MC-TopLog: Complete Multi-clause Learning Guided by a Top Theory

Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad

Department of Computing, Imperial College London

Abstract. Within ILP much effort has been put into designing methods that are complete for hypothesis finding. However, it is not clear whether completeness is important in real-world applications. This paper uses a simplified version of grammar learning to show how a complete method can improve on the learning results of an incomplete method. Seeing the necessity of having a complete method for real-world applications, we introduce a method called $\top$-directed theory co-derivation, which is shown to be correct (ie. sound and complete). The proposed method has been implemented in the ILP system MC-TopLog and tested on grammar learning and the learning of game strategies. Compared to Progol5, an efficient but incomplete ILP system, MC-TopLog has higher predictive accuracies, especially when the background knowledge is severely incomplete.

1 Introduction

As first pointed out by Yamamoto [22], hypotheses derivable from Progol [11] are restricted to those which subsume E relative to B in Plotkin's sense [17]. This type of incompleteness can be characterised as deriving only single-clause hypotheses. In this paper, we compare entailment-incomplete single-clause learning systems to entailment-complete multi-clause learning systems.

Yamamoto uses the learning of odd-numbers to demonstrate Progol's incompleteness. His example involves recursion and mutually dependent predicates (odd and even), making it unclear whether only applications with these properties might be affected by this type of incompleteness. To the authors' knowledge it has not subsequently been demonstrated conclusively that the incompleteness of single-clause learning noticeably restricts the application of single-clause learners. It might reasonably be supposed that in real-world applications learned theories can always be built by sequentially adding single clauses.

Grammar learning is central to language translation software, automated booking systems and grammar checking for word processes. Section 2 uses a simplified version of grammar learning, which is artificially designed and does not involve recursion or mutually dependent predicates, to show how a complete method can improve the learning results of an incomplete method. This is further demonstrated in section 4 via experiments with two real-world data sets. More experiments with real-world applications can be found in [9], where target hypotheses are unknown for knowledge discovery tasks. The focus of this

S.H. Muggleton, A. Tamaddoni-Nezhad, F.A. Lisi (Eds.): ILP 2011, LNAI 7207, pp. 238–254, 2012.

paper is to introduce a new complete approach called ⊤-directed theory co-derivation(⊤DTcD). The following two subsections highlight the two key features that distinguish ⊤DTcD from other multi-clause learning methods.

1.1 Common Generalisation

The idea of common generalisation was first introduced in Plotkin's Least General Generalisation (LGG) [17] and Reynolds' Least Common Generalisation (LCG) [21]. This idea is used in this paper to extend ⊤DTD (⊤-directed theory derivation) to ⊤DTcD. We refer co-generalisation for methods that restrict their search spaces to common generalisations of multiple examples, as opposed to solo-generalisation for methods that generalise a single example.

Although doing co-generalisation can lead to higher efficiency, it has been introduced in few ILP systems. Among the systems based on Inverse Entailment (IE) [11], ProGolem [15] extending from Golem [13] is the only one that can do co-generalisation. Unfortunately, it suffers from similar type of incompleteness as that in Progol. While all the existing complete methods that are IE based can only do solo-generalisation, e.g. CF-Induction [5], XHAIL [20] and IMPARO [7]. These are reflected in table 1, which classifies several typical ILP systems based on their generalisation methods. Although CF-Induction and XHAIL can generalise multiple examples all at once, their search spaces are not bound to the common generalisations, therefore they are *not* in the category of co-generalisation.

The inability to do co-generalisation is less of an issue for ILP systems like HYPER [3] and TILDE [1], which use all the training examples to guide a top-down search. Also the generalisation methods in these systems are not IE based, thus they do not suffer from Progol's incompleteness. On the other hand, they lose the advantage provided by IE, that is, their search spaces are not bound to those hold for $B \wedge H \models E$. Also, these systems cannot handle abduction, thus not applicable to the grammar learning example given in this paper, where the background knowledge is incomplete.

1.2 Top Theory and TopLog Family

A top theory ⊤ is a logic program representing a declarative bias. Compared to the mode declaration [11] used in many ILP systems, a top theory has the

Table 1. Classifying ILP systems based on their generalisation methods

	Solo-Generalisation	**Co-Generalisation**
Single-clause	TopLog Progol FOIL	LGG and LCG Golem and ProGolem
Multi-clause	CF-Induction XHAIL IMPARO TAL MC-TopLog (⊤DTD)	 HYPER TILDE MC-TopLog (⊤DTcD)

advantage of encoding a strong declarative bias. Although there exists other forms of declarative bias that are comparable to the top theory in terms of their expressive power in encoding strong declarative bias, they are in the meta-level, such as antecedent description language (ADL) [4] and its extension $\mathcal{D}$LAB [18]. In contrast, a top theory is in the object-level as a logic program. This makes it possible for a top theory to be reasoned directly with background knowledge, so that the derived hypotheses are bound to those hold for $B \wedge H \models E$. In other words, a top theory not only provides a mechanism for naturally encoding the strong declarative bias, but also facilitate a method to bound the search space. A top theory is also similar to Spectre's [2] starting-point theory (an overly general theory to be unfolded), but the top theory make a clear distinction between terminal and non-terminal predicates. This is a powerful mechanism for distinguising between search control and the object language.

A top theory was first introduced in a method called $\top$-directed hypothesis derivation ($\top$DHD), which is implemented in the ILP system TopLog [14]. To overcome TopLog's limitation of single-clause learning, $\top$DHD is extended to $\top$DTD. The resulting system is named MC-TopLog (Multi-clause TopLog). $\top$DTD and $\top$DTcD correspond to two different learning modes in MC-TopLog: generalising single example or multiple examples. Inherited from $\top$DHD, both $\top$DTD and $\top$DTcD use a top theory to represent their declarative bias.

2 Multi-clause Learning vs. Single-clause Learning

Progol's entailment-incompleteness can be characterized by single-clause learning. Because a hypothesis H will not be derived by Progol, unless it subsumes an example e relative to B in Plotkin's sense. This condition requires H to be a single clause, and this clause is used only once in the *refutation* of the example e. This leads to our definition of single-clause and multi-clause learning as that in Definition 1. Please note that they are defined in terms of the number of hypothesised clauses used in a refutation of an example, rather than the number of clauses in H. Accordingly, even if the number of clauses in H is only one, it can still be multi-clause learning. For example, in Yamamoto's example of learning odd-numbers, the hypothesised clause $odd(s(X)) \leftarrow even(X)$ is used twice when proving the positive example $odd(s(s(s(0))))$, thus deriving such a hypothesis H from that example is multi-clause learning even though H appear to be a single clause. And vice versa: even if the number of clauses in H is more than one, it may be essentially single-clause learning. Such example will be given later.

Definition 1. *Let c_i be a clause, which is either from background knowledge B or hypothesis H. Suppose $R = \langle c_1, c_2, ..., c_n \rangle$ is a refutation sequence that explains a positive example e. Let M be the number of clauses in R that is from H. It is single-clause learning (SCL) if $M = 1$; while it is multi-clause learning (MCL) if $M \geq 1$.*

Positive and Negative Examples E:
e_1:$s([an, unknown, alien, hits, the, house], [])$.
e_2:$s([a, small, boy, walks, a, dog], [])$.
e_3:$s([a, dog, walks, into, the, house], [])$.
e_4:$\neg s([dog, hits, a, boy], [])$.

Hypothesis language $\mathcal{L}$:
Predicates $=\{s, np, vp, det, noun, verb...\}$
Variables $=\{S_1, S_2, S_3, ...\}$
Constants $=\{a, the, ...\}$

Background Knowledge B:
b_1:$np(S1, S2) \leftarrow det(S1, S3), noun(S3, S2)$.
b_2:$vp(S1, S2) \leftarrow verb(S1, S2)$.
b_3:$vp(S1, S2) \leftarrow verb(S1, S3), prep(S3, S2)$.
b_4:$det([a|S], S)$. b_5:$det([an|S], S)$. b_{13}:$det([the|S], S)$.
b_6:$noun([dog|S], S)$. b_7:$noun([boy|S], S)$.
b_8:$noun([house|S], S)$. b_9:$noun([alien|S], S)$.
b_{10}:$verb([hits|S], S)$. b_{11}:$adj([small|S], S)$.
b_{12}:$prep([into|S], S)$.

Part of Hypothesis Space $\mathcal{H}$:
h_1:$s(S1, S2) \leftarrow det(S1, S3), S3 = [Word|S4], noun(S4, S5), vp(S5, S6), np(S6, S2)$.
h_2:$s(S1, S2) \leftarrow det(S1, S3), adj(S3, S4), noun(S4, S5), S5 = [Word|S6], np(S6, S2)$.
h_3:$s(S1, S2) \leftarrow np(S1, S3), S3 = [Word|S4], prep(S4, S5), np(S5, S2)$.
h_4:$s(S1, S2) \leftarrow np(S1, S3), vp(S3, S4), np(S4, S2)$.
h_5:$np(S1, S2) \leftarrow det(S1, S3), adj(S3, S4), noun(S4, S2)$
h_9:$np(S1, S2) \leftarrow det(S1, S3), prep(S3, S4), noun(S4, S2)$
h_6:$verb([walks|S], S)$. h_7:$adj([unknown|S], S)$. h_8:$prep([unknown|S], S)$.

Fig. 1. Grammar Learning Example

2.1 Grammar Learning Example

Fig. 1 shows a simplified version of grammar learning, which is used here to exemplify Definition 1. In this grammar learning task, multi-clause and single-clause learning methods will derive $H_{mc} = \{h_4, h_5, h_6, h_7\}$ and $H_{sc} = \{h_1, h_2, h_3\}$, respectively. Although there are multiple clauses in H_{sc}, each of them is derived independently from different examples by a single-clause learner. Specifically, h_1, h_2 and h_3 are generalised independently from e_1, e_2 and e_3, respectively. In contrast, clauses in H_{mc} are dependent, and they have to be generalised together in order to explain an example. For instance, hypothesising h_4 alone is not able to complete the refutation proof of the example e_1, since the definition about np is incomplete in B and the type of the word 'unknown' is also missing from B. Thus another two clauses, either $\{h_5, h_7\}$ or $\{h_8, h_9\}$, have to be derived together with h_4 in order to explain e_1.

In this artificially designed example, H_{mc} is the target hypothesis which is not derivable by a single-clause learner. H_{mc} is also more compressive than H_{sc}, because H_{mc} has a shorter description length[1]than H_{sc} while covers the same number of examples. The shorter description length of H_{mc} results from a multi-clause learner's ability to hypothesise multiple dependent clauses. For example, h_4 is simpler than any of h_1, h_2 and h_3, because it is derived together with other clauses, such as $\{h_5, h_7\}$ or $\{h_8, h_9\}$.

2.2 Distinctions from MPL and LMC

As discussed earlier, clauses within H_{mc} are dependent. This is similar to that in multiple predicate learning (MPL), where clauses about different predicates

[1] In this paper, the description length (DL) of a clause is defined by the number of literals in the clause; while the compression is defined as $p - n - DL$ where p and n are the number of positive and negative examples covered by the clause.

depend on each other. However, the MPL discussed in [19] is essentially single-clause learning. Because each predicate to be learned are observable and provided as examples. Therefore there is only one clause about the observed predicate to be hypothesised for each example. If applying an MPL method to the learning problem in Fig. 1, it would require the predicates *np* and *vp* to be observable and provided as training examples.

The term *learning multiple clauses* (LMC) is used to describe a global-optimisation approach, in which multiple clauses that compressed from the *whole* set of examples are refined together, as opposed to a local-optimisation approach like the covering algorithm, where clauses compressed from a *subset* of examples are added to the final H iteratively. However, LMC and MCL are related to different issues. LMC is related to the issue of selecting hypotheses globally rather than locally. The hypotheses from which it selects can be derived either by MCL or SCL. Even if a learning algorithm's search space only consists of single clauses derived by SCL, its final hypothesis may still have multiple clauses, which are aggregated from single clauses generalised from different examples. In contrast, MCL is to do with generalising an example to multiple clauses instead of a single clause. It can be combined with a selection method that is either global or local.

2.3 Increase in Hypothesis Space

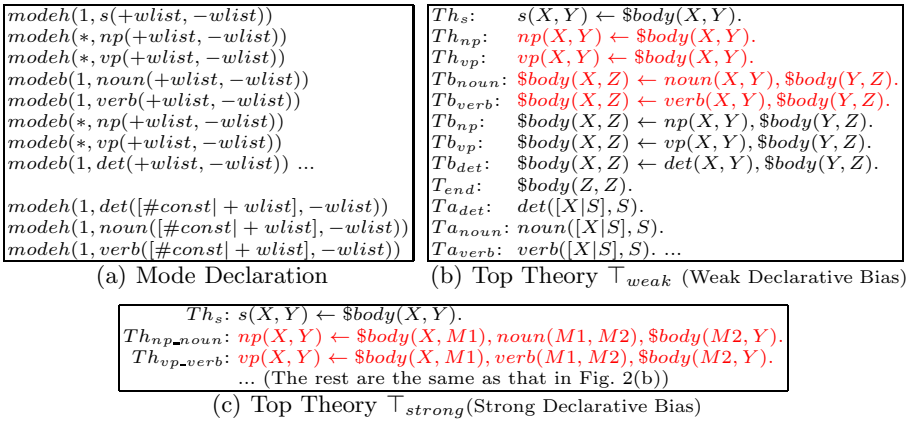
Although the complete hypothesis space of MCL makes it possible to find hypotheses with higher compression than SCL, this comes at the cost of a much larger search space. Specifically, the upper bound on the hypothesis space of a single-clause learner is $O(2^N)$, where N is the number of distinct atoms derivable from a hypothesis language. In contrast, it is $O(2^{2^N})$ for a multi-clause learner, because it does not ignore the hypotheses with dependent clauses. That is why it is particularly important for MCL to bound its search space to the candidates hold for $B \wedge H \models E$ and makes use of the strong declarative bias that is available to further constrain the hypothesis space.

3 MC-TopLog

This section introduces top theories first, and then explains how to derive a hypothesis using a top theory. Finally, we explain how to constrain the search space to common generalisations using the ⊤DTcD algorithm.

3.1 Top Theories as Declarative Bias

A top theory $\top$ is a declarative bias in the form of a logic program. As a context-free grammar, a top theory consists of the terminals and non-terminals. The terminal literals are those in the hypothesis language, such as $s(X, Y)$ in Fig. 2(b); while the non-terminal literals like $\$body(X, Y)$ in Fig. 2(b) are not allowed to appear in neither the hypothesis language nor background knowledge. In order to distinguish the non-terminals, they are prefixed with the symbol '$'.



(a) Mode Declaration (b) Top Theory $\top_{weak}$ (Weak Declarative Bias)

(c) Top Theory $\top_{strong}$(Strong Declarative Bias)

Fig. 2. Declarative Bias of Grammar Learning

Although the non-terminals do not appear in the hypothesis language, they play important role in composing the hypothesis language. More examples of various non-terminals can be found in [8].

Composing Hypothesis Language. There are two operators for composing hypothesis language from a top theory: SLD-resolution and substitution. By applying SLD-resolution to resolve all the non-terminals in an SLD-derivation sequence, a hypothesis clause with only terminals can be derived. For example, a hypothesis clause $s(S1, S2) \leftarrow np(S1, S3), vp(S3, S4), np(S4, S2)$ can be derived from an SLD-derivation sequence $[Th_s, Tb_{np}, Tb_{vp}, Tb_{np}, T_{end}]$. Different from SLD-resolution, which is to do with connecting terminal literals, substitution is required to deal with ground values in the terminal literals. For example, abductive hypotheses are ground facts, while their corresponding top theories are universally quantified, e.g. $noun([X|S], S)$ in Fig. 2(b). All the hypotheses derived from $\top$ by applying SLD-resolution or substitution hold for $\top \models H$. In this paper, *translation* refers to the process of deriving H from $\top$, and $\top$ *version of* H refers to the set of clauses in $\top$ that derive H.

Strong Declarative Bias. Fig. 2(a) shows a mode declaration, whose corresponding version of a top theory is in Fig. 2(b). This kind of declarative bias only tells what predicates are allowed in the head/body of a hypothesis clause. However, a stronger declarative bias may exist for a learning task. In that case, it is definitely worth to use that information to further constrain the hypothesis space. For example, in the grammar learning task, we know a noun phrase always consists of a noun and a verb phrase always has a verb. This provides information about how predicates should be connected. However, there is no way for a mode declaration to capture this information, while a top theory can encode it as that in Fig. 2(c). Such a top theory will avoid deriving clauses like $np(S1, S3) \leftarrow det(S1, S2), adj(S2, S3)$, which defines a noun phrase without

a noun. Another example of strong bias exists for learning tasks whose target hypothesis is known to be recursive. In that case, it would be more efficient if non-recursive clauses are excluded from the hypothesis space. Apart from the strong bias about the connection of predicates, there are other strong biases, such as the restriction on function terms. For example, in Yamamoto's example of learning odd-numbers, it would be undesirable to have a clause like $odd(s(X)) \leftarrow even(s(s(X)))$ in the hypothesis space, since it will lead to the expansion of function terms during reasoning.

3.2 $\top$-Directed Theory Derivation ($\top$DTD)

$\top$DTD is to derive all the candidate hypotheses that satisfy (1), where $\vdash_h$ denotes a derivation in at most h resolutions. $\top$DTD uses the top theory to direct the search for such hypotheses. Specifically, it finds all the refutations of e that satisfy (2), where $\vdash_{h'}$ has the same semantic as $\vdash_h$ except $h' \geq h$[2]. It is the use of $\top$ that makes refutations of e derivable, otherwise, e cannot be proved by B alone, because of the missing clauses to be hypothesised. After deriving all the refutations of e, each refutation sequence R_i is processed to derive the corresponding H_i. This process includes the following two main steps. (a) Extracting derivation sequences D_i from each refutation sequence R_i. Each extracted sequence in D_i preserves the same order as that in R_i. This guarantees that the pair of literals resolved in D_i is the same as that in R_i. To facilitate the extraction, it requires R_i to be recorded as a list with nested sub-lists, instead of a linear sequence. To facilitate the extraction, it requires R_i to be recorded as a list with nested sub-lists, instead of a linear sequence. More details about how to extract D_i from the R_i can be found in [8]. (b) Translating D_i into H_i, which are explained in Section 3.1. In the case that ground values are required, the values to be substituted come from the unification that happens when refuting e using $\top$ and B. Therefore it requires R_i to record the ground values unified during the refutation. The full description of $\top$DTD algorithm and its corresponding cover set algorithm are given in Algorithm 1 and 2, respectively. The correctness (ie. soundness and completeness) of $\top$DTD is proved in Theorem 1. An example of how $\top$DTD works is given in Example 1.

$$B \wedge H \ \vdash_h e \ (e \in E^+) \tag{1}$$

$$B \wedge \top \vdash_{h'} e \ (e \in E^+, h' \geq h) \tag{2}$$

$$\top \models H \tag{3}$$

$\top$DTD resembles Explanation-based Generalisation (EBG)[6] in that both algorithms find all possible explanations for the seed example first and then construct generalisations based on the derived explanations. However, EBG is essentially deductive learning, while $\top$DTD can achieve inductive learning. Specifically, EBG derives its generalisations from background knowledge, while $\top$DTD's generalisations are derived from a top theory, which can compose hypothesis language that do not exist in the background knowledge.

[2] Because apart from the terminals in (1), (2) also have non-terminals to be resolved.

Algorithm 1. $\top$-directed Theory Deriavation ($\top$DTD)

Input: a positive example e, background knowledge B, top theory $\top$ and h'
Output: $\mathcal{H} = \{H_i : B \wedge H_i \vdash_h e\}$, where $h \leq h'$
1: Let $\mathcal{H} = \emptyset$
2: $\mathcal{R} = \{R_i : R_i = Refs(e, B, \top, h)\}$ %Find all the refutations of e that satisfy the formula 2
3: **for all** R_i in $\mathcal{R}$ **do**
4: $D_i = DSeqs(R_i)$ %Obtain derivation sequences D_i by extracting $\top$ clauses from R_i.
5: $H_i = Trans(D_i)$ %Translate D_i into a hypothesis theory H_i
6: $\mathcal{H} = \mathcal{H} \cup H_i$
7: **end for**
8: **return** $\mathcal{H}$

Algorithm 2. Cover set algorithm of $\top$DTD

Input: examples E, background knowledge B, top theory $\top$ and h'
Output: a hypothesis H
1: Let $H = \emptyset$ and E^+ = all positive examples in E
2: **for all** $e_i \in E^+$ **do**
3: $\mathcal{H}_i = TDTD(e_i, B, \top, h')$
4: $\mathcal{H} = \mathcal{H} \cup \mathcal{H}_i$
5: **end for**
6: **while** $E^+ \neq \emptyset$ **do**
7: Let H_1 be the one in $\mathcal{H}$ with highest compression and $H = H \cup H_1$
8: Let E' be the positive examples covered by H_1 and $E^+ = E^+ - E'$
9: Let $\mathcal{H}'$ be the ones in $\mathcal{H}$ that only cover none of E^+ and $\mathcal{H} = \mathcal{H} - \mathcal{H}'$
10: **end while**
11: **return** H

Theorem 1. Correctness of $\top$DTD *Given e, B, $\top$ and h', Algorithm 1 returns all candidate hypotheses that satisfy (1), where H is within the hypothesis space defined by $\top$.*

Sketch Proof. *Assume the theorem is false. Then either (a) the algorithm does not terminate or (b) a theory H derived by the algorithm does not satisfy (1) or (c) the algorithm cannot derive a theory H that is within the hypothesis space defined by $\top$ and satisfies (1).*

First consider (a). Due to the restriction of at most h' resolutions in formula(2), $\mathcal{R}$ derived at step 2 is a finite set. Therefore there are only finite number of loops between step 3 and 7. Also each operation within the loop terminates in finite time. This refutes (a).

Secondly suppose (b) is true, which means $B \wedge H \wedge \neg e \not\models \square$. But at step 2, a refutation R_i that satisfies (2) can be found, which means clauses appearing in R_i form pairs of complementary literals. Following step 4, derivation sequences D_i can be extracted from the refutation sequence R_i. Then at step 5, there are three possible ways to translate D_i into H: (1) only SLD-resolution (2) only substitution; (3) both SLD-resolution and substitution. In case (1), all the non-terminals are resolved using SLD-resolution in order to compose hypothesis clauses with only terminals. The resolved literals must be in pairs, otherwise there will be at least one literal left unresolved, which means there will be non-terminals remaining in the derived H. If replacing the $\top$ clauses in R_i with their corresponding H,

whose only difference from the replaced $\top$ *clauses are pairs of non-terminals, then the clauses in this new sequence still form pairs of complementary literals. Therefore it contradicts the assumption that* $B \wedge H \wedge \neg e \not\models \square$. *In case (2), if replacing the* $\top$ *clauses with* H, *which is derived by substituting the variables in* $\top$ *with the ground values unified during the refutation, then the clauses in this new sequence still form pairs of complementary literals. Thus it also contradicts the assumption. In case (3), the assumption is also contradicted considering both case (1) and (2).*

Lastly consider (c), which implies that the corresponding $\top$ *version of* H *from which it is translated cannot be used to prove* e *with* B, *that is, the step 2 cannot be executed. However, considering that* H *is translatable from* $\top$, *that is, within the hypothesis space defined by* $\top$, *the formula (3) holds. Then (4) holds and (2) can be derived accordingly. This means a refutation using* B *and the* $\top$ *version of* H *does exist for* e. *This contradicts the assumption and completes the proof.*

$$B \wedge \top \models B \wedge H \tag{4}$$

Example 1. For the learning task in Fig. 1, one of the refutations for e_1 is as shown in Fig. 3. Its corresponding SLD-refutation sequence is recorded as $R_1 = [\neg e_1, [Th_s, Tb_{np}, [Th_{np\_noun}, Tb_{det}, b_5, Tb_{prep}, [Ta_{prep}(unknown)], T_{end}, b_9, T_{end}], Tb_{vp}, b_2, b_{10}, Tb_{np}, b_1, b_{13}, b_8, T_{end}]]$. Using the extraction algorithm explained in [8], D_1 consisting of three derivation sequences can be extracted from R_1. They are: $d_1 = [Th_s, Tb_{np}, Tb_{vp}, Tb_{np}, T_{end}]$, $d_2 = [Th_{np\_noun}, Tb_{det}, Tb_{prep}, T_{end}, T_{end}]$ and $d_3 = [Ta_{prep}(unknown)]$, which are highlighted by the three square boxes in Fig. 1. Then by applying SLD-derivation and substitution to D_1, $T_1 = \{h_4, h_8, h_9\}$ can be derived, where h_i is in Fig. 1.

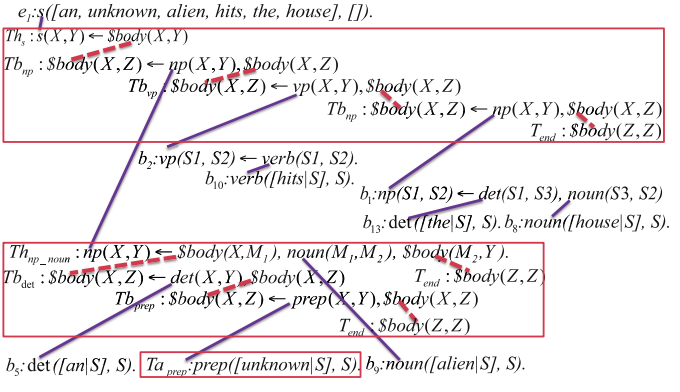
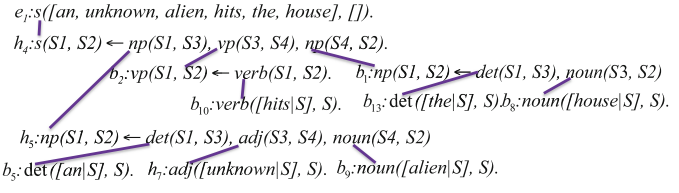


Fig. 3. Refutation of e_1 using clauses in B and $\top_{strong}$(Fig. 2(c)). The dash lines represent resolving a pair of non-terminal literals, while the solid lines correspond to the terminals.

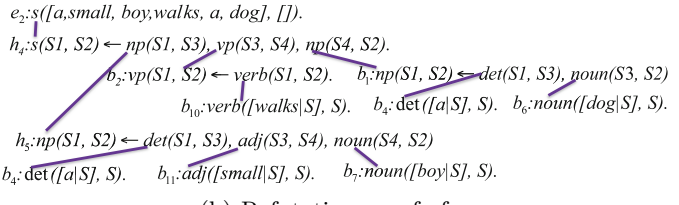
3.3 ⊤-Directed Theory Co-Derivation (⊤DTcD)

In order to constrain the derivable hypotheses to common generalisations, ⊤DTcD extends ⊤DTD based on co-refutation. Co-refutation combines the refutations that are the same except the instantiation of variables. Co-refutation is feasible via program transformation. Specifically, literals of the same predicate can be combined into one literal by combining their corresponding arguments into a compound. For example, the refutation proof in Fig 4(c) is the result of combining the two refutation proofs in Fig 4(a) and Fig 4(b). Co-refutation has the advantage of proving several examples together. More importantly, it proves them using the same non-ground clauses.

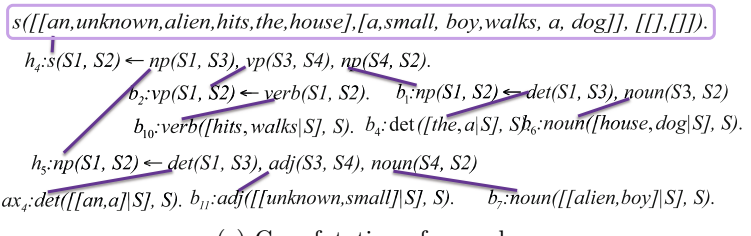
The design of ⊤DTcD is based on the fact that if a theory is common to multiple examples E, then the refutation proofs of each example in E using that common theory will have the same structure, that is, the proofs are the same except the instantiation of variables. Those same-structure refutation proofs can be combined into co-refutation by combining corresponding arguments. It is the combined proof that forces the co-generalised examples to be proved using the same non-ground rules. The next question is how to choose the examples to be generalised together. Rather than randomly sample a pair of examples as



(a) Refutation-proof of e_1



(b) Refutation-proof of e_2



(c) Co-refutation of e_1 and e_2

Fig. 4. Combine same structure refutation-proofs

that in ProGolem, ⊤DTcD takes all positive examples as input, while those do not fit are filtered out along the derivation of a refutation proof. At the end of a refutation, not only a hypothesis is derived, but also the maximum set of examples that can be explained by that hypothesis.

The algorithm of ⊤DTcD is given in Algorithm 3. It is the same as Algorithm 1 except (1) its input and output; (2) its step 2 and 3, where it combines examples to be generalised together into a compound and queries the compound instead of a single example. The cover set algorithm of ⊤DTcD is also slightly different from that of ⊤DTD, since the output of ⊤DTcD contains the candidate hypotheses for all the positive examples, rather than just one example. Specifically, the steps 2-5 in Algorithm 2 are replaced with a single step 2 in Algorithm 4. The correctness of ⊤DTcD is given in Theorem 2. We also give an example of how ⊤DTcD works in Example 2.

Although ⊤DTcD requires its co-generalised examples to have the same structure of refutation proofs, it is still applicable to learning recursive theories. Because the refutations using a recursive theory have at least one recursive step in common, even though the lengths of refutations may vary because of applying the recursive theory different times. Details have to be omitted here due to the limited space, but it is demonstrated in the experiments of learning game strategies that it is feasible to apply ⊤DTcD for learning recursive theories.

$$B \wedge H \ \vdash_h E_i \tag{5}$$

$$B \wedge \top \ \vdash_{h'} E_i \tag{6}$$

$$where\ h' \geq h \wedge (E_i \subset E^+ \wedge |E_i| > 1) \wedge (\forall e_j \in E_i, sameRefStru(e_j)) \tag{7}$$

Algorithm 3. ⊤-directed Theory co-Deriavation (⊤DTcD)

Input: All positive examples E^+ , background knowledge B, top theory $\top$ and h'
Output: $\mathcal{H} = \{H_i : B \wedge H_i \vdash_h E_i\}$, where $E_i \subset E^+$, $|E_i| > 1$ and $h \leq h'$
1: Let $\mathcal{H} = \emptyset$
2: $e_{comp} = Aggr(E^+)$ %Aggregate all positive examples E^+ into a compound example e_{comp}
3: $\mathcal{R} = \{R_i : R_i = Refs(e_{comp}, B, \top, h)\}$ %Find all the refutations that satisfy the formula 6
4: **for all** R_i in $\mathcal{R}$ **do**
5: $D_i = DSeqs(R_i)$ %Obtain derivation sequences D_i by extracting $\top$ clauses from R_i.
6: $H_i = Trans(D_i)$ %Translate D_i into a hypothesis theory H_i
7: $\mathcal{H} = \mathcal{H} \cup H_i$
8: **end for**
9: **return** $\mathcal{H}$

Algorithm 4. Cover set algorithm of ⊤DTcD

Input: examples E, background knowledge B, top theory $\top$ and h'
Output: a hypothesis H
1: Let $H = \emptyset$ and E^+ = all positive examples in E
2: $\mathcal{H} = TDTcD(E^+, B, \top, h')$
3: **while** $E^+ \neq \emptyset$ **do**
4: Let H_1 be the one in $\mathcal{H}$ with highest compression and $H = H \cup H_1$
5: Let E' be the positive examples covered by H_1 and $E^+ = E^+ - E'$
6: Let $\mathcal{H}'$ be the ones in $\mathcal{H}$ that only cover none of E^+ and $\mathcal{H} = \mathcal{H} - \mathcal{H}'$
7: **end while**
8: **return** H

Theorem 2. Correctness of ⊤DTcD *Given E^+, B, $\top$ and h', Algorithm 3 returns all candidate hypotheses that hold for (5), where (1) H is within the hypothesis space defined by $\top$; (2) $E_i \subset E^+, |E_i| > 1$ and each $e_j \in E_i$ shares the same structure of refutation proofs.*

Sketch Proof. *Assume the theorem is false. Then either (a) the algorithm does not terminate or (b) a theory H is derived by the algorithm as a co-generalisation of E_i, while $\exists e_j \in E_i, B \wedge H \not\models e_j$. or (c) the algorithm cannot derive a theory H that is within the hypothesis space defined by $\top$ and satisfies (5).*

First consider (a). Similar to that in the proof of Theorem 1, case (a) is refuted because: (1) the bound h' on the resolution steps guarantees that $\mathcal{R}$ is a finite set; (2) each operation within the for-loop terminates in finite time.

Secondly suppose (b) is true, but at step 3, a co-refutation of E_i using B and $\top$ can be found, which means $\forall e_j \in E_i, B \wedge \top \vdash_{h'} e_j$. Considering that the rest of the algorithm is the same as that in Algorithm1 and the correctness of Algorithm1 which is proved in Theorem 1, the hypothesis H derived will satisfy $\forall e_j \in E_i, B \wedge H \vdash_h e_j$, which contradicts the assumption and refutes (b).

Lastly consider (c), which implies that the step 3 cannot be executed either because (1) the corresponding $\top$ version of H from which it is translated cannot be used to prove E_i with B; or (2) the refutation of each e_j in E_i cannot be combined into a co-refutation. For case (1), similar to that in the proof of Theorem 1, (6) can be derived from the formulae (3) and (5). This means refutation using B and the $\top$ version of H does exist for the set of examples E_i that share the same structure of refutation proofs. The case (2) contradicts the fact that each $e_j \in E_i$ shares the same structure of refutation proofs so that their refutations can be combined, therefore completes the proof.

Example 2. For all the positive examples in Fig. 1, the ⊤DTcD method first combines them into a compound example as $s([[an, unknown, alien, hits, the, house], [a, small, boy, walks, a, dog], [a, dog, walks, into, the, house]], [[], [], []])$, and then proves it using clauses in B and $\top$. In this way, we can derive the hypothesis $H_2 = \{h_4, h_5, h_7\}$ that co-generalises examples e_1 and e_2. Please note that H_2 does not cover e_3, since e_3 is filtered out in the refutation using the $\top$ version of H_2. As visualised in Fig. 5, e_3 would be filtered out at the goal marked with a cross symbol, because the word 'dog' in e_3 is known to be a noun, rather than an adjective, thus it has to be filtered out in order to succeed the other part of the compound goal. Here we also give an example of the hypotheses that are

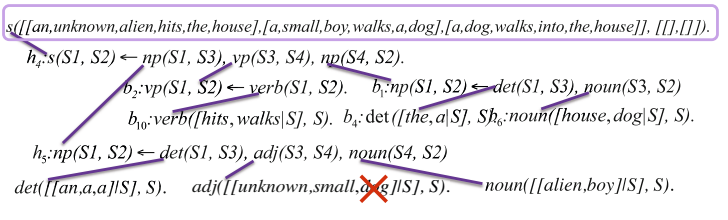


Fig. 5. Filter

pruned due to non-common generalisations: the hypothesis $H_1 = \{h_4, h_8, h_9\}$ derived when generalising e_1 alone is no longer derivable because apart from e_1 it cannot generalise either e_2 or e_3. Specifically, both e_2 and e_3 have their second words known as non-prepositions according to the given background knowledge, therefore they do not fit into the co-refutation using the $\top$ version of H_1.

4 Experiments

The null hypotheses to be empirically investigated in the study are as follows. (a) A multi-clause learning method does not have higher predictive accuracies than a single-clause learning method. MC-TopLog and Progol5 [12] are the two ILP systems used in this experiment. (b) The search space of a co-generalisation method ($\top$DTcD) is not smaller than that of a solo-generalisation method ($\top$DTD). All used materials can be found at http://ilp.doc.ic.ac.uk/mcTopLog.

4.1 Grammar Learning

Materials. The complete theory for parsing a grammar is in Fig. 6. The background knowledge B for each learning task is generated by randomly removing certain number of clauses from the complete theory, and those left-out clauses form the corresponding target hypothesis. Part of the training examples are in Fig. 7. There are 50 training examples and half of them are negative. Therefore the default accuracy is 50%.

```
s(S1,S2) :- np(S1,S3), vp(S3,S4), np(S4,S2).
s(S1,S2) :- np(S1,S3), vp(S3,S4), np(S4,S5), prep(S5,S6), np(S6,S2).
np(S1,S2) :- det(S1,S3), noun(S3,S2).
np(S1,S2) :- det(S1,S3), adj(S3,S4), noun(S4,S2).
vp(S1,S2) :- verb(S1,S2).
vp(S1,S2) :- verb(S1,S3), prep(S3,S2).
det([a|S],S).      det([the|S],S).
adj([big|S],S).    adj([small|S],S).    adj([nasty|S],S).
noun([man|S],S).   noun([dog|S],S).     noun([house|S],S).    noun([ball|S],S).
verb([takes|S],S).  verb([walks|S],S).   verb([hits|S],S).
prep([at|S],S).    prep([to|S],S).      prep([on|S],S).       prep([in|S],S).     prep([into|S],S).
```

Fig. 6. A Complete Theory for Parsing a Grammar

```
s([the,dog,takes,the,ball,to,the,house],[]).      ¬s([the, dog],[]).
s([the,small,dog,walks,on,the,house],[]).          ¬s([dog,the,man,the,walks],[]).
s([a,ball,hits,the,dog],[]).                       ¬s([ball,a,dog,a,hits],[]).
```

Fig. 7. Part of the Training Examples for Grammar Learning

Methods. The null hypothesis(a) was investigated by comparing the learning results of MC-TopLog and Progol5[12] for randomly chosen samples. For each size of leave-out, we sampled ten times and the predictive accuracies results of ten samples were averaged. The predictive accuracies were measured by leave-one-out cross validation. The null hypothesis(b) was examined by comparing

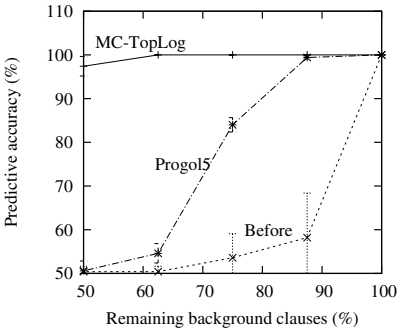


Fig. 8. Predictive Accuracies (Grammar)

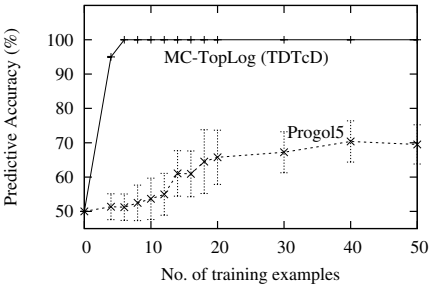


Fig. 9. Predictive Accuracies (Nim)

the search spaces and running time of ⊤DTD and ⊤DTcD. The search space is measured by the number of candidate hypotheses generated during learning.

Results. The predictive accuracies are given in Fig. 8, where the x-axis denotes the percentage of clauses remaining in the background knowledge. The smaller the percentage, the more clauses are left-out and to be learned. The label 'before' means before learning, and its accuracy line shows the degree of incompleteness in the background knowledge. Progol's accuracy line is above the 'before learning' line, which shows the effectiveness in learning. However, when the percentage of remaining clauses decreases to half, Progol5 fails to reconstruct the multiple missing clauses due to its single-clause limitation, therefore its accuracy drops to default. In contrast, MC-TopLog's ability of deriving multi-clause hypotheses makes it possible to hypothesise the missing clauses or their approximations even when half of the complete theory is left-out. Therefore MC-TopLog's predictive accuracies are always higher than that of Progol5 in this experiment, and their difference increases as the background knowledge becomes more incomplete. Thus the null hypothesis (a) is refuted. The accuracy line of MC-TopLog actually has two lines overlapped, thus for this dataset there is no significant difference between ⊤DTD and ⊤DTcD in terms of accuracies. Fig. 11 shows that the search space is reduced dramatically when the learning method switches from ⊤DTD to ⊤DTcD, thus the null hypothesis (b) is refuted. The running time compared in Fig. 12 shows similar pattern to that in Fig. 11, which further confirms the improvement of ⊤DTcD over ⊤DTD in terms of efficiency.

4.2 Learning Game Strategies

Materials. We choose the game Nim [16] for this experiment, because the target hypothesis not only has recursion, but also involves non-observable predicate learning. The learning task is to generalise a theory for identifying a P-position, which is a position that players are guaranteed to win if continue to play optimally, that is, identifying the precondition for grasping the winning strategy. Although [16] has suggested a single-clause hypothesis as $play(HeapN_1, HeapN_2,$

$HeapN_3) \leftarrow xor(HeapN_1, HeapN_2, HeapN_3)$, this is not the target hypothesis unless the number of heaps N is fixed to be three. To handle a more general case where N is not fixed, that hypothesis is too specific and needs to be further generalised. The background knowledge available for this learning task includes the definition of mathematical functions like *and*, *or* and *xor*. The training examples are in the form of $play([3,4,5])$, in which the number sequence records the number of sticks in each heap.

Methods. Similar to the experiment of grammar learning, the null hypothesis(a) was investigated by comparing the learning results of MC-TopLog and Progol5. However, different from the previous experiment, the background knowledge is fixed, since its size is too small to be randomly sampled. The accuracy curves in Fig. 9 are drawn with the number of examples on the x-axis. The null hypothesis(b) was examined by comparing the search spaces and running time of ⊤DTD and ⊤DTcD. Again, we varied the number of examples to see how the search space shrinks with more examples available to be co-generalised.

Results. As shown in Fig. 9, MC-TopLog only needs 6 examples to achieve accuracy of 100%, while Progol5 is not able to achieve accuracy of 100% even given 50 examples. Therefore the null hypothesis (a) is refuted. Progol's significantly lower accuracies results from its single-clause hypotheses which are too specific. For example, $\forall c_i \in H_s, H_m \models c_i$, where H_s and H_m are in Fig. 10(a) and 10(b), respectively. H_m not only consists of a recursive clause, but also involves a non-observable predicate *compute*, therefore even methods that can learn recursive theories (e.g. [10]) are not able to derive H_m.

MC-TopLog's accuracy line in Fig. 9 is derived under the learning mode of co-generalisation,while solo-generalisation is impractical for this learning task. Because there are so many mathematical functions which can be fit into a single example that the size of candidate hypotheses is much larger than what YAP (a Prolog interpreter) can handle. Therefore the null hypothesis (b) is refuted since Fig. 13 shows that ⊤DTcD is applicable for this learning task where ⊤DTD fails due to a too large search space. Fig. 13 also shows that the power of co-generalisation is more effective with more examples. As can be seen from Fig. 13, the number of search nodes decreases dramatically with increasing number of examples. This is consistent with the fact that the common part of different sets shrinks as the number of sets increases. In terms of running time, it decreases accordingly with the decreasing search space, as shown in Fig. 14. However, the running time increases slightly after the number of examples increases to 20. This is due to the counteracting effect of binding more variables.

$play([HeapN_1, HeapN_2, HeapN_3]) \leftarrow$
$\quad xor(HeapN_1, HeapN_2, HeapN_3).$
$play([HeapN_1, HeapN_2, HeapN_3, HeapN_4]) \leftarrow$
$\quad xor(HeapN_1, HeapN_2, MidResult),$
$\quad xor(MidResult, HeapN_3, HeapN_4).$

(a) H_s by Progol

$play(Heaps) \leftarrow compute(Heaps, 0, Result).$
$compute([Heap|Heaps], ResultSofar, Result) \leftarrow$
$\quad xor(Heap, ResultSofar, NewResultSofar),$
$\quad compute(Heaps, NewResultSofar, Result).$

(b) H_m by MC-TopLog

Fig. 10. Hypotheses suggested by different ILP systems

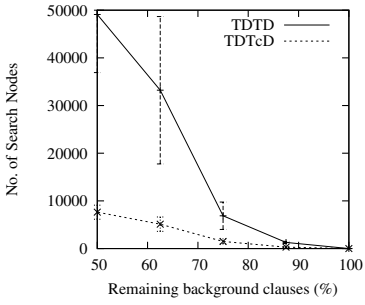


Fig. 11. Search Spaces (Grammar)

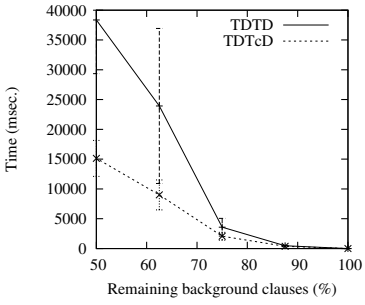


Fig. 12. Running Time (Grammar)

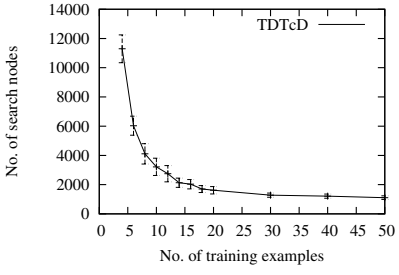


Fig. 13. Search Spaces (Nim)

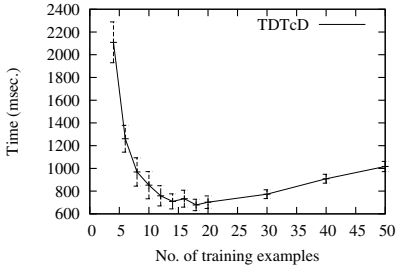


Fig. 14. Running Time (Nim)

5 Conclusions and Future Work

The simplified version of grammar learning shows the importance of having a complete method, even for learning problems without recursion and mutually dependent predicates. Both ⊤TDTD and ⊤TDTcD are sound and complete for deriving hypotheses, but ⊤TDTcD is more efficient than ⊤TDTD, while the improvement in efficiency does not come at the cost of lower predictive accuracy. We intend to compare MC-TopLog to other complete systems in future work.

Acknowledgements. This work is part of the Syngenta University Innovation Centre (UIC) on Systems Biology at Imperial College, which is funded by Syngenta Ltd. The first author also would like to thank the Royal Academy of Engineering and Microsoft for funding his present 5 year Research Chair. The authors also want to thank Changze Xu for providing the data set about learning game strategies.

References

1. Blockeel, H., De Raedt, L.: Top-down induction of first order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
2. Boström, H., Idestam-Almquist, P.: Induction of logic programs by example-guided unfolding. The Journal of Logic Programming 40, 159–183 (1999)

3. Bratko, I.: Refining Complete Hypotheses in ILP. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 44–55. Springer, Heidelberg (1999)
4. Cohen, W.: Grammatically biased learning: Learning logic programs using an explicit antecedent description language. Artificial Intelligence 68, 303–366 (1994)
5. Inoue, K.: Induction as consequence finding. Machine Learning 55, 109–135 (2004)
6. Kedar-Cabelli, S.T., McCarty, L.T.: Explanation-based generalization as resolution theorem proving. In: Proceedings of ICML 1987, pp. 383–389. Morgan Kaufmann, Los Altos (1987)
7. Kimber, T., Broda, K., Russo, A.: Induction on Failure: Learning Connected Horn Theories. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 169–181. Springer, Heidelberg (2009)
8. Lin, D.: Efficient, complete and declarative search in inductive logic programming. Master's thesis, Imperial College London (September 2009)
9. Lin, D., Chen, J., Watanabe, H., Muggleton, S.H., Jain, P., Sternberg, M., Baxter, C., Currie, R., Dunbar, S., Earll, M., Salazar, D.: Does Multi-clause Learning Help in Real-world Applications? In: Muggleton, S.H., Tamaddoni-Nezhad, A., Lisi, F.A. (eds.) ILP 2011. LNCS (LNAI), vol. 7207, pp. 222–238. Springer, Heidelberg (2012)
10. Malerba, D.: Learning recursive theories in the normal ILP setting. Fundamenta Informaticae 57, 39–77 (2003)
11. Muggleton, S.H.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
12. Muggleton, S.H., Bryant, C.H.: Theory Completion Using Inverse Entailment. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 130–146. Springer, Heidelberg (2000)
13. Muggleton, S.H., Feng, C.: Efficient induction of logic programs. In: ALT 1990, pp. 368–381. Ohmsha, Tokyo (1990)
14. Muggleton, S.H., Santos, J.C.A., Tamaddoni-Nezhad, A.: TopLog: ILP Using a Logic Program Declarative Bias. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 687–692. Springer, Heidelberg (2008)
15. Muggleton, S., Santos, J., Tamaddoni-Nezhad, A.: ProGolem: A System Based on Relative Minimal Generalisation. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 131–148. Springer, Heidelberg (2010)
16. Muggleton, S.H., Xu, C.: Can ILP learn complete and correct game strategies? In: Late-breaking Proceedings of ILP. Imperial College London Press (2011)
17. Plotkin, G.D.: Automatic Methods of Inductive Inference. PhD thesis, Edinburgh University (August 1971)
18. De Raedt, L., Dehaspe, L.: Clausal discovery. Machine Learning 26, 99–146 (1997)
19. De Raedt, L., Lavrac, N., Dzeroski, S.: Multiple predicate learning. In: IJCAI, pp. 1037–1043 (1993)
20. Ray, O.: Nonmonotonic abductive inductive learning. Journal of Applied Logic 7(3), 329–340 (2009)
21. Reynolds, J.C.: Transformational systems and the algebraic structure of atomic formulas. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 5, pp. 135–151. Edinburgh University Press, Edinburgh (1969)
22. Yamamoto, A.: Which Hypotheses can be Found with Inverse Entailment? In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 296–308. Springer, Heidelberg (1997)