   Call: (11) rewrite(1 x (0 p y), _30326) ? creep
   Exit: (11) rewrite(1 x (0 p y), 0 p y) ? creep
   Call: (11) simplify(0 p y, _29874) ? creep
   Call: (12) rewrite(0 p y, _30458) ? creep
   Exit: (12) rewrite(0 p y, y) ? creep
   Call: (12) simplify(y, _29874) ? creep
   Call: (13) rewrite(y, _30590) ? creep
   Fail: (13) rewrite(y, _30634) ? creep
   Redo: (12) simplify(y, _29874) ? creep
   Call: (13) primitive(y) ? creep
   Call: (14) variable(y) ? creep
   Exit: (14) variable(y) ? creep
   Exit: (13) primitive(y) ? creep
   Exit: (12) simplify(y, y) ? creep
   Exit: (11) simplify(0 p y, y) ? creep
   Exit: (10) simplify(1 x (0 p y), y) ? creep
X = y .
```

simplify(1 x (0 p y), W)

rewrite(1 x (0 p y), V)

Done | V := 0 p y

simplify((0 p y), W)

rewrite((0 p y), V2)

Done | V2 := y

simplify(y, W)

W := y

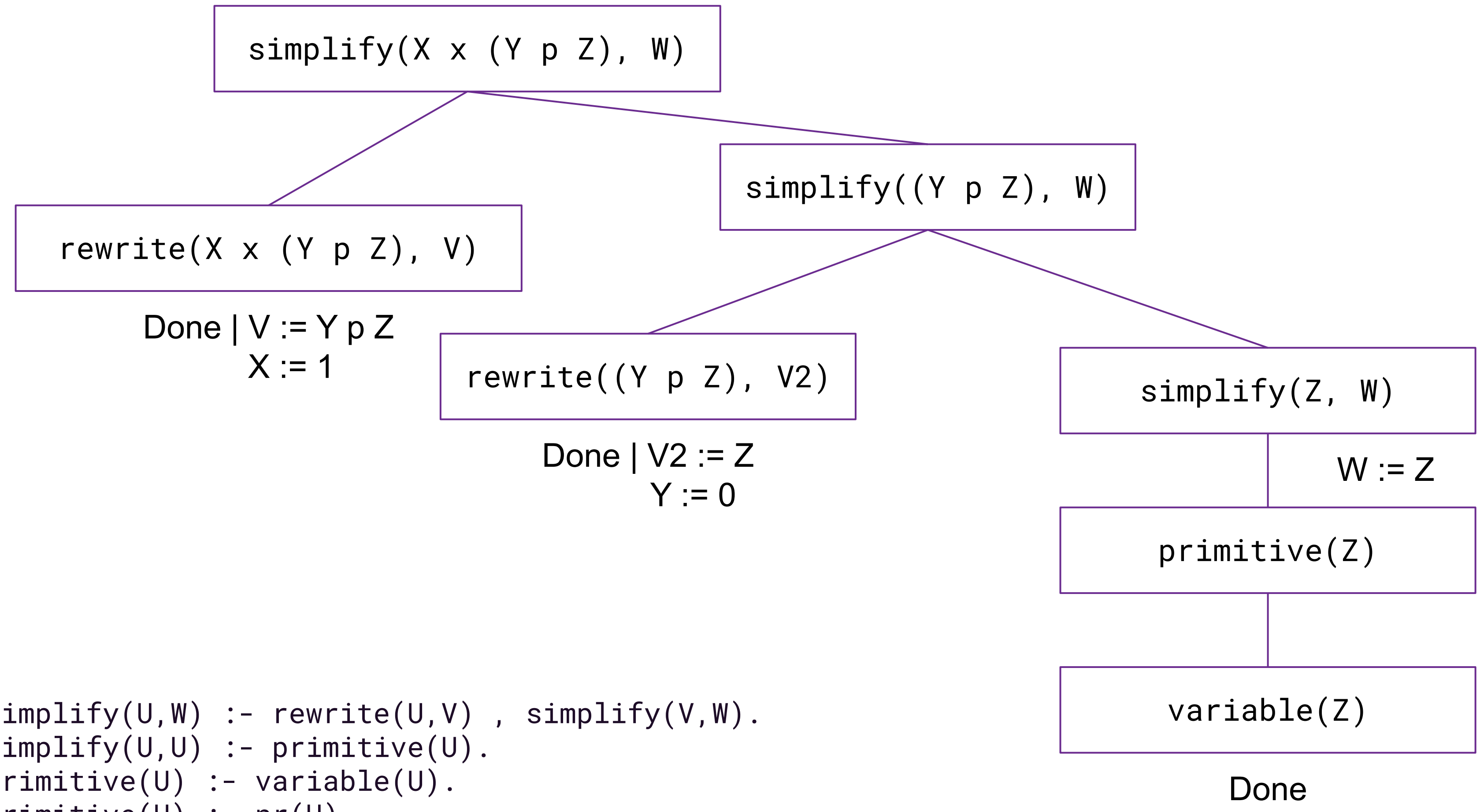
primitive(y)

variable(y)

Done

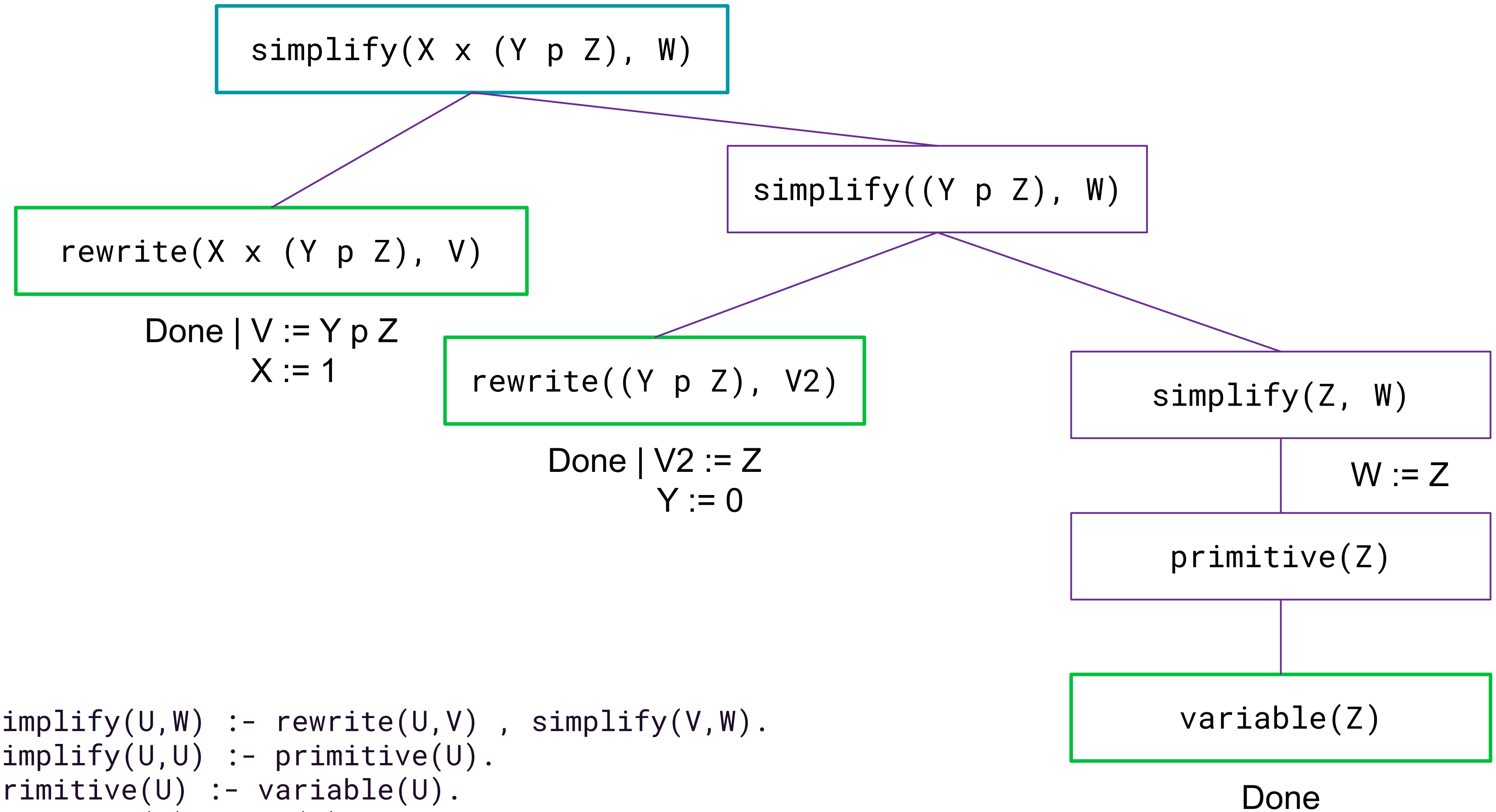
```
simplify(U,W) :- rewrite(U,V) , simplify(V,W).
simplify(U,U) :- primitive(U).
primitive(U) :- variable(U).
primitive(U) :- nr(U).
rewrite(1 x U, U).
rewrite(0 p U, U).
....
variable(y).
....
```

Generalised SLD




```
simplify(U,W) :- rewrite(U,V) , simplify(V,W).
simplify(U,U) :- primitive(U).
primitive(U) :- variable(U).
primitive(U) :- nr(U).
rewrite(1 x U, U).
rewrite(0 p U, U).
....
variable(y).
....
```

Generalised SLD




```
simplify(U,W) :- rewrite(U,V) , simplify(V,W).
simplify(U,U) :- primitive(U).
primitive(U) :- variable(U).
primitive(U) :- nr(U).
rewrite(1 x U, U).
rewrite(0 p U, U).
....
variable(y).
....
```

New rule

```
simplify(X x (Y p Z), W)
```

```
rewrite(X x (Y p Z), V)    rewrite((Y p Z), V2)    variable(Z)
```

V := Y p Z, X := 1 V2 := Z, Y := 0 W := Z

```
simplify(1 x (0 p Z), Z) :- rewrite(1 x (0 p Z), 0 p Z) , rewrite(0 p Z, Z) , variable(Z) .
```

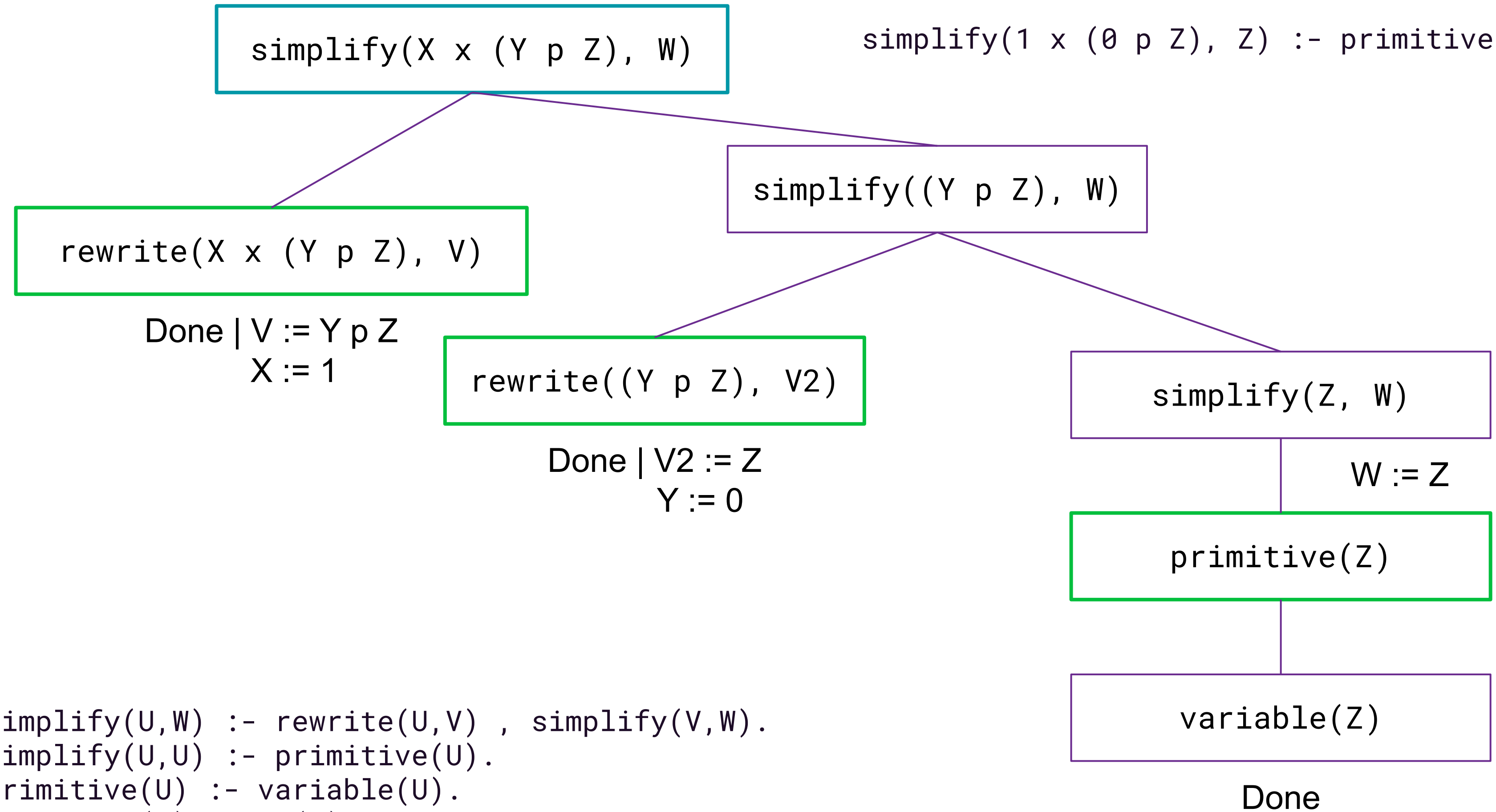
true, regardless of value of Z

```
simplify(1 x (0 p Z), Z) :- variable(Z) .
```

```
[...]
rewrite(1 x U, U).
rewrite(0 p U, U).
[...]
```

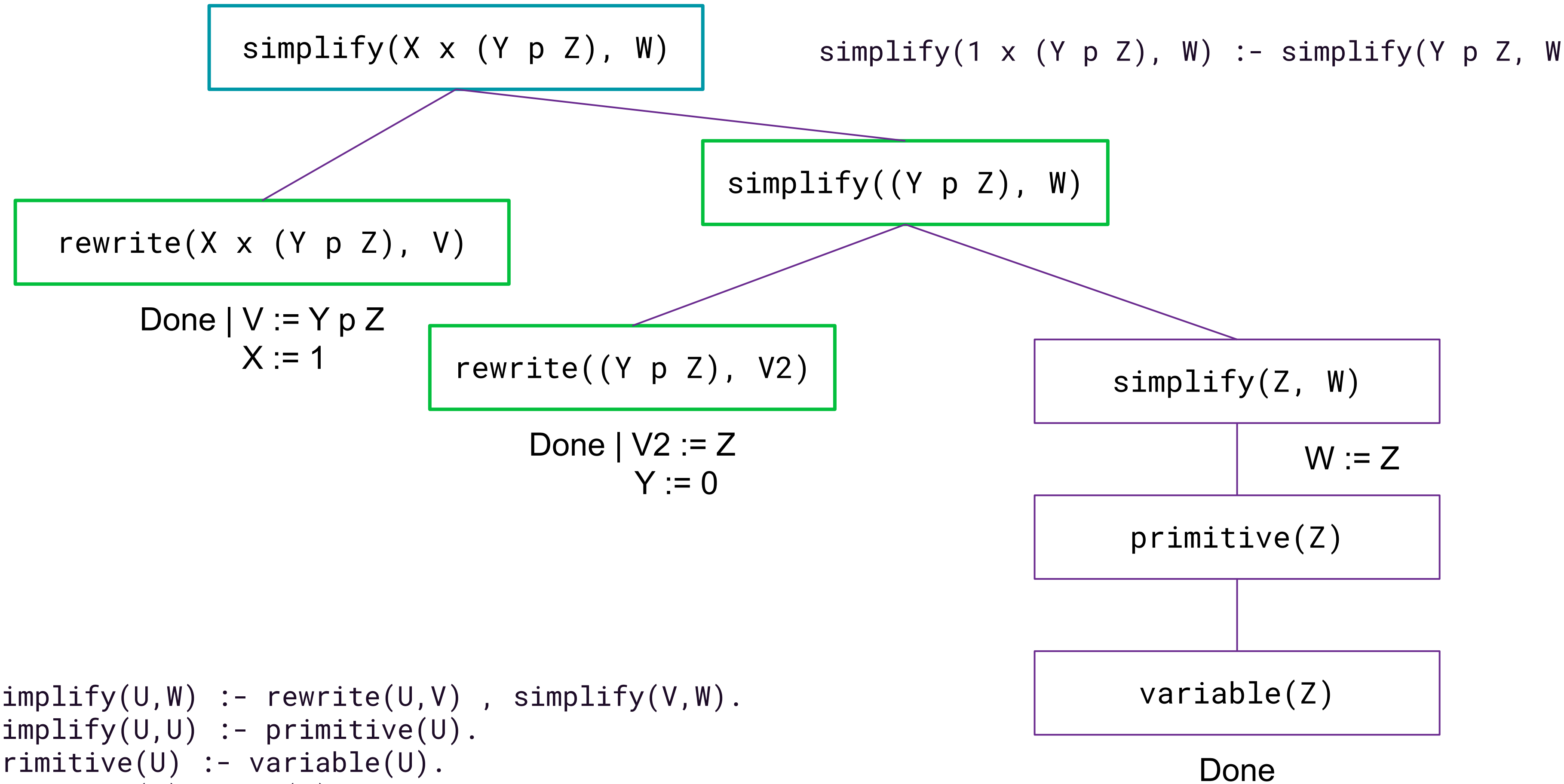
Generalised SLD

```
simplify(1 x (0 p Z), Z) :- primitive(Z)
```




```
simplify(U,W) :- rewrite(U,V) , simplify(V,W).
simplify(U,U) :- primitive(U).
primitive(U) :- variable(U).
primitive(U) :- nr(U).
rewrite(1 x U, U).
rewrite(0 p U, U).
....
variable(y).
....
```

Generalised SLD




```
simplify(U,W) :- rewrite(U,V) , simplify(V,W).
simplify(U,U) :- primitive(U).
primitive(U) :- variable(U).
primitive(U) :- nr(U).
rewrite(1 x U, U).
rewrite(0 p U, U).
....
variable(y).
....
```


MANCHESTER 1824 The University of Manchester

EBL Algorithm Sketch

```
simplify(U,W) :- rewrite(U,V), simplify(V,W).
simplify(U,U) :- primitive(U).
primitive(U) :- variable(U).
primitive(U) :- nr(U).
rewrite(1 p U), U).
rewrite(0 x U), U).
....
variable(y).
....
```

Given:

- Domain Theory (rules)
- Situation Description (facts)
- Example (statement to derive with rules & facts) $1 \times (0 + y) \Leftrightarrow y$
- "Operationality criterion"

Other terms that can appear in the generalised rule

Result:

- Rule consistent with the example but is more general that the proof used to prove example

EBL Algorithm Sketch

1. Construct proof for the example using background knowledge
2. Construct a generalised proof, substituting the literals in the proof goal with variables, keep track of instantiated variables
3. Construct a new rule from generalised tree: body are leaves of the proof tree and head is the root, substitute variables for their instantiations
4. Drop conditions from the body that are true, regardless of the values of the variables in the head

How is that learning?

Did we learn anything new?

Didn't we just deduce the rule from what was from our knowledge?

In a perfect world, the deductive system would already know everything in its knowledge base closure.

In the real world, there are constraints, such as time, memory, etc.

EBL to speed up inference

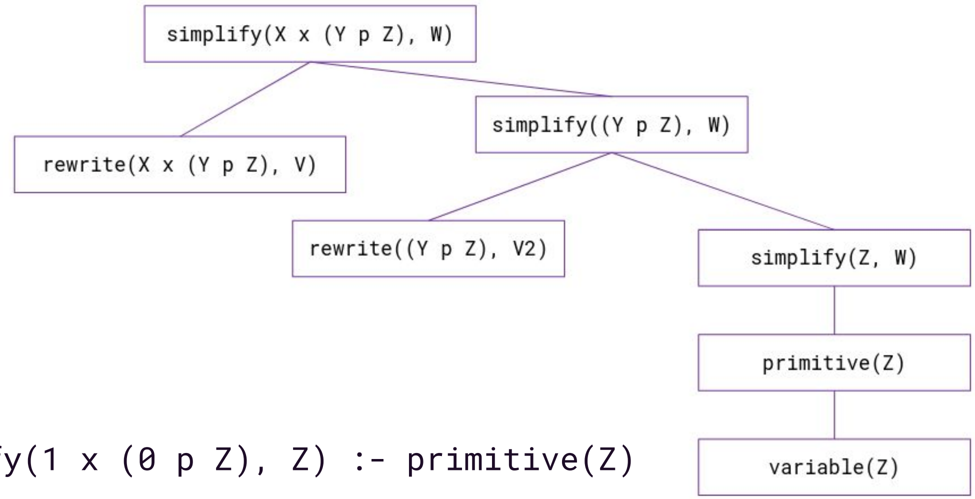
```
[trace]  ?- simplify(1 x (0 p y), X).
   Call: (10) simplify(1 x (0 p y), _29874) ? creep
   Call: (11) rewrite(1 x (0 p y), _30326) ? creep
   Exit: (11) rewrite(1 x (0 p y), 0 p y) ? creep
   Call: (11) simplify(0 p y, _29874) ? creep
   Call: (12) rewrite(0 p y, _30458) ? creep
   Exit: (12) rewrite(0 p y, y) ? creep
   Call: (12) simplify(y, _29874) ? creep
   Call: (13) rewrite(y, _30590) ? creep
   Fail: (13) rewrite(y, _30634) ? creep
   Redo: (12) simplify(y, _29874) ? creep
   Call: (13) primitive(y) ? creep
   Call: (14) variable(y) ? creep
   Exit: (14) variable(y) ? creep
   Exit: (13) primitive(y) ? creep
   Exit: (12) simplify(y, y) ? creep
   Exit: (11) simplify(0 p y, y) ? creep
   Exit: (10) simplify(1 x (0 p y), y) ? creep
X = y .
```

```
[trace]  ?- simplify(1 x (0 p y), X).
   Call: (10) simplify(1 x (0 p y), _15692) ? creep
   Call: (11) variable(y) ? creep
   Exit: (11) variable(y) ? creep
   Exit: (10) simplify(1 x (0 p y), y) ? creep
X = y .
```

```
simplify(1 x (0 p Z), Z) :- variable(Z)
```


MANCHESTER 1824
The University of Manchester

Which rules to add?



If we add too many rules to our KB, we increase the search space, perhaps unnecessarily.

We would like to know the "utility" of a rule, before adding it.

```
simplify(1 x (0 p Z), Z) :- primitive(Z)

simplify(1 x (Y p Z), W) :- simplify(Y p Z, W)

simplify(1 x (0 p Z), Z) :- variable(Z)
```

Time saved when using this rule

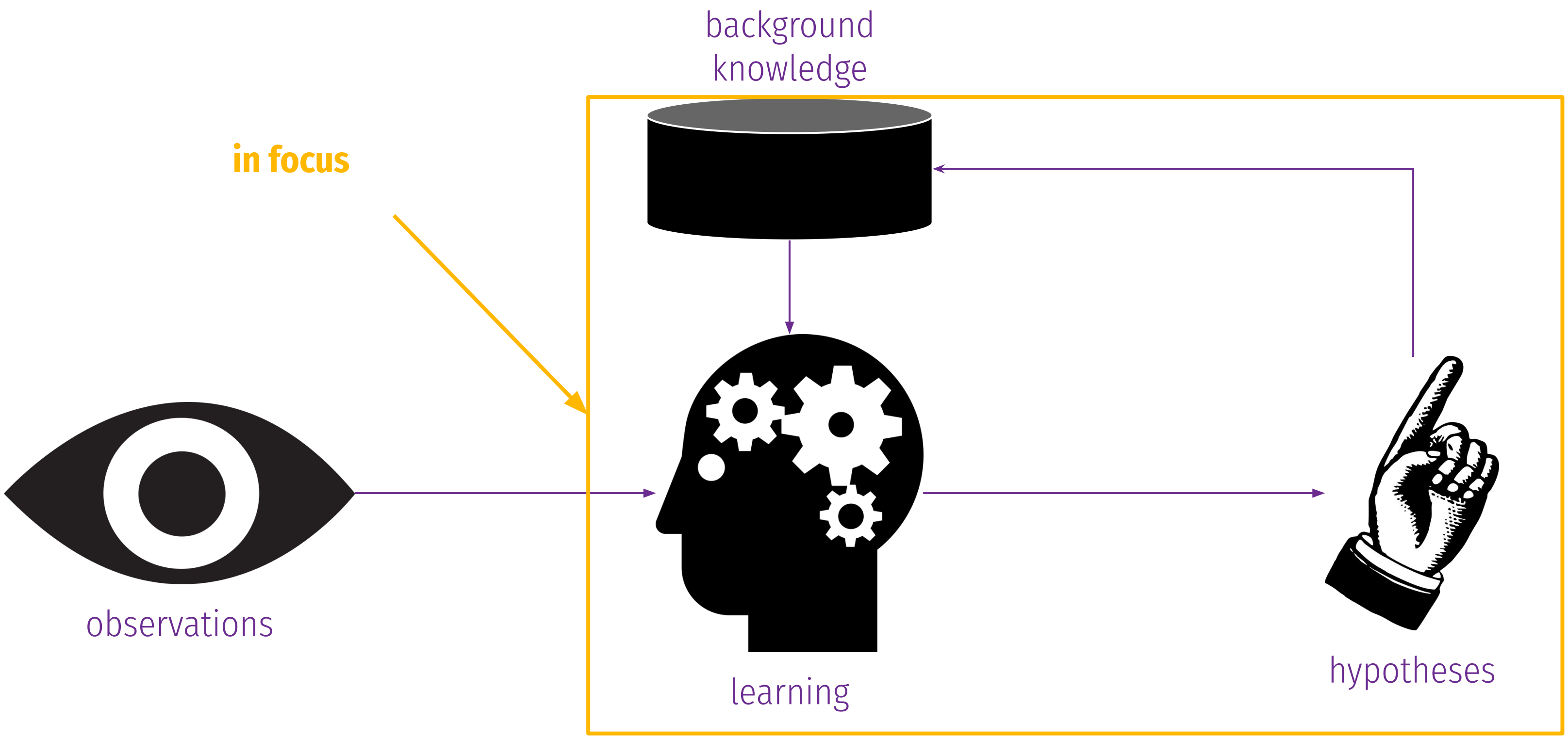
How long it takes to check the body

`Utility = (timeSaved * frequency) - matchCost`

Probability of the rule succeeding if we check the body

Learning, idealised

An AI agent should be able to use prior knowledge as well.



Consider another example:

When you need to predict, whether a block will float in water, you ask for the material and not necessarily for its size or its finish.

The reason for this is because you **know** that whether an object will float depends on its density (which in turn is dictated by the object's material).

Determinations

In other words, knowing the material, the “floatiness” of a block can be fully determined.

This is called a **functional dependency** or **determination**.

$Material(x, m) \succ Floats(x)$

The University of Manchester

Determining the hypothesis space

Determinations provide a **sufficient set of attributes** that need to be considered when constructing hypotheses.

This is useful to reduce the dimensionality of the hypothesis space.

- Speed up learning (smaller search space)
- Learn from fewer examples
- In spirit, this is similar to **feature selection techniques**

$$Hypothesis \land Descriptions \models Classification$$
$$Background \land Descriptions \land Classifications \models Hypothesis$$

Learning determinations

In practice, for a learning problem we might not know the determinations. Therefore, we want to **learn** them from observations.

A determination $\bigwedge_{i=0}^{n} A_i \succ C$ is **consistent** with a set of examples,

if all examples with the same values for all A_i have the same value for C.

A consistent determination of size *n* is **minimal** if there is no consistent determination of smaller size.

Minimal Consistent Determinations

```
def is_consistent(attributes, examples):
  h := dict()
  for e in examples:
     if any(class of e =! c for c in h[e[attributes]):
        return false
     h[e[attributes]] = class of e
  return true
```

map from attribute values to class of example

Check if there are any examples with the same attribute combination but different classifications

```
def minimal_consistent_determinations(attributes,
example):
   for subset in sorted(powerset(attributes), key=len):
    if is_consistent(subset, examples):
       return subset
```

All possible subsets of attributes

$\Rightarrow$ Runtime $\mathcal{O}(|examples|^n)$ in theory, in practice, in many domains n is sufficiently small.

Takeaways

- We can exploit prior knowledge to improve the learning
- Explanation based learning learns general rules from specific observations by using deductive reasoning
- Relevance based learning relies upon/learns functional dependencies to reduce the size of the hypothesis space
- We still didn't learn anything "complicated!"

Learning relations

Viktor Schlegel

What about relations?

So far, we have regarded learning as a classification problem.

$$\forall x : Goal(x) \Leftrightarrow Attributes(x)$$

This is a restricted view of learning.

Specifically we have only one variable in the head and the body of the learned rule.

Representing relations

Let's consider learning a relation like the following:

$$Grandparent(x, y) \Leftrightarrow [\exists z \;\; Parent(x, z) \wedge Parent(z, y)] \, .$$

How can we represent some examples for this relation using the attribute-based notation?

MANCHESTER 1824
The University of Manchester

Representing relations

First problem: how do we represent the target relation to be learned as an attribute?

- Represent pairs as single objects, i.e

  Grandparent(<Elsa, Viktor>)
  Grandparent(<Robert, Andy>)
  ....

Second problem: how do we then represent the relations between elements of the pair?

→ Introduce new attributes

SecondElementIsMale(<Elsa, Viktor>)
FirstElementIsFatherOfElena(<Robert, Andy>)
....

Representing relations II

Let’s substitute the question for an easier one:

Is X the grandparent of Y → Has X a grandchild?

Parent (X)	**Child (Y)**	**HasGrandChild**
Elena	Olga	True
Olga	Sophie	False

Something is obviously missing

So far, we have only considered algorithms that process example independently. This is not given in this example!

Representing relations III

Let's manually flatten out the relation.

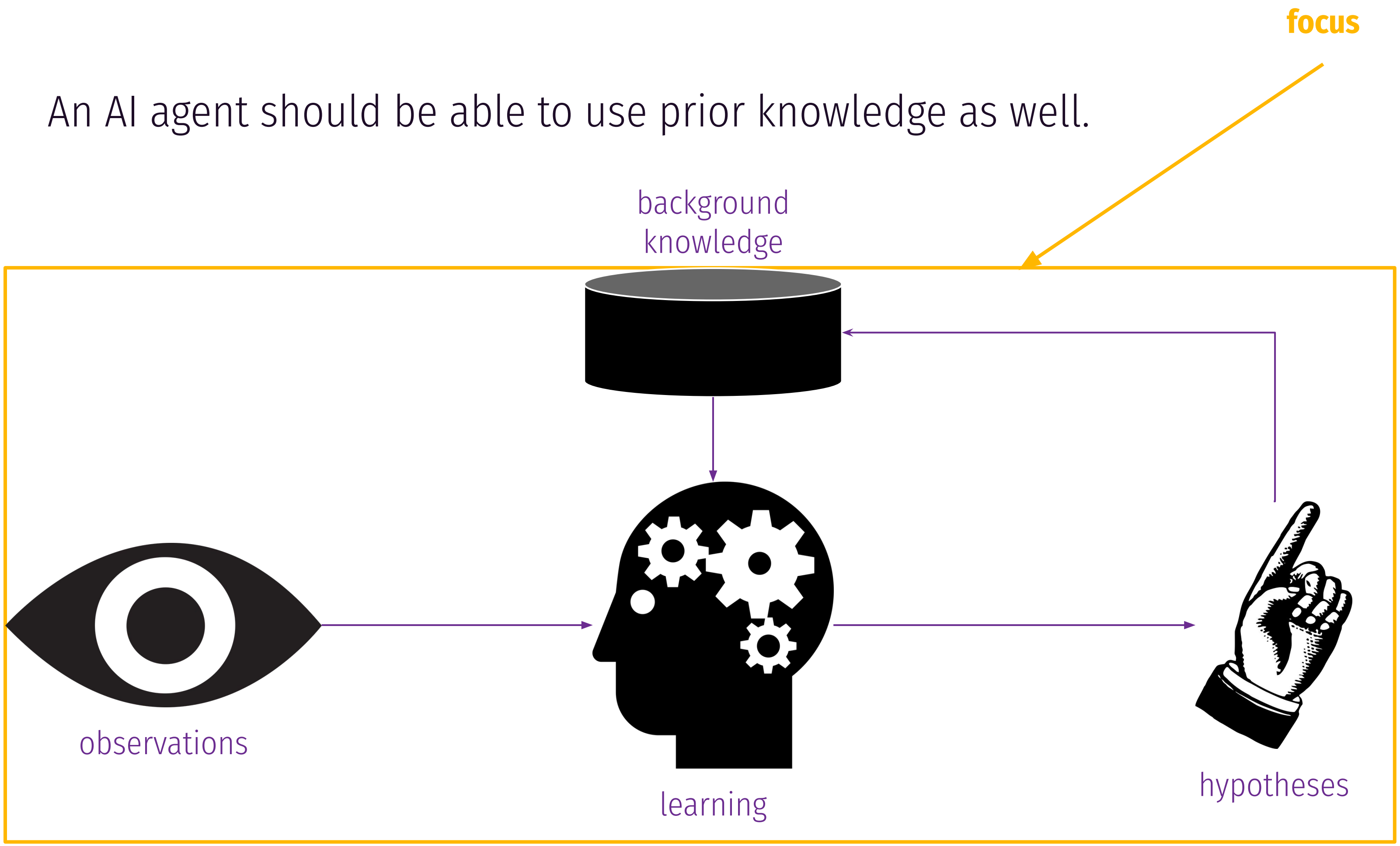
Parent (X)	**Child (Y)**	**ChildOfChild**	**HasGrandChild**
Elena	Olga	Sophie	True
Olga	Sophie	NULL	False

Then the search for a hypothesis becomes trivial.

We will see later how this type of transformation can be achieved automatically.

Learning, idealised

An AI agent should be able to use prior knowledge as well.



The University of Manchester

Learning relations

Given:

- Knowledge base
- Target predicate with arity
- Positive examples of target relation
- Negative examples of target relation

Goal:

- Learn target relation (disjunction of horn clauses)

FOIL algorithm sketch

- Start with most general clause Target relation as head, empty body (true)
- specialise until it covers some positive examples and no negative examples
- Remove positive examples covered by hypothesis
- Repeat from the beginning until no positive examples left
- Return disjunction of clauses as hypothesis

Learn one rule

Sequential covering

FOIL Algorithm

```
def foil(pos_ex, neg_ex, predicates, target):
  clauses := set()
  while pos_examples not empty:
     clause := new_clause(pos_ex, neg_ex, predicates, target)
     pos_examples = [e for e in pos_examples if not clause |= e]
     clauses.push(clause)
  return disjunction of clauses as hypothesis

def new_clause(pos_ex, neg_ex, predicates, target):
  clause = (target, []) # (head, body)
  while neg_ex not empty:
     candidates := generate_candidates(clause)
     lit := argmax_c(foil_gain(pos_ex, neg_ex, c) for c in candidates)
     clause[1].push(new_literal)
     update pos_ex, neg_ex with all bindings for new variables in lit
  return clause
```

Generate Candidates

Similar to decision tree learning, we first look at all possible ways to specialise a clause.

Possibilities:

1. Predicate literals `grandparent(X,Y) :- parent(X, V_1)`
   a. Must be a "known" predicate from KB
   b. Arguments must be variables
   c. At least one variable must be in the rule already (head or body so far)

Generate Candidates

Similar to decision tree learning, we first look at all possible ways to specialise a clause.

Possibilities:

2. Equality `grandfather(X,Y) :- father(X, V_1), father(V_1, V_2), V_2 = Y`
   a. Between variables: both variables must be in the rule
   b. Between constants: must be provided additionally

Generate Candidates

Similar to decision tree learning, we first look at all possible ways to specialise a clause.

Possibilities:

3. Negation
   a. Negated predicate literals
   b. Negated Equality (disequality)

```
sibling(X,Y) :- parent(X,v_1), parent(Y,v_1), X != Y
```

Generate Candidates

Similar to decision tree learning, we first look at all possible ways to specialise a clause.

Other possibilities:

3. Arithmetic comparisons `liquid(X,Y) :- [...] , X > 50 .`
   a. Learn to split continuous values
4. Target predicate `ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y) .`
   a. Additional checks to prevent infinite recursion
   b. need to add target predicate to the KB at some point!

Choose candidate

Similar to decision tree learning FOIL chooses greedily the candidate specialisation that has the highest **information gain**.

```
def foil_gain(pos_ex, neg_ex, candidate):
  pos_ex_new, neg_ex_new = examples updated with bindings
                           for all new variables in candidate
  p_1, n_1, p_0, n_0 = len(pos_ex_new), len(neg_ex_new), len(pos_ex), len(neg_ex)
  t = len(p for p in pos_ex if p is represented by any p' in pos_ex_new)
  return t * (log(p_1/(p_1+n_1)) - log(p_0/(p_0+n_0)))
```

Examples covered by hypothesis and by specialised hypothesis

Entropy of hypothesis specialised with candidate literal

Entropy of current hypothesis

FOIL: example

```
mother(anne, peter) .
mother(anne, zara) .
mother(sarah, beatrice) .
mother(sarah, eugenie) .
mother(elizabeth, anne) .
mother(elizabeth, andrew) .

father(mark, peter) .
father(mark, zara) .
father(andrew, beatrice) .
father(andrew, eugenie) .
father(philip, anne) .
father(philip, andrew) .

male(philip) .
male(mark) .
male(andrew) .
male(peter) .
```

```
female(elizabeth) .
female(anne) .
female(sarah) .
female(zara) .
female(beatrice) .
female(eugenie) .
```

FOIL: Generate Candidates Example

1. Predicate literals
   a. Must be a “known” predicate from KB
   b. Arguments must be variables
   c. At least one variable must be in the rule already (head or body so far)

```
clause = parent(X,Y) :- true

predicates = [Mother/2]
```

Possible Candidates:

```
mother(X,Y)
mother(Y,X)

mother(X,X)
mother(Y,Y)

mother(X,V_0)
mother(Y,V_0)
mother(V_0,X)
mother(V_0,Y)
```

Impossible Candidates:

```
mother(V_0,V_0)

mother(V_0, V_1)
mother(V_1, V_0)

father(X,Y)
```

FOIL: Update bindings Example

Extend example with binding based on current knowledge base

```
literal = father(X, V_0)

example_1 = {'X': 'mark', 'Y': zara}

⇒ {'X': 'mark', 'Y': 'zara', 'V_0': 'zara'}

⇒ {'X': 'mark', 'Y': 'zara', 'V_0': 'peter'}

example_2 = {'X': 'elizabeth', 'Y': andrew}

⇒ []
```

```
[...]

father(mark, peter) .
father(mark, zara) .
father(andrew, beatrice) .
father(andrew, eugenie) .
father(philip, anne) .
father(philip, andrew) .

[...]
```

FOIL: Gain calculation example

```
candidate = mother(X,Y)
ex_pos = [{"X": "elizabeth", "Y": "anne"},
          {"X": "elizabeth", "Y": "andrew"}]
ex_neg = [{"X": "anne", "Y": "eugenie"},
          {"X": "beatrice", "Y": "eugenie"}]

ex_pos_new = ex_pos
ex_neg_new = []

represented = ex_pos ⇒ t = 2
                       0                -1
⇒ ig = 2 * (log2(2/(2+0)) - log2(2/(2+2))) = 2
```

```
[...]

mother(anne, peter) .
mother(anne, zara) .
mother(sarah, beatrice) .
mother(sarah, eugenie) .
mother(elizabeth, anne) .
mother(elizabeth, andrew) .

[...]
```

```
candidate = mother(X,V_0)
ex_pos = [{"X": "elizabeth", "Y": "anne"}, {“X": "elizabeth", "Y": "andrew"}]
ex_neg = [{"X": "anne", "Y": "eugenie"}, {“X": "beatrice", "Y": "eugenie"}]

ex_pos_new = [{"X": "elizabeth", "Y": "anne", “V_0”: “andrew”},
              {"X": “elizabeth", "Y": "anne", “V_0”: “anne”},
              {"X": "elizabeth", "Y": "andrew", “V_0”: “anne”},
              {"X": "elizabeth", "Y": "andrew", “V_0”: “andrew”}]
ex_neg_new = [{"X": "anne", "Y": "eugenie", “V_0”: “peter”},
              {"X": "anne", "Y": "eugenie", “V_0”: “zara”}]

represented = ex_pos ⇒ t = 2
                        -0.585           -1
⇒  ig = 2 * (log2(4/(4+2)) - log2(2/(2+2))) = 0.83
```

FOIL: Summary

- FOIL can learn relations that simple, attribute-based learning algorithms cannot learn
- FOIL can learn *recursive* relations
- FOIL learns hypotheses as a disjunction of horn clauses
  - Each clause covers a subset of positive examples in the training set
  - Specific to general search
- These hypotheses are generated by specialising, until they cover no negative examples
  - Greedy, general to specific search

FOIL: Drawbacks and Improvements

- Clauses can only be "specialised, until they cover no negative examples" if there is no noise in data
  - With noise, the refinement can take forever
  - As a solution, a stopping criterion based on the length of the current clause can be added
- The search space when generating candidate literals can be very large
  - Use domain theory (= background knowledge) to speed up search: EBL

FOCL Sketch

Given:

- Domain Theory (rules)
- Situation Description (facts)
- Example (statement to derive with rules & facts)
- "Operationality criterion"

In addition to training set, FOCL uses an (incomplete) domain theory & operational criterion as additional inputs

- When generating candidates, FOCL first generates logically sufficient conditions for the target based on the domain theory
- Unnecessary literals are pruned
- The resulting condition is added as a conjunction to the set of all candidates

FOCL Example

```
concavityPointsUp(object8).
fragile(object8).
hasConcavity(object8).
madeOfCeramic(object8).
negative_ex = [{"X" : 'object8'}...]
```

$$
\begin{aligned}
Cup &\leftarrow Stable,\ Liftable,\ OpenVessel\\
Stable &\leftarrow BottomIsFlat\\
Liftable &\leftarrow Graspable,\ Light\\
Graspable &\leftarrow HasHandle\\
OpenVessel &\leftarrow HasConcavity,\ ConcavityPointsUp
\end{aligned}
$$

Operational criterion

Domain Theory

	Cups				Non-Cups					
BottomIsFlat	✓	✓	✓	✓	✓	✓	✓			✓
ConcavityPointsUp	✓	✓	✓	✓	✓		✓	✓		
Expensive	✓		✓				✓		✓	
Fragile	✓	✓			✓	✓		✓		✓
HandleOnTop					✓		✓			
HandleOnSide	✓			✓					✓	
HasConcavity	✓	✓	✓	✓	✓		✓	✓	✓	✓
HasHandle	✓			✓	✓		✓		✓	
Light	✓	✓	✓	✓	✓	✓	✓		✓	
MadeOfCeramic	✓				✓		✓	✓		
MadeOfPaper				✓					✓	
MadeOfStyrofoam		✓	✓			✓				✓

FOCL Example

Domain Theory

$$
\begin{aligned}
Cup &\leftarrow Stable,\ Liftable,\ OpenVessel\\
Stable &\leftarrow BottomIsFlat\\
Liftable &\leftarrow Graspable,\ Light\\
Graspable &\leftarrow HasHandle\\
OpenVessel &\leftarrow HasConcavity,\ ConcavityPointsUp
\end{aligned}
$$

Operational criterion

BottomIsFlat
ConcavityPointsUp
Expensive
Fragile
HandleOnTop
HandleOnSide
HasConcavity
HasHandle
Light
MadeOfCeramic
MadeOfPaper
MadeOfStyrofoam

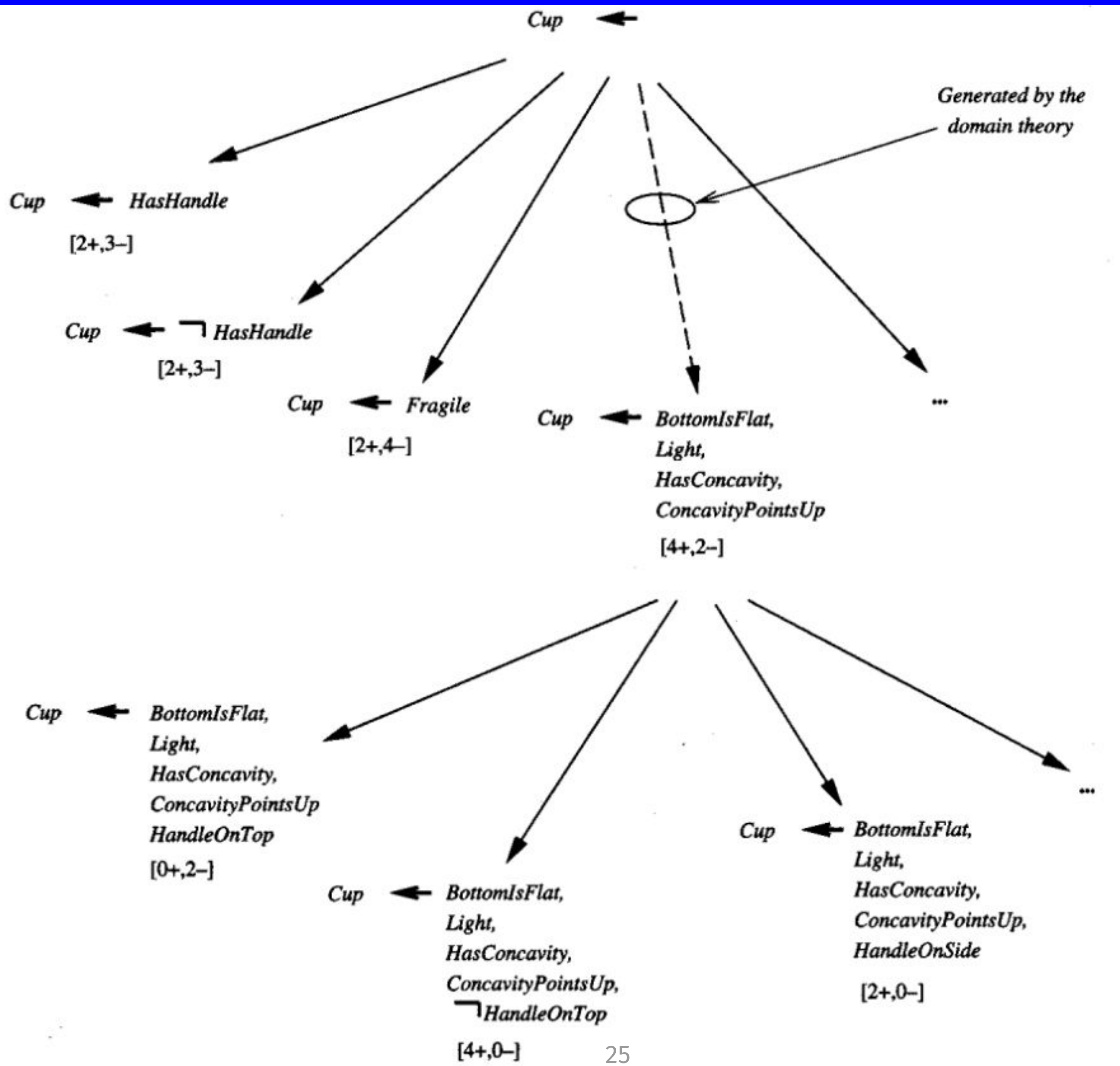
$$Cup \leftarrow BottomIsFlat, \boxed{HasHandle}, Light, HasConcavity, ConcavityPointsUp$$

Training Set

	Cups				*Non-Cups*					
BottomIsFlat	√	√	√	√	√	√	√			√
ConcavityPointsUp	√	√	√	√	√		√	√		
Expensive	√		√				√		√	
Fragile	√	√			√	√		√		√
HandleOnTop					√		√			
HandleOnSide	√			√					√	
HasConcavity	√	√	√	√	√		√	√	√	√
HasHandle	√			√	√		√		√	
Light	√	√	√	√	√	√	√		√	
MadeOfCeramic	√				√		√	√		
MadeOfPaper				√					√	
MadeOfStyrofoam		√	√			√				√

$$Cup \leftarrow BottomIsFlat, Light, HasConcavity, ConcavityPointsUp$$

FOCL Example



Induction as inverted deduction

Induction is just inverted deduction!

$$Hypothesis \wedge Descriptions \wedge Background \models Classifications$$

Invert resolution to arrive at hypothesis starting from example observations

Current Subgoal

from knowledge base, generated by other resolutions, etc..

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1 \theta_1 \theta_2^{-1}\}$$

New subgoal

Induction as inverted deduction

- Inverted deduction is utilising inverted resolution and learns a hypothesis starting from single examples, “bottom up”
- Like FOCL, can make use of domain theory to guide and speed up the search
- Like many other algorithms we observed, unsuitable to handle noisy data

Current Subgoal

from knowledge base, generated by other resolutions, etc..

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1 \theta_1 \theta_2^{-1}\}$$

New subgoal

Statistical vs Symbolic Approaches to learning & AI

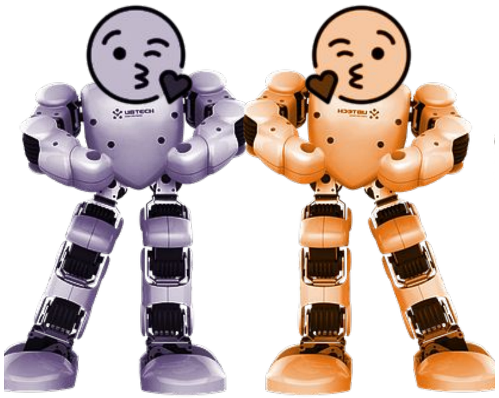
Statistical AI

- Focus on learning
- Performs well with noisy training data
- Can operate on raw input (text, pictures)
- Requires minimal human interaction

Symbolic AI

- Focus on inference
- Can be verified formally
- Produces results interpretable by humans
- Allows to incorporate existing knowledge intuitively

Statistical + Symbolic Approaches to learning & AI



- ➔ Use statistical learning approaches to learn to guide the search in large hypothesis spaces
  - E.g. in Automated Theorem Provers (ATP)
- ➔ Use statistical learning approaches to generate symbolic representations from noisy input data
  - Semantic Parsing
  - Scene graph generation
- ➔ Utilise knowledge to bootstrap statistical learning (e.g. in recommender systems)
- ➔ Use statistical models to learn to operate over symbolic representations