Course: 304 Logic-Based Learning

Hybrid Abductive Inductive Learning

- Combining bottom-up and top-down search
  - » *Observation predicate learning (OPL)*
  - » *Non-observation predicate learning (NOPL)*
  - » *Progol5 and incompleteness*
  - » *Abduction and Induction Cycle*
- Hybrid Abductive Inductive Learning (HAIL)
  - » *Generalised Bottom Set Semantics*
  - » *Algorithm*
  - » *Examples*

© Alessandra Russo Unit 4 – HAIL, slide 1

So far we have introduced the notion of Inductive Logic Programming and related to the well established notion of concept learning in both data mining and machine learning. We have seen that in the context of concept learning various rule-based algorithms have been proposed, which aim at learning a concept in terms of rules that can be extracted using a version space and a notion of coverage of the observed data. We have seen that we can have "top-down" search for hypothesis (rules), using a notion of θ-subsumption, as it is the case of systems called FOIL and HYPER, or "bottom-up" search from specific to general using notions such as least general generalisation or inverse resolution. The latter is used in systems like CIGOL. The first case can be seen as applying a logic principle (θ-subsumption) to an induction context (version space), whereas in the second case it is more like applying induction to a richer logic-based context (inverse resolution is the richer logic-context) and induction is the process of inductively searching for general rules via steps of inverse resolution.

A third approach was proposed in the early '90. It combines top-down and bottom-up mechanisms by applying a notion of **inverse entailment** (instead of inverse resolution) and **general-to-specific** search through a refinement lattice of θ-subsumption relations between possible hypothesis. This new approach was put forward by Professor Muggleton in a system called **PROGOL**. The focus was to support the learning of concepts whose instances are directly observed (i.e. *observation predicate learning)*.

In this lecture we will present briefly the notion of observation predicate learning and the semantic notion of Bottom Generalisation (or Generalised Bottom Set Semantics) used in Progol. We then introduce the notion of *non-observation predicate learning* and describe Progol5, extension of Progol, which was developed to address the problem of non-observation predicate learning. We will show however limitations of this system in learning concepts that are not directly observed and show how a much more powerful learning approach can be achieved by integrating abductive and inductive inference. This integrated (or hybrid) approach, called **HAIL**, was developed by myself, Krysia Broda and Oliver Ray in the 2004. It extended the boundary of the state-of-the-art of logic-based learning but it has also shed light upon the open problem of how to capture computationally the philosophical notion of "abduction and induction cycle" for knowledge discovery.

A good reading on the HAIL approach is the paper "Hybrid Abductive Inductive Learning: a Generalisation of Progol" , Oliver Ray, Krysia Broda and Alessandra Russo, ILP 2003, available for download from the course website.

Course: 304 Logic-Based Learning

Inverse Entailment

$B, H \models E$ iff $B, \neg E \models \neg H$

Assumes H and E to be single Horn clauses:

(h) $l_1 \vee \neg l_2 \vee \ldots \vee \neg l_n$ (¬h) $\neg l_1\theta \wedge l_2\theta \wedge \ldots \wedge l_n\theta$

(e^+) $a_1 \vee \neg a_2 \vee \ldots \vee \neg a_m$ ($\neg e^+$) $\neg a_1\partial \wedge a_2\partial \wedge \ldots \wedge a_m\partial$

Sets of Skolemised ground literals

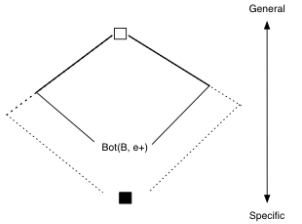
1. $B, \neg e^+ \models \underbrace{\neg l_1\partial \wedge l_2\partial \wedge \ldots \wedge l_n\partial}_{\neg Bot(B, e^+)}$

   Let's denote $Bot(B, e^+)$ as ground($h_\perp$)

   $B, ground(h_\perp) \models e^+$

2. $h \models Bot(B, e^+)$

   h θ-subsumes $Bot(B, e^+)$



© Alessandra Russo Unit 4 – HAIL, slide 2

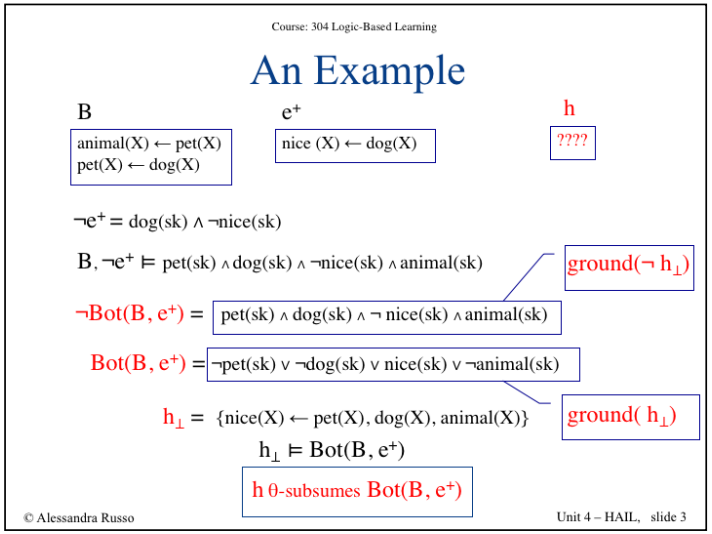
Let's consider the notion of **Inverse Entailment (IE)**, introduced by Prof. Muggleton and first implemented in PROGOL. Inverse entailment combines the advantages of both bottom-up (i.e. narrow the search by starting from what is known already) and top-down (i.e. refine a general hypothesis to form a new one) search. The idea comes from the observation that $B, H \models E$ is equivalent to $B, \neg E \models \neg H$ (note that H and E are clauses, so variables are universally quantified, and when you negate them they become existentially quantified - skolemised).

So, consider a positive example e^+. We can compute the (potentially infinite) set of ground literals that are derivable from (or entailed by) $B \wedge \neg e^+$. These are all ground literals that are true in all models of $B \wedge \neg e^+$. We refer to the set of these ground literals as the **negation of the Bottom Set** (denoted as $\neg Bot(B, e^+)$). To generate the Bottom Set $Bot(B, e^+)$, we negate every literal in $\neg Bot(B, e^+)$.

A Bottom set (denoted as **Bot(B, e^+)) is therefore the set of ground literals whose negation can be entailed by** $B \wedge \neg e^+$. We can refer to $Bot(B, e^+)$ as the ground clause **ground($h_\perp$)** , and equally we can refer to $\neg Bot(B, e^+)$ and **ground($\neg h_\perp$).** So we have that $B \wedge \neg e+ \models$ **ground($\neg h_\perp$). By Inverse entailment we have that** $B \wedge$ **ground($h_\perp$)** $\models e^+$, so **ground($h_\perp$) (or equally the Bottom Set) is the more specific ground clause that together with B explains the example** e^+.

If we replace every constant in **ground($h_\perp$)** with a unique variable we obtain a universally quantified clause $h_\perp$ that θ-subsumes the set $Bot(B, e^+)$ (i.e. $h_\perp \models Bot(B, e+)$). Hence, $h_\perp$ correspond to the most specific **unground** hypothesis at the bottom of the lattice space that θ-subsume $Bot(B, e^+)$. Computing the Bottom set first, allows for a more efficient search for hypotheses, as the search can be limited now to the sub-lattice space delimited by the $Bot(B, e^+)$ as bottom element and the empty clause as the top element. Solutions will be clauses that θ-subsume the BottomSet $Bot(B, e^+)$. So, once the unground bottom clause $h_\perp$ has been generated, the second step would be to compute more general clause that subsume this bottom set through top-down refinement.

These two steps together are referred to as "*learning by Bottom Generalisation*". **We say that an hypothesis H is derivable by bottom generalisation from B and e^+ if and only if $H \geq Bot(B, e^+)$.** For your interest you can read more about this in Muggleton's paper ["Inverse Entailment and Progol", New Generation Computing, 1995]. In the next slide we consider an example.

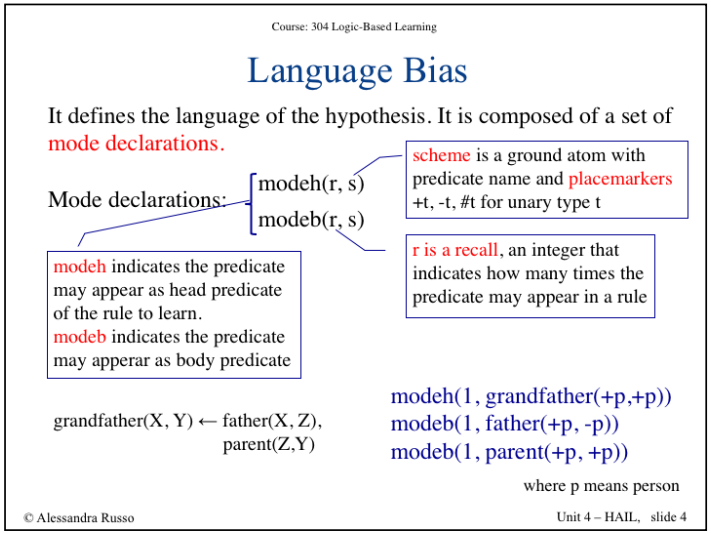


This is an example taken from Muggleton paper (mentioned in the previous slide) on Bottom Generalisation to solve a learning task. The task here is an Observation Predicate Learning (OPL) as it aims to learn the definition of a predicate that is directly observed. The task takes in input a background knowledge (B) and a single positive example (e^+). Note that this approach can only learn definite clauses. We are interested in learning a definite clause (h) that added to the background knowledge B covers the positive example (i.e. B, h ⊨ e^+).

Note that when we have more than one positive example, the computation of a final hypothesis can be done iteratively through a process of coverage loop (more on slide 9).

This slide shows how an hypothesis is derived by Bottom Generalisation from a given seed example and background knowledge. The negation of the example is added to the background knowledge. In this case the example is unground, so its negation includes skolem constants. All possible ground literals are then derived from B ∧¬e^+ to compute the set ¬Bot(B, e^+), which is a conjunction of ground literals including the skolem terms. Let ground(¬$h_\perp$) be the set ¬Bot(B, e^+). This assumes that our final solution would involve all the literals that are included in the Bottom Set (but note that this is not necessarily the case as more literals that what are needed in the final hypothesis could be derived). We then negate each literal in **ground(¬$h_\perp$)** and we get a ground clause **ground($h_\perp$)** that is the same as **Bot(B, e^+).** This is our **BottomSet**. This is the Bottom clause. We then "lift" Bot(B, e^+) to an unground clause by replacing every constant in it with a unique variable name, which gives us the clause $h_\perp$, which we more specific generalization of the Bottom Clause. This clause constitutes the bottom of the lattice of our possible hypothesis. Clearly $h_\perp$ ⊨ Bot(B, e^+). Note that any clause that subsumes now this bottom hypothesis is a possible solution for this particular learning task. So for instance **nice(X) ← animal(X)** would be another possible hypothesis as it subsumes the clause $h_\perp$.

So we would say in this case that **nice(X) ← animal(X)** is derivable by Bottom Generalisation from B and e^+.

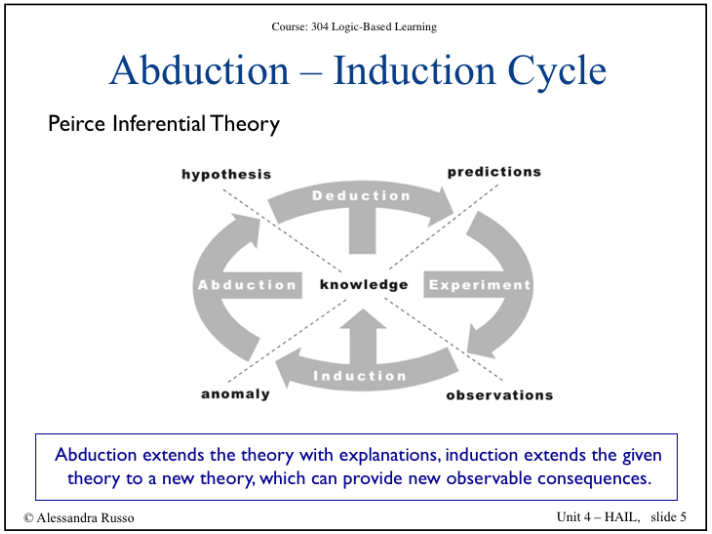


Learning using inverse entailment, and the two steps given in the notes of Slide 2, is also called **Bottom Generalisation** inference method, i.e. in essence it computes the BottomSet and generalises it using a notion of θ-subsumption lattice with the unground version of the BottomSet as the bottom clause and the empty clause as the top clause. The confine search space for potential hypothesis is clearly an advantage in the computation.

To improve even further the search for hypotheses in this lattice structure, it is possible to use a notion of **language bias**, i.e. the specification of the language in which the hypothesis is expected to be written. ILP, in fact any form of induction, may involve a potentially huge search space for hypotheses, which can render even fairly simple problems intractable. Therefore, it is important to be able to define a hypothesis language in order to limit this search space to those hypotheses that are of genuine interest.

Progol allows for the specification of such a bias in terms of what are known as **mode declarations**. A mode declaration is essentially a template for a literal that can appear in an hypothesis, specifying the predicate, the kind of terms that may appear as arguments of the predicate, and the mode of those terms. The mode of a term is either 'input', 'output' or 'constant'. Input terms are terms that are expected to be instantiated before the predicate is evaluated; output terms are terms that are expected to be instantiated as outcome of the evaluation process. For instance, in a clause of the form: path(X,Y) ← arc(X,Z), path(Z,Y), the variable terms X, Y are input variables, whereas the variable term Z is an output variable.

In the example given in this slide, we have one mode head and two mode bodies. The mode head (modeh) states that the rule to learn will have to have *grandfather* as head predicate. The two model bodies state that predicates *father* and *parent* can appear in the body of a learned rule. The schema "s" for the modeh is *grandfather(+p, +p),* which means that it takes two input variables of type person (p). The schema of one of the modeb is *parent(+p, +p)*, which takes also two input variables. But as it appears in the body, its input variable can either link to an output variable of a predicate that appear before it in the body of the rule, or link to an input variable in the head. In the example given here the input variable in the modeb *parent(+p, +p)* links to the output variable of the modeb *father(+p,-p)*. Essentially the variable Z gets instantiated as result of evaluating *father(X, Z)* and this value is passed to the input variable X of *parent* before it gets evaluated.

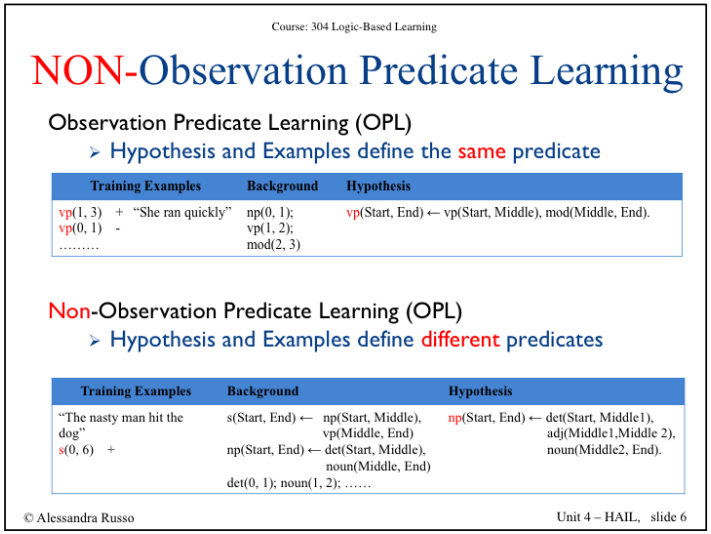


In slide 7 of Unit 2, we have given a brief example of the three types of reasoning: deduction, abduction and induction. These definitions follow what Peirce called the *syllogistic theory*. The abductive argument, in which the case is inferred from the rule and a given result, provides a plausible explanation of why the result share a certain characteristics (e.g., these beans are all white), relative to the general rule. Conversely, the inductive argument, in which the rule is inferred from the case and result, provides a probable general hypothesis of why some representative individuals (these beans) of a certain class (the beans in this bag) all share some particular characteristic (being white). The syllogistic view of induction reflects closely the notion of observation predicate learning, as it aims to learn rules about the given observations.

However, there is much more to be said about the interaction and relationship between abduction and induction in particular with respect to the problems related to *scientific knowledge discovery*. In early '90 Peirce put forward the idea of *inferential theory* where the notion of abduction and induction underwent a shift of emphasis. In this view abduction and induction are seen as complementary processes cooperating with deduction and experiment in a cycle of *scientific knowledge discovery*. The cycle starts with observations that cannot be explained using a current knowledge. Plausible hypothesis can then be found (what is here called abduction). Based on this hypothesis deductive reasoning can be used to compute predictions. These predictions must then be compared against the results of the experiments. If the predictions are confirmed by the experiments that the hypothesis is tentatively accepted. But if not, then the hypothesis maybe ruled out in favor of another, which may result in the discovery of new anomalies, itself in need of further explanations.

In essence, abduction can be seen as a step for constructing explanations about observations, by making hypothesis on missing information about the observations. Induction can then be used to generate knowledge in the form of general rules that can be added to the current theory and used to infer new observations. This new notion of cycle of abduction and induction is directly related to non-observation predicate learning.

This notion of abduction and induction cycle is at the core of **non-observation predicate learning**, as it implies the need for discovering knowledge that is not directly related to observed concepts.

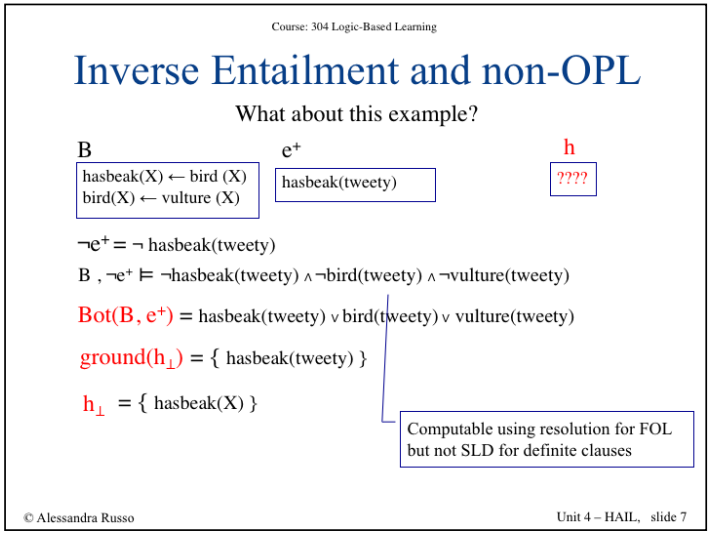


In Unit 3 we have introduced the notion of logic-based learning (and ILP) through the pure machine learning point of view of concept learning. We have shown so far examples of computing only hypotheses that define the same predicate as that used to express the examples. For example, in slide 8 of Unit 3 we had an example dataset about *EnjoySport* and the hypothesis to learn was a rule-based definition of the concept *EnjoySport*. This type of learning is called *Observation Predicate Learning (OPL)*. The head predicate of the learned rules is the same as the predicate of the observed examples. Whereas this type of learning is ingrained in Machine Learning approaches, ILP and logic-based learning advocate a more general-purpose form of learning: the *non-Observation Predicate Learning (NOPL)*.

In many application domains the knowledge about the problem domain is not complete and general (possibly interdependent) principles may be needed to be learned in order to explain observed complex phenomena. For instance, in the case of functional genomics, when new gene behaviors are observed and cannot be explained by existing biological knowledge, a NOPL task would look for general new knowledge about gene interactions that can be used together with the existing knowledge to explain the new observations. The learned rules may be rules about the specific observed phenomena, but also more general rules, such as the type of regulatory interaction (inhibition/activation) between genes depending for instance on their functions, interactive potentials etc.

A simple example is given here within the context of learning natural language grammar. The example is taken from the paper "Theory completion using inverse entailment", Stephen Muggleton, ILP 2000, available on the course website. The NOPL is particularly appropriate for learning grammar as most of the grammar of natural language is known and the idea is to refine/extend the grammar to cover for new expressions of a language.

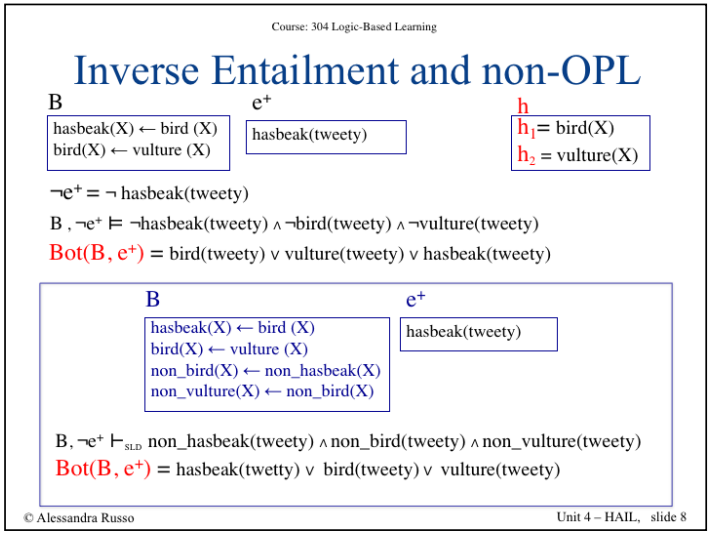
The question now is how to compute *non-observation predicate learning* in the context of *definite clauses*?



In the definition of inverse entailment (IE) and Bottom Generalisation in general, we look for bottom clauses that are definite clauses, i.e. the BottomSet has to include at most one positive literal. Clearly, since B is a definite clause all the consequences from B would be positive literals, so when negated, to form the BottomSet, they will all become negative literals. The only negated literal inferred from B∧¬e$^+$ is actually just ¬e$^+$. This literal when negated becomes positive and therefore forms the head of the bottom clause. So, you can see why IE is well suited for computing Observation Predicate Learning. *You could also see that to compute the bottom clause we need just to add the skolemised body literals of the seed example to the background knowledge as facts, generate all the positive consequences and add to this set the negated skolemised head of the given seed example. So, learning the bottom clause becomes in essence a deductive inference. The computation of "shortest" clauses that subsumes the Bottom clause, constitutes the generalisation step of the learning*. Comparing this to the cycle of abduction and induction shown in slide 5, we can see that this process is ok if we assume a complete background knowledge. Progol and OPL do that.

But consider the example given in this slide. In this case, the set of consequences from B ∧ ¬e$^+$ includes more than one negated literal. So, if we use full first-order logic resolution, we would be able to compute the full set of consequences. And in more complex examples, generate heads of clauses that refer to predicates that are not observed (e.g. bird(X), or vulture(X)). But, if we use SLD, or Prolog , we cannot compute all these consequences, since Prolog would only derive positive literals.

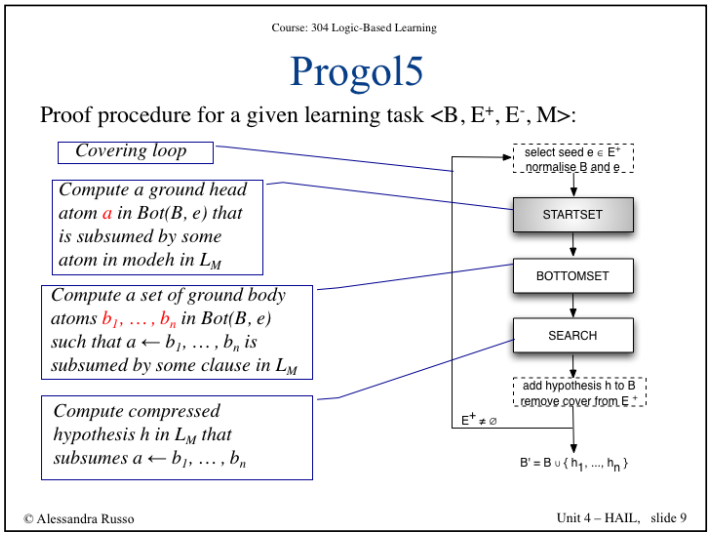
This shows that, theoretically, the principle of inverse entailment would be applicable, but the computational procedure described so far would fail to compute hypothesis about predicates that are not directly observed. Progol system, in fact, returns in this case just {*hasbeak(tweety)*} as Bottom Set, and therefore generate only rules about the predicates that are observed (e.g. *hasbeak(X)* in this case). Not very helpful!



It you notice, the hypotheses that we would like to learn in this case are about concepts that are different from the observed predicate, namely we would like to explain why tweety has a beak in terms of other concepts, such as *bird* or *vulture*. The link between these new concepts and the given example is provided by the background knowledge. In the background knowledge, for instance, vulture are birds and birds have beaks, but what do we know about vulture? No much. Could we learn a concept *vulture* that would explain the given examples relative to the given background knowledge? In our example if we had a way of computing in Prolog the other negated consequences, we would be able to apply the same method described in the previous slides and generate a Bottom Clause that defines predicate not observed. For instance, we could learn the Bottom Clause vulture(X) that is a very general notion of vulture (namely "everything is a vulture"). And, if we had a more complex problem and richer dataset of positive and negative examples, we could may be learn a more refined rule for the notion of a vulture.

Intuitively the key problem is how to compute information that is missing in the given background knowledge and use these new notions to generate hypotheses. Computationally, how can we infer negated literals using Prolog? Muggleton has proposed an extension of the Progol system that incorporates the notion of "contrapositive" used in first-order theorem proving.

According to this notion a clause including n literals in the body is also equivalent to n different clauses generated by contrapositive in the following way. For instance**, p ←q, r** is equivalent to clauses (i) **¬q←¬p, r** and (ii) **¬r←¬p, q**. You can see that rules (i) and (ii) have negated literals in the head. We could then encode this in Prolog by inventing new names: for instance the first rule can be written in Prolog as (i) **non_q ← non_p, r**; and (ii) **non_r ← non_p, q**. Now the background knowledge includes in addition to the given clauses also their equivalent contrapositive (re-written) clauses.



Having seen in the previous slides how contrapositive allows us to infer, using SLD, from B and $\neg e^+$ negated predicates that are different from the example predicates (i.e. NOPL), we can now briefly consider the general idea of the algorithm that is behind Progol5.

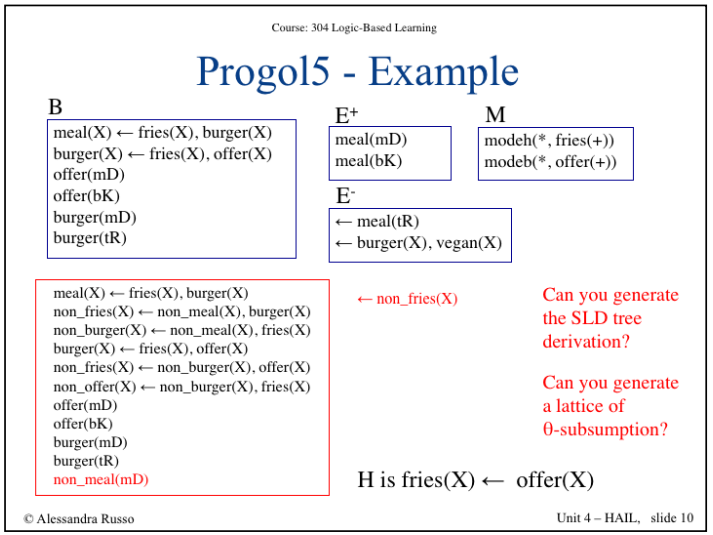
Since the assumption is that programs are definite clauses (and therefore monotonic), the algorithm uses a coverage loop approach, where at each iteration a positive example e^+ is picked from the current uncovered set E of examples. This chosen example is called **seed example**. An hypothesis h is then computed (as a single definite clause) as a solution that covers this seed example. Once this hypothesis is learned, it is added to the background knowledge and all other positive examples in E covered by this new hypothesis, together with the knowledge, are eliminated from the current set E of uncovered examples and the next iteration is executed. The process is repeated until E becomes empty.

Let's see in detail how an hypothesis for a given seed example is computed. This requires three steps.

1) The **StartSet** considers the given background knowledge, augments it with the contrapositives of each clause in B, adds to it the (skolemised version) of the body literals of the seed example and executes for each mode head declaration s in M an SLD query of its negated form. For each successful ground substitution, this step returns the atom $a = s\theta$. Each of these answers gives the head of an hypothesis.

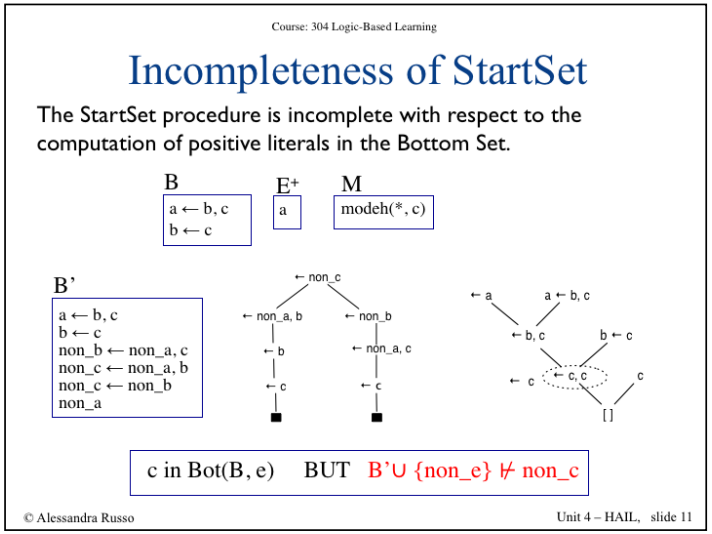
2) In **BottomSet**, the same program is used to derive now ground body atoms that are compatible with the given mode body declarations (i.e. b_i). The output of this step are part of the Bottom clause with head predicates computed in the StartSet. $a \leftarrow b_1, \ldots, b_n$

3) **Search** is where the negative examples are taken into consideration together with the positive examples. This step takes in input the initial background knowledge, the positive and negative examples and the learned bottom clause "lifted up" by replacing constants with variables. It returns in output a most compressed (i.e. clause with least number of literals) hypothesis H in M such that H subsumes the ground bottom clause and such that $B \cup E^+ \cup E^- \cup H$ is consistent. This latter case is computed by writing each negative example as a clause with a special constant symbol *f* in the head and failing to prove *f*.



We present here an example of the three steps of Progol5. The problem is related to fast food outlets, or bistros. The first rule states that to have a meal in an outlet X, it is sufficient to have a burger and fries there. The second rule states that a free burger comes with every order of fries at outlets with a special offer. Two of the facts state that mcDonalds and burgerKing are currently participating in such an offer and the other two facts state that burgers have been eaten at the mcDonalds and theRitz (for free). The positive examples state that a meal has been eaten at mcDonalds and at the burgerKing, whereas the negative examples state that not meal has been eaten at theRitz and that one cannot have burger and also have vegan. No skolemisation is needed in this case, since the positive examples are already ground and the negative examples are taken essentially as constraints. The negative examples will be transformed into the clauses *false* ← *meal(tR)* and *false* ← *burger(X), vegan(X)*. The background knowledge is augmented with the contrapositive rules and the first example is chosen as seed example, i.e. meal(mD). The mode declaration states that hypothesis should have the predicate *fries* in the head and occurrences of the predicate *offer* in the body. So the StartSet will start an SLD derivation of ←*non_fries(X)*, since we would like to derive (instances of this negated literal) in the negation of the Bottom set, so that when it is negated it becomes positive and be part of the bottom set and the bottom clause could have it as possible head predicate**. Can you construct the SLD derivation for this query?** This is done in class (Notes in CATE will include the answer).

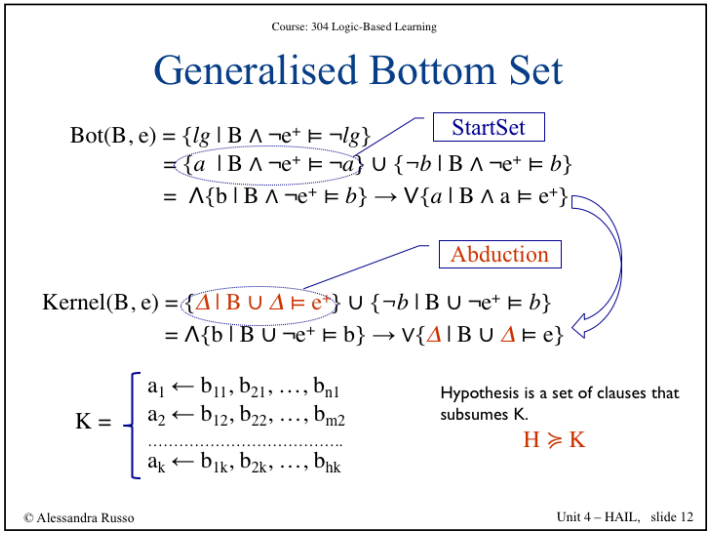
The following tree is the SLD proof tree for the given query, which returns a unification X/mD. So fries(mD) is a possible head for the bottom clause. Given this current unification and the fact that there is only one mode body, the BottomSet runs then the query ←*offer(mD)*. This trivially succeeds so the BottomSet returns the possible bottom set non_fries(mD) ∧ offer(mD), which when negated gives the bottom clause hypothesis $h_\perp$ = *fries(X)* ← *offer(X)*. Next to the proof tree is the lattice space given by M. The solution computed in this search is *fries(X)* ← *offer(X)*.



The use of contrapositive mechanism in StartSet can be seen as a technique for computing rules whose head is not defined as an example. This is because of the ability of proving deductively the negated head of the hypothesis rule from the background knowledge augmented with the contrapositive extension and the negated examples. But in practice this deductive mechanism of proving abductive answers for given examples is only applicable and works in limited cases, specifically when the **negated abducible needs to be used only once in the deductive proof**. This was demonstrated and proved in the paper "Hybrid Abductive Inductive Learning: a Generalisation of Progol", ILP 2003, O. Ray, K. Broda and A. Russo. An example is given here.

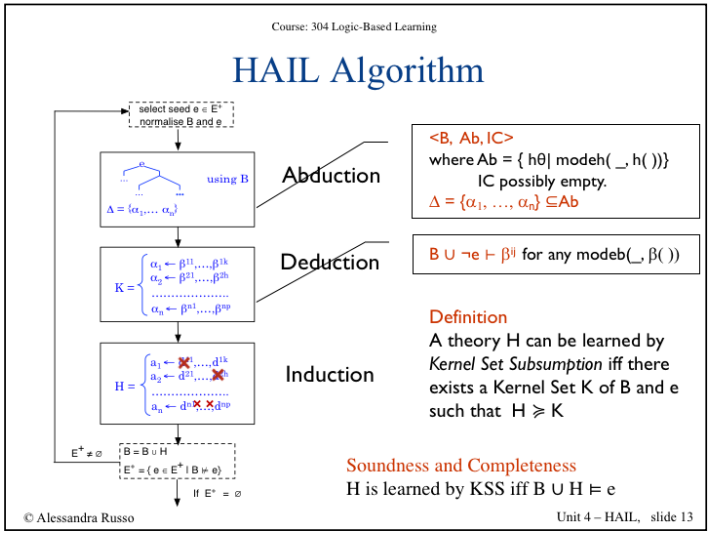
Let's consider the above simple learning task with background knowledge B, example E^+ and modeh given by the propositional variable *c*. [Note the * symbol in modeh means that c can occur any number of times in the solution.] By definition of Bot(B, a), B ∪{¬ a} ⊨ ¬c. This is equivalent to show that B ∪{¬ a} ∪{c} ⊨ ⊥, which is indeed the case, if you look at the resolution tree given on the right of the slide. You must notice though that in the resolution derivation the atom c needs to be used *twice*. But, what happens in the StartSet procedure? We extend B with contrapositives and apply SLD derivation once we add the negated example and the negated query ← c. We can easily see that the SLD derivation given in the middle of this slide fails. This means that the atom c is not derivable from the contrapositive technique of Progol 5. **Progol 5 is not complete.** StartSet procedure is therefore only complete with respect to derivations where the head predicate needs to be used only once in the proof. This example shows that the semantics of Bottom Generalisation is more general than what Progol5 can compute. So, the question is:
"**Is there a computational mechanism that is able to compute hypothesis derived by Bottom Generation**"?

We basically need a proper abductive process to compute the head of such hypothesis.



We have here rewritten the definition of BottomSet used in inverse entailment. We have seen that the BottomSet is the set of literals who negation can be derived from B∪ ¬e. We can divide this set into two parts: literals whose negated version is derived from B∪ ¬e (this is the first set of positive literals) and the set of negated literals whose positive version is derivable from B∪ ¬e (this is the case of the second set of ¬b literals). The first set is what the StartSet computes in Progol5 system and it is the part of the procedure that is incomplete as shown in the previous slide.

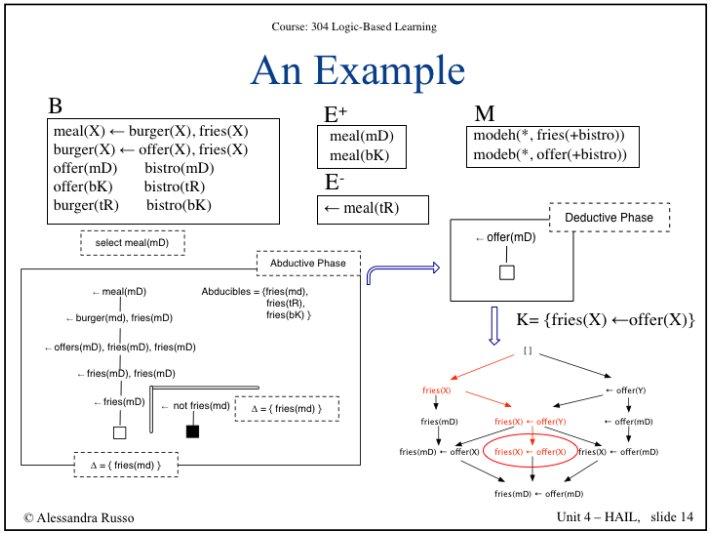
In our paper on "Hybrid Abductive Inductive Learning: A Generalisation of Progol" we have shown that the notion of Bot(B, e) can be generalised into a new notion called **Kernel Set**. The underlying idea is to replace the StartSet procedure with a full abductive reasoning procedure. Given a background knowledge and a seed example, the heads of a Kernel Set hypothesis can be computed directly using abduction instead of contrapositives, with the seed example as observation to explain by abductive derivation. Abduction is a complete procedure and will generate all alternative explanations for the seed example using B. The body literals of the hypothesis can still be generated as in Progol5, namely by deriving literals (specified in the given language bias) from B ∪ $\neg e^+$. These correspond to the two sets given in the definition of Kernel(B, e^+) shown in this slide. Note that
Kernel(B, e^+) is ground, but the rules in K are unground (according to the language bias). This is done again by replacing every constant in Kernel(B, e^+) with a new variable name whenever the mode declaration is expecting an input or output variable. The most specific hypothesis is then the unground set of definite clauses given in K, whose heads are (unground version of the) predicates that appear in the first set of Kernel(B, e^+) and body literals are unground version of the literals that appear in the second set Kernel(B, e^+).



Having introduced a notion of Kernel Set, we can say that an hypothesis H is **inferred (or derived) by Kernel Set Subsumption** (instead of by Bottom Generalisation) if and only if it is possible to compute a Kernel Set K, from a given background knowledge, seed example and mode declarations, and H is proved to subsume K. We have developed a system that computes this class of hypotheses. This algorithm shows for the first time a systematic way in which abduction can be integrated with induction in order to compute knowledge discovery. We refer to this approach as HAIL, for Hybrid Abductive Inductive Learning.

The algorithm reflects a similar structure as that of Progol5. It uses also a covering loop for which a seed example e from the given set of positive examples is selected at each iteration. Then an abductive proof procedure is run that takes B as background knowledge, seed example e as the goal or observation to explain, and the set of abducibles given by the set of ground instances of the mode head declarations that are given in the learning task. The abductive step produces a possible explanation of the example. This is a set of positive ground abducibles. In the second step, the background knowledge and negated seed example are used to compute ground instances of the mode body predicates according to the language bias. This computation is a pure deductive inference. At the end of the second step we will have generated a set of ground clauses that are very specific and that cover the positive examples. We will refer to this set as Kernel Set **Kernel(B, e)**. The unground version of this set of clauses forms the bottom element of the lattice that we will have to search for more compressed hypothesis. The computation, or search, for such an hypothesis that will subsume the Kernel Set, is the third step of the algorithm, referred here as induction, since it will search for more general hypothesis that will have the same examples coverage but will be more "compressed", i.e. more general.

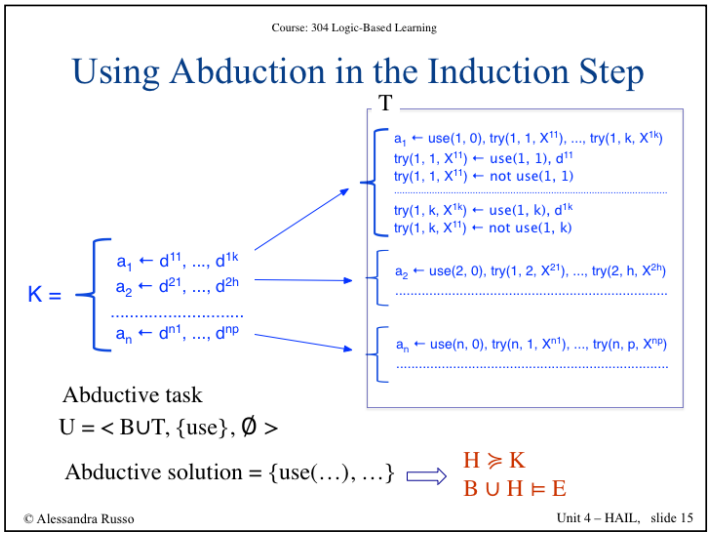
At the end of an iteration, once an hypothesis is computed, this is added to the background knowledge. All positive examples that are covered by this augmented (learned) knowledge will be removed from the set E^+. A new iteration is started if the set E^+ is still not empty.



Let's consider now the fast food example given in this slide. The background knowledge B describes a domain with three bistros: mcDonalds (mD), burgerKing (bK) and the Ritz (tR). To have a *meal* in a *bistro* it is sufficient to have a *burger* and some *fries* and free *burger* comes with every fries at bistros in a special *offer*. The positive examples state that a meal has been eaten at both the *mcDonalds* and at the *burgerKing*. The negative example states that a meal has not been eaten at the *Ritz*. The mode declarations state that atoms of the form *fries(X)* may appear in the head of an hypothesised clause and that atoms of the form *offer(X)* may appear in the body of the hypotheses.

It can be verified that an hypothesis can be derived by Bottom Generalisation, however the Progol5 system, using StartSet, is not capable of computing any hypothesis for this problem. In Tutorial 2 you are asked to demonstrate this.

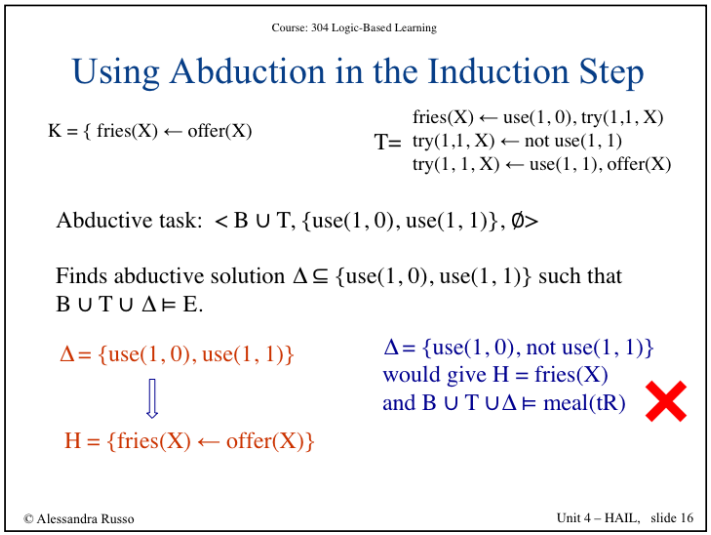
What this slide shows instead is that, using the HAIL algorithm it is possible to compute an inductive solution for this problem. The HAIL algorithm starts with picking a seed example. Let's assume we pick the first positive example *e = meal(mD)*. We then run an abductive proof procedure by setting the observation (goal) to be the seed example, abducibles to be the ground instances of mode head predicates (according to the type of the variables) and the background knowledge to be B itself. The abductive procedure returns in this case the explanation *fries(mD)*. In fact you can verify that adding this fact to B it is possible to derive *meal(mD)*. The second step of HAIL is the deductive phase. This consists of considering the grounding generated by the abductive procedure and prove if instances (with this grounding) of mode body literals can be derived by B ∪{¬e} (i.e. B ∧¬ meal(mD)). Given that we have only one modeb we check in this second phase if B ∧¬ meal(mD) ⊢ offer(md). If this is the case than a ground Kernel(B, e) has been computed, namely *fries(mD) ← offer(mD)*. It is again possible to show that B ∪ *fries(mD) ← offer(mD)* ⊢ *meal(mD)*. Finally the third step tries to compute a more general hypothesis, within the same language bias, that θ-subsumes this ground Kernel set. The lattice structure given in this slide reflects all the possible clauses within the language bias that are ordered according to θ-subsumption relation and that are bound by the empty clause and the Kernel set. The search now can starts from the top and traverse the lattice downwards whenever the current clause is not good enough, i.e. it covers some negative examples.



In the previous example we have briefly mentioned that once the Kernel set is computed a solution is an set of clauses that θ-subsumes the Kernel Set theory. The θ-subsumption lattice is bounded from the bottom by the Kernel set and from the top by the empty set of clauses. In the previous example, the most general and most compressed hypothesis is the unground version of the Kernel set itself. Any other clause would cover negative examples.

The computation of the most compressed hypothesis can also be performed using an abductive proof procedure. Computing the most compressed hypothesis that subsumes the kernel set means identify for each clause in the kernel set another clause with less number of literals that subsumes the chosen clause. Subsumption means finding some substitution that makes the subsuming clause a subset of the subsumed clause. So we could think of computing the subsuming clause by checking which literals from the subsumed clause can be removed and still preserve the correct coverage of the examples.

We have proposed a method for using abduction to compute the most compressed hypothesis, essentially to perform the inductive search! This is described in this slide. A theory T is constructed from the Kernel Set in the following way. For each clause in the Kernel Set, a special set of clauses is added to T in the following way. Use the same head predicate; then for each body literal d^{ij} add a *try(i, j, X^{ij})*, where X^{ij} includes the variables that appear in d^{ij}. Add also a *use(i, 0)*, to represent that the rule *i* is needed in the final hypothesis. For each *try(i, j, X^{ij})* two clauses are added *try(i, j, X^{ij})* ← *not use(i, j)* and *try(i, j, X^{ij})* ← *use(i, j), d^{ij}* where *use* is an *abducible* predicate. The abducible *use(i, j)* means that the j body literal is needed in clause i. The intuition is that, in order for a head atom a_i from *K* to contribute towards the satisfaction of an example *e* in *E*, each of the body atoms *try*(*i*, *j*, X^{ij}) must be satisfied. Thanks to the two rules added for this atom, its truth can be ensured in one of two ways: by simply assuming *not use*(*i*, *j*); or by abducing *use*(*i*, *j*) and proving d^{ij}. The former effectively ignores d^{ij} as if it were not there, while the latter solves d^{ij} as if it had been part of the clause. Similarly, the atom *use*(*i*,0) determines if the *i*th clause is included in the hypothesis, or not.



We can consider the example before and see how part of the induction step can be performed using the transformation described before and abductive inference. The Kernel Set K generated in Slide 14 is composed of only one clause, K = {fries(X) ← offer(X)}. The set of clauses T generated from the Kernel Set is given in the right hand side of this slide. In this case there are only three clauses, because the given clause in the kernel is only one and it has only one body literal.

Once this theory is created, an abductive task can be run using as background knowledge BUT, as abducibles all possible ground instances of the predicate use( ), integrity constraint are empty in this case and as goals the set of given examples. Note that the negative examples are expressed as negated literals in this case. An abductive answer to this task will be a set of ground instances of use( ). Instances of the form use(i, 0) in the abductive answer indicate that the clause i in the Kernel Set is needed. Instances of the form use(i, j) for some specific constants i and j indicate that the body literal j in the clause i of the kernel Set is needed. In our example here if we run an abductive inference on the given abductive task with E as goal, we would get as answer the abducibles {use(1, 0), use(1, 1)}, because in this case the full clause in the Kernel is needed and with its single body literal. If you develop the abductive proof yourself, you would notice that during the proof you could have constructed an abductive answer {use(1, 0), not use(1, 1)} which would have corresponded to the hypothesis *fries(X)*. This would have been good to explain the positive example, but it would also have covered the negative example, i.e. be able to fail to prove *not fries(tR)*, or equivalently be able to prove *fries(tR)* which is a negative example. In Tutorial 2 you are asked to expand the abductive proofs for the computation of hypothesis that subsume the Kernet Set and show that for Δ = {use (1,0), not use(1,1)} the negative example would be covered.

Course: 304 Logic-Based Learning

Abduction – Induction Cycle Revisited

Hybrid Abductive Inductive Learning is an approach that truly combines abduction and induction to support knowledge extraction.

- Abduction makes assumptions on relevant missing information.
- Deduction helps constructing the Kernel Set of clauses that bridge the background knowledge with the abductive explanations.
- Induction is the process of generalising: looking for alternative hypothesis that would still cover the examples, what we expect to be true or to be false.

Abduction could also be used to help searching for most compressed hypothesis in a subsumption lattice

The Hybrid Abductive and Inductive Learning approach demonstrates that this notion of abduction and induction cycle is indeed essential to discover knowledge that is not directly related to observed concepts.

Course: 304 Logic-Based Learning

Comparing approaches

HAIL ⩾ Alecto ⩾ Progol5 ⩾ Aleph ⩾ Progol

	Rules using Non-OP	Rules for Non-OP	Multiple clauses	Fully automated
Progol	✗	✗	✗	✓
Aleph	✓	✗	✗	✓
Progol5	✓	✓	✗	✓
Alecto	✓	✓	✓	✗
Hail	✓	✓	✓	✓

We have seen so far two main approaches for computing non observation predicate learning tasks. These are Progol5 and Hail. They all automate the computation of hypothesis whose predicates are not directly observed. The underlying general mechanism is similar: (i) Transform observed predicates (positive examples) into predicates that are not observed but declared as mode head predicates in the language bias, (ii) compute most specific clause that entails the predicate(s) computed in step 1 using the principle of inverse entailment, and then generalise the bottom clause by searching over the θ-subsumption lattice for more general clause that subsumes it. Step (ii) is computed in all cases by following the notion of inverse entailment. So they all fall within the context of inverse entailment semantics.

But they are, however, different in the set of hypothesis that they can actually compute, as they tackle step (i) differently. In Progol5 this approach has been further extended by using the notion of contrapositive transformation of the background knowledge. In this case rules defining non observed predicates can be learned. But the contrapositive mechanism can only generate one single ground atomic explanation per seed example, so only one single clause hypothesis can be learned from a given seed example. In the HAIL approach the use of a proper abductive proof procedure provides a fully automated mechanism for generating multiple ground atoms that explain a given seed example. The search over the lattice space can also be done simultaneously over the full Kernel Set and can generate multiple clause hypothesis.

Summary

- Defined
  - OPL and NOPL
  - Inverse Entailment as a mechanism for combining bottom-up/top-down search of hypothesis
  - Language Bias (mode declaration)
- Introduced Bottom Set Generalisation
- Presented Progol5 and its contrapositive mechanism for computing NOPL
- Shown incompleteness of Progol5 with respect to Bottom Set Generalisation.
- Described Hybrid Abductive Inductive Learning as a way for realising Pierce's abduction-induction cycle.

© Alessandra Russo