

# Was the Year 2000 a Leap Year? Step-wise Narrowing Theories with Metagol

Michael Siebers<sup>[0000-0003-1891-3572]</sup>, ✉ and Ute Schmid

Cognitive Systems, University of Bamberg, Germany  
{[michael.siebers](mailto:michael.siebers), [ute.schmid](mailto:ute.schmid)}@uni-bamberg.de

**Abstract.** Many people believe that every fourth year is a leap year. However, this rule is too general: year  $X$  is a leap year if  $X$  is divisible by 4 *except* if  $X$  is divisible by 100 *except* if  $X$  is divisible by 400. We call such a theory with alternating generalisation and specialisation a step-wise narrowed theory. We present and evaluate an extension to the ILP system Metagol which facilitates learning such theories. We enabled Metagol to learn over-general theories by allowing a limited number of false positives during learning. This variant is iteratively applied on a learning task. For each iteration after the first, positive examples are the false positives from the previous iteration and negative examples are the true positives from the previous iteration. Iteration continues until no more false positives are present. Then, the theories are combined to a single step-wise narrowed theory. We evaluate the usefulness of our approach in the leap year domain. We can show that our approach finds solutions with fewer clauses, higher accuracy, and in shorter time.

**Keywords:** step-wise narrowed theory · Metagol · over-generalization

## 1 Introduction

Most people assume that every fourth year, that is every year divisible by four, is a leap year. Though this is correct in most cases, on average 3% of years identified as leap years by this rule are no leap years. The rule is slightly too general when compared to the true leap year rule:

Every year that is exactly divisible by 4 is a leap year, *except* for years that are exactly divisible by 100, *but* these centurial years are leap years if they are exactly divisible by 400. [1, p. 599; emphasis by the authors]

The exact leap year rule follows a not uncommon structure:  $A$  holds if  $B$  holds *but not* if  $C$  holds. Rules of this structure can be used to explain, for example family relations: person  $X$  is Person  $Y$ 's half brother if  $X$  is male and both have the same mother *but not* if they also have the same father (or vice versa). In Quinlan's famous Saturday example [2, EnjoySport in 3, also known as play-golf or play-tennis] a Saturday is a positive day if the outlook is overcast or if there is no wind *unless* the outlook is sunny. These examples are characterized using one exception rule from a general rule. This schema can be applied cascadingly to

constrain an exception rule by another exception rule. For example, a company might have the policy that business trips must be made by train unless the trip by air plane saves at least 30% travel time. However, the train must be used if it is at least 25% cheaper.

There are several ILP systems which are able to learn rules similar to the ones presented above. The classical system CIGOL [4, 5] induces rules in interactive sessions. The user presents examples to the system which in turn generalizes rules from theses. Then, the user must specify whether a presented rule is always correct. Exceptions are handled by adding a new auxiliary predicate to the original rule which holds for all counterexamples. No rules can be learned for the auxiliary predicate. ATRE [6, 7] is especially suited to learn (mutual) recursive rules. The rules to be learned are organised in layers such that rules in higher layers only depend on rules in lower layers. When adding a new rule would lead to inconsistencies in lower layers involved predicates are renamed to new auxiliary predicates. XHAIL [8] combines abduction, deduction, and induction to learn rules incorporating negation. Predicates which may be negated must be explicitly named, as no auxiliary predicates can be invented.

Inventing auxiliary predicates, known as predicate invention [9], is a key concept to the state-of-the-art ILP system Metagol. In recent years, Metagol has been successfully applied in different domains such as learning grammar rules [10], functional string transformations [11], and higher-order theories [12]. This use of predicate invention has been shown to be beneficial regarding theory size, learning time, and predictive accuracy [13]. However, Metagol cannot use negation in learned theories.<sup>1</sup> We will extend Metagol to incorporate exceptions.

**Our contributions.** In this paper, we define the concept of step-wise narrowed theories and introduce  $\text{Metagol}_{SN}$  which can induce a subset of such theories. In detail,

- we define step-wise narrowed theories,
- we relax Metagol’s prohibition to cover negative examples,
- we define  $\text{Metagol}_{SN}$  by using this relaxed Metagol recursively, and
- we show that  $\text{Metagol}_{SN}$  may yield smaller theories, in number of clauses, may require less background knowledge, and may run faster than Metagol.

In the next section, we introduce the meta-inductive learning framework and its realization in Metagol. Then, we will present our modifications and extensions to meta-inductive learning and Metagol which allow learning step-wise narrowed theories.  $\text{Metagol}_{SN}$  will be evaluated in Section 4. We conclude the paper with a short discussion and further ideas.

---

<sup>1</sup> Of course, predicates with a “negative” semantic, like *not\_father/2*, can be supplied in the background knowledge. Nevertheless, no syntactic negation can be induced.

## 2 Meta-interpretative Learning and Metagol

Metagol is the realization of the meta-interpretative learning (MIL) framework in Prolog. In MIL a higher-order datalog program is learned from examples using abduction. First, we introduce some logic notation.

### 2.1 Logic Notation

A variable is represented by a single upper case letter followed by a string of letters and digits. A function or predicate symbol is a lower case letter followed by a string of letters and digits. The arity of a function or predicate symbol is the number of arguments it takes. A constant is a function symbol with arity zero. The set of all constants is referred to as the constant signature and denoted  $\mathcal{C}$ . The set of all predicate symbols is denoted  $\mathcal{P}$ .

Variables and constants are terms, and a function symbol immediately followed by a bracketed  $n$ -tuple of terms is a term. An atom, or positive literal, is a predicate symbol (of arity  $n$ ) or a variable immediately followed by a bracketed  $n$ -tuple of terms. The negation of an atom  $\neg A$  is called negative literal. A variable is higher-order if it can be substituted for by a predicate symbol.

A finite set of literals is called a clause. A clause represents the disjunction of its literals. A clause is unit if and only if it contains exactly one literal. A Horn clause is a clause which contains at most one positive literal which is then called the head of the clause. The negative literals are collectively called the body of the clause. If a Horn clause contains exactly one positive literal it is called definite.

A clausal theory is a set of clauses and represents the conjunction of its clauses. A clausal theory in which each clause is a definite Horn clause is called a definite program. Literals, clauses, and clausal theories are well-formed formulas (wffs) in which all variables are assumed to be universally quantified. Let  $E$  be a wff and  $\sigma, \tau$  be sets of variables.  $\exists\sigma E$  and  $\forall\tau E$  are wffs.  $E$  is said to be ground whenever it contains no variables.  $E$  is said to be datalog if it contains no function symbols other than constants. A clausal theory which contains only datalog Horn clauses is called a datalog program. The set of all ground atoms constructable from  $\mathcal{P}, \mathcal{C}$  is called the datalog Herbrand base.

### 2.2 Meta-inductive Learning

In MIL a datalog program is abduced from examples where only clauses conforming to one of the user-supplied templates, called meta-rules, may be abduced. Every meta-rule is a uniquely named wff

$$\exists\sigma\forall\tau P(s_1, \dots, s_m) \leftarrow Q_1(t_1^1, \dots, t_{n_1}^1), \dots, Q_r(t_1^r, \dots, t_{n_r}^r)$$

where  $\sigma$  and  $\tau$  are disjoint sets of variables,  $P, Q_1, \dots, Q_r \in \sigma \cup \tau \cup \mathcal{P}$ , and  $s_1, \dots, s_m, t_1^1, \dots, t_{n_r}^r \in \sigma \cup \tau \cup \mathcal{C}$  [13, definition 1]. Table 1 shows some example meta-rules. Muggleton *et al.* define the MIL learning setting as follows [13, definition 2]:

```

metagol(Pos,Neg,Prog) :-
    between(MinClauses,MaxClauses,ClauseLimit),
    prove_all(Pos,[],Prog),
    length(Prog,NProg),
    NProg =< ClauseLimit,
    prove_none(Neg,Prog).

prove_all([],Prog,Prog).
prove_all([Atom|Atoms],Prog1,Prog2) :-
    prove_one(Atom,Prog1,Prog3),
    prove_all(Atoms,Prog3,Prog2).

prove_one(Atom,Prog,Prog) :- call(Atom). % use background knowledge
prove_one(Atom,Prog1,Prog2) :- % use or create abduction
    metarule(Name,MetaSub,(Atom :- Body)),
    store(sub(Name,MetaSub),Prog1,Prog3),
    prove_all(Body,Prog3,Prog2).

prove_none([],Prog).
prove_none([Atom|Atoms],Prog) :-
    not(prove_one(Atom,Prog,Prog)),
    prove_none(Atoms,Prog).

```

**Fig. 1.** Metagol in pseudo Prolog code. *Pos* are positive, *Neg* negative examples. *Prog* is a program. Only the last argument of every predicates is an output argument. Clause limit is modulated from *MinClauses* to *MaxClauses*, which can be set by the user. *metarule/3* gets a meta-rule with existentially quantified variables *MetaSub* from the database, *store/3* tests if this abduction is already known or adds it otherwise.

Given [a set of] meta-rules  $M$ , definite program background knowledge  $B$  and ground positive and negative unit examples  $E^+$ ,  $E^-$ , MIL returns a higher-order datalog program hypothesis  $H$  if one exists such that  $M, B, H \models E^+$  and  $M, B, H, E^-$  is consistent.

### 2.3 Metagol

Metagol is a Prolog meta-interpreter which realizes a limited MIL setting. There are different Metagol variants for special use cases or incorporating different features, for example  $\text{Metagol}_{DF}$  [11] for function induction or  $\text{Metagol}_{CF}$  [10] for learning context free grammars. We focus on the most recent version,  $\text{Metagol}_{AI}$  [12, 14]. If the version is clear from the context we omit the subscript. In the following, we present Metagol’s features and restrictions relevant to this work.

Metagol is a two step approach. First, a candidate program is abduced by successively proving all positive examples. The number of clauses in the candidate program is limited. Second, Metagol verifies that no negative examples can be proven using this program. Prolog’s backtracking ensures that a solution is found if it exists. If no solution can be found within the given clause limit, it is

step-by-step increased using an iterative deepening strategy. Thus, the smallest program (in number of clauses) will be found first. Figure 1 shows the Metagol algorithm in pseudo-code.

In the first step (*prove\_all* in Figure 1), each example is proven either

- using only background knowledge,
- using clauses from the already induced program, or
- by matching it against the head of one meta-rule, adding the resulting clause to the program, and proving the meta-rule body recursively.

In the second step (*prove\_none* in Figure 1), Metagol verifies that no negative example can be proven using only background knowledge and already abduced clause, that is without changing the program.

Technically, background knowledge is provided by indicating eligible predicate symbols and their arity. Clauses for these predicates, possibly including predefined predicates, are provided in the Prolog database. Meta-rules with their name and a list of variables to be existentially quantified are given as clauses with an appropriate head in the Prolog database. Positive and negative examples are presented as lists of positive ground atoms. Metagol assumes that all positive examples share the same predicate symbol and arity. However, this is not strictly required.<sup>2</sup>

### 3 Step-wise Narrowed Theories

In the following, we present a modification to the MIL framework and Metagol to deal with domains such as leap year. That is, we propose a framework for learning over-general theories and their step-wise narrowing to characterize exceptions. First, we define step-wise narrowed theories (SNTs), then we present an induction framework for such theories, and finally, an implementation as  $\text{Metagol}_{SN}$ .

Let  $\Phi : \mathcal{P} \rightarrow \mathcal{P}$  be a mapping between predicate symbols. Then,  $\mathcal{M}_\Phi$  is a mapping between atoms such that  $\mathcal{M}_\Phi : p(t_1, \dots, t_n) \mapsto \Phi(p)(t_1, \dots, t_n)$ .

**Definition 1 (Step-wise Narrowed Theory).** *Every higher-order datalog program is a step-wise narrowed theory (SNT). Let  $H$  be a higher-order datalog program,  $\Phi$  a mapping between predicate symbols and  $S$  a SNT. Then,  $\langle H, \Phi, S \rangle$  is also a SNT.*

**Definition 2 (SNT Depth).** *The depth of a SNT is the number of “unpacking” steps required to access the innermost datalog program:*

$$d(S) = \begin{cases} d(S') + 1 & \text{if } S = \langle H, \Phi, S' \rangle \\ 0 & \text{otherwise} \end{cases}.$$

---

<sup>2</sup> Nevertheless, Metagol can only abduce clauses for the predicate symbol of the first positive example and invented predicate symbols derived thereof.

**Definition 3 (SNT Size).** *The size of a SNT is the number of clauses it contains. If  $H$  has  $n_H$  and  $S$  has  $n_S$  clauses we say  $\langle H, \Phi, S \rangle$  has  $n_H + n_S$  clauses. If SNT  $S_1$  has less clauses than SNT  $S_2$  we say that  $S_1$  is smaller than  $S_2$ .*

Given meta-rules  $M$  and some definite program background knowledge  $B$  the SNT  $\langle H, \Phi, S \rangle$  models the positive ground literal  $A$  if  $A$  is modelled by  $H$  but  $\mathcal{M}_\Phi(A)$  is not modelled by  $H$  and  $S$ , that is

$$M, B, \langle H, \Phi, S \rangle \models A \quad \equiv \quad M, B, H \models A \wedge M, B, H, S \not\models \mathcal{M}_\Phi(A).$$

### 3.1 Inducing Step-wise Narrowed Theories

Inducing a SNT from positive and negative examples is called a *step-wise narrowing learning (SNL) task* or *SNL setting*.

**Definition 4 (SNL setting).** *Given a set of meta-rules  $M$ , definite program background knowledge  $B$ , and ground positive and negative unit examples  $E^+$ ,  $E^-$ , SNL returns a SNT  $S$  if one exists such that  $M, B, S \models E^+$  and  $M, B, S, E^-$  is consistent.*

We propose a general solution to the SNL setting based on MIL. For this purpose, we abolish the constraint that  $M, B, H, E^-$  must be consistent. We call this the *relaxed MIL setting* (rMIL).

**Definition 5 (Relaxed MIL setting).** *Given a set of meta-rules  $M$ , definite program background knowledge  $B$ , a non-negative integer  $l$ , and ground positive and negative unit examples  $E^+$ ,  $E^-$  relaxed MIL returns a higher-order datalog program hypothesis  $H$  if one exists along with a partitioning of  $E^-$  in true negative examples  $E^-_{\ominus}$  and false positive examples  $E^-_{\oplus}$  such that  $E^-_{\oplus}$  has at most  $l$  elements,  $M, B, H \models E^+, E^-_{\oplus}$ , and  $M, B, H, E^-_{\ominus}$  is consistent. We call  $l$  the false positive limit.*

**Proposition 1 (rMIL decidable).** *The rMIL setting is decidable in the case  $M, B, E^+, E^-$  are datalog and  $\mathcal{P}, \mathcal{C}$  are finite.*

*Proof.* Follows from the fact that the Herbrand base is finite.

Our general solution realizes SNL by recursively applying rMIL. Given a set of meta-rules  $M$ , definite program background knowledge  $B$ , and ground positive and negative unit examples  $E^+$ ,  $E^-$  we use rMIL to get a datalog program  $H$  and false positives  $E^-_{\oplus}$ . If there are no false positives,  $H$  is a solution in the SNL setting. Otherwise, we define  $\Phi$  such that every predicate symbol used in  $E^+$  and  $E^-$  is mapped to a new predicate symbol. The solution is then  $\langle H, \Phi, S \rangle$  where  $S$  is the solution to the SNL setting with background knowledge  $B \cup H$ , the mapped false positive examples as positive examples, and the negated mapped positive examples as negative examples. A pseudo-code algorithm for the general solution is provided in Algorithm 1.

---

**Algorithm 1:** Pseudo-code algorithm for the general solution to the SNL setting. `rMIL` denotes the relaxed MIL setting.

---

**Function**  $SNL(M, B, l, E^+, E^-)$  **is**  
  **Input:** meta-rules  $M$ , background knowledge  $B$ , false positive limit  $l$ ,  
  positive examples  $E^+$ , and negative examples  $E^-$   
  **Result:** a SNT  
   $H, E_{\ominus}^-, E_{\oplus}^- \leftarrow \text{rMIL}(M, B, l, E^+, E^-)$  ;  
  **if**  $E_{\oplus}^- = \emptyset$  **then**  
  | **return**  $H$ ;  
  **else**  
  | Let  $\Phi$  map every predicate symbol from  $E^+ \cup E^-$  to a new symbol;  
  |  $E'^+ \leftarrow \{\mathcal{M}_{\Phi}(A) \mid \neg A \in E_{\oplus}^-\}$ ;  
  |  $E'^- \leftarrow \{\neg \mathcal{M}_{\Phi}(A) \mid A \in E^+\}$ ;  
  | **return**  $\langle H, \Phi, SNL(M, B \cup H, E'^+, E'^-) \rangle$ ;  
  **end**  
**end**

---

### 3.2 Metagol<sub>SN</sub>

Based on Cropper & Muggleton’s Metagol<sub>AI</sub> implementation<sup>3</sup> we implemented a Prolog realisation of SNL, Metagol<sub>SN</sub>.<sup>4</sup> As first step, we relaxed Metagol to conform to rMIL. The structure of relaxed Metagol is similar to Metagol. First, a program is abduced from all positive examples as in Metagol. Then, false positives are collected by checking which negative examples can be proven without adding clauses (see Figure 2). We did not change the presentation of meta-rules, background knowledge, or examples.

As second step, we realize Metagol<sub>SN</sub> using relaxed Metagol. As mentioned above, Metagol assumes that all positive examples share the same predicate symbol and arity. We extend that assumption to all examples: Metagol<sub>SN</sub> assumes that all examples share the same predicate symbol and arity. Thus, constructing  $\Phi$  degenerates to inventing one new predicate symbol. We prefix the original predicate symbol with an underscore. Implementing the steps of the general SNL solution is straight forward (cf. Figure 3):

1. We use rMIL to induce a program and collect false positives.
2. If there are no false positives, we return the induced program.
3. Otherwise, we rename predicates in false positives and positive examples, add the program to the Prolog database, and recurse on the new examples.

---

<sup>3</sup> We forked from commit 1524600225a65237de9578e46127049f6f95d1a4 in the GitHub Metagol repository [14].

<sup>4</sup> Metagol<sub>SN</sub> is available at <https://github.com/michael-siebers/metagol/tree/ilp2018>.

```

metagol_relaxed(Pos,Neg,MaxClauses,Prog,FalsePos) :-
    between(1,MaxClauses,ClauseLimit),
    prove_all(Pos,[],Prog),
    length(Prog,NProg),
    NProg =< ClauseLimit,
    prove_some(Neg,Prog,FalsePos).

prove_some([],Prog,[]).
prove_some([Atom|Atoms],Prog,[Atom|Proven]) :-
    prove_some(Atoms,Prog,Proven),
    prove_one(Atom,Prog,Prog).
prove_some([Atom|Atoms],Prog,Proven) :-
    prove_some(Atoms,Prog,Proven),
    not(prove_one(Atom,Prog,Prog)).

```

**Fig. 2.** Relaxed MIL in pseudo Prolog code. *Pos* are positive, *Neg* negative examples. *Prog* is a program and *FalsePos* are false positive examples. For *prove.some/3* the last, for *metagol\_relaxed/5* the last two arguments are output arguments. The maximally allowed number of clauses *MaxClauses* is a user-given parameter. For *prove.all* and *prove.one* see Figure 1.

**Parameters.** For every rMIL run, the *false positive limit* is calculated as the size of the negative examples times the parameter *maximal false positive fraction*, rounded down. As rMIL may result in a program  $H$  such that there are no true negatives, SNL is in general undecidable. For  $\text{Metagol}_{SN}$ , we require that there are strictly less false positives than negative examples, that is *maximal false positive fraction* must be lower than 1. Then, the number of positive and negative examples reduces at every second recursion step and  $\text{Metagol}_{SN}$  is decidable.

To further guide the search, we impose an iterative deepening schema on the size and the depth of the induced SNT. An upper limit for both may be set as a parameter. The depth is deepened within the size of the SNT. That is, size  $n$  with depth  $m + 1$  is explored before size  $n + 1$  with depth  $m$ . This assures that  $\text{Metagol}_{SN}$  finds the smallest theory.

**SNT Flattening.** As  $\text{Metagol}_{SN}$  only induces theories for single predicate symbol arity pairs, any induced SNT can easily be flattened into a single clausal theory. If the induced SNT is a datalog program, thus a clausal theory, no flattening is required. Otherwise, given any SNT  $\langle H, \Phi, S \rangle$  induced for predicate symbol  $p$  and any atom  $A = p(t_1, \dots, t_n)$ , let  $H'$  be a copy of  $H$  where  $\text{not}(\mathcal{M}_\Phi(A))$  is appended to every clause with head  $A$ . Then the flattening of  $\langle H, \Phi, S \rangle$  is the union of the flattening of  $S$  and  $H'$  (see Figure 4).

## 4 Evaluation

To evaluate the usefulness of our approach, we compared  $\text{Metagol}_{SN}$  and  $\text{Metagol}_{AI}$  on the leap year domain. As introduced above, years must be separated in leap



```

metagol_sn(Pos,Neg,MaxClauses,SNT) :-
  let P be the predicate symbol used in Pos,
  between(1,MaxClauses,ClauseLimit),
  metagol_relaxed(Pos,Neg,MaxClauses,Prog1,FalsePos),
  if FalsePos=[]
    SNT = Prog1
  else
    let PPrime be P prefixed with '_',
    let PosNext be FalsePos with P renamed to PPrime,
    let NegNext be Pos with P renamed to PPrime,
    assert_prog(Prog1),
    metagol_sn(PosNext,NegNext,MaxClauses - length of Prog1,Prog2),
    SNT=snt(Prog1,PPrime,Prog2)
  end if.

```

**Fig. 3.** Metagol<sub>SN</sub> in pseudo Prolog code. *Pos* are positive, *Neg* negative examples. *assert\_prog/1* adds its argument to the Prolog database and removes it on backtracking. Maximal theory size *MaxClauses* is configurable by the user.

```

flatten_snt(P,snt(Prog1,PPrime,Prog2),Prog3) :-
  flatten_snt(PPrime,Prog2,Prog4),
  for each P(T1,...,TN) :- Body in Prog1
    replace it with P(T1,...,TN) :- Body, not(PPrime(T1,...,TN)),
  append(Prog1,Prog4,Prog3).
flatten_snt(P,Prog,Prog).

```

**Fig. 4.** Flattening of a SNT in pseudo Prolog code. *P* and *PPrime* are predicate symbols, *ProgN* are programs. *P* is the predicate symbol *Prog1* was induced for.

years and not leap years. That is, the target predicate *leapyear(X)* shall hold if and only if *X* is a leap year. We defined two predicates describing numbers:

***divisible/2*** where *divisible(X,Y)* holds if and only if the integer *Y* divides the integer *X* exactly and

***not\_divisible/2*** where *not\_divisible(X,Y)* holds if and only if *X* and *Y* are integers and *divisible(X,Y)* does not hold.

Using these two predicates as background knowledge, two clauses suffice to solve the leap year problem (Figure 5).

To explore the effect of narrowing, first, we focus on a single induction episode on a very small set of examples. As a proof of concept we want to show that extending Metagol with narrowing will result in smaller theories, which can be induced in shorter time, with less background knowledge. Afterwards, we will compare both systems on a larger dataset, reporting predictive accuracy and run times.

**Table 1.** Meta-rules used during evaluation. Name, well-formed formula, existential quantified variables  $\sigma$ , and universal quantified variables  $\tau$  are shown.

name	wff	$\sigma$	$\tau$
Const	$P(A, B) \leftarrow$	$\{P, B\}$	$\{A\}$
And1	$P(A) \leftarrow Q(A), R(A)$	$\{P, Q, R\}$	$\{A\}$
Chain	$P(A, B) \leftarrow Q(A, C), R(C, B)$	$\{P, Q, R\}$	$\{A, B, C\}$
Curry	$P(A) \leftarrow Q(A, B)$	$\{P, Q, B\}$	$\{A\}$

leapyear(X)  $\leftarrow$  divisible(X,4), not\_divisible(X,100).  
leapyear(X)  $\leftarrow$  divisible(X,400).

**Fig. 5.** Clausal theory to classify leap years using background knowledge  $B_{ndiv}$ .

#### 4.1 Proof of Concept Evaluation

As small proof of concept, we tried to induce the leap year rule from a limited number of examples using  $\text{Metagol}_{SN}$  and  $\text{Metagol}_{AI}$ . For positive examples, we used the years 4, 20, and 400. Negative examples were the years 2, 100, and 200. The positive examples were either sorted ascending (dataset *poc-asc*) or descending (dataset *poc-desc*), negative examples were always sorted descending.

**Meta-rules and Background Knowledge.** We provided the meta-rules *And1* and *Curry* (see Table 1) to both systems. We tested the systems with three sets of background knowledge  $B_{div}$ ,  $B_{ndiv}$ , and  $B_{max}$ . All three contain the predicate *divisible/2*, while only  $B_{ndiv}$  and  $B_{max}$  contain *not\_divisible/2*. Both predicates were provided as rules. Since the arguments of the predicates are integers, the set of constants  $\mathcal{C}$  is infinite and thus learning is neither in the MIL setting nor in the SNL setting decidable. In order to have a decidable problem, we limit possible constant choices to a finite number. For *divisible(X, Y)* and *not\_divisible(X, Y)* in  $B_{div}$  and  $B_{ndiv}$ ,  $Y$  may be any natural number from 1 to  $X$ . In  $B_{max}$ ,  $Y$  may be any natural number which exactly divides any positive or negative example.  $X$  is naturally constraint by the example years. An overview of the defined variants of background knowledge is given in Table 2.

**Parameters.** For  $\text{Metagol}_{SN}$  we varied the maximal allowed false positive fraction with possible values 0.34, 0.5, 0.67, and 0.75. Additionally, we limited the SNT depth to 1, 2, or 3. For both systems learning was cut off after 30 minutes.

**Results.** Both  $\text{Metagol}_{AI}$  and  $\text{Metagol}_{SN}$  were able to learn the correct theory (in some configurations) as shown in Figures 6 and 7. Main results for single runs on the *poc-asc* dataset are given in Table 3 and on the *pos-desc* dataset in Table 4.  $\text{Metagol}_{AI}$  was not able to induce a theory using  $B_{div}$ , neither for the *poc-asc* data set nor for *poc-desc*. However, this was to be expected as there is no datalog solution to the learning task with this background knowledge.

**Table 2.** Background knowledge defined for leap year domain.  $divisible(X, Y)$  and  $not\_divisible(X, Y)$  indicate whether these predicates are available in the background knowledge.

	$B_{div}$	$B_{ndiv}$	$B_{max}$
$divisible(X, Y)$	✓	✓	✓
$not\_divisible(X, Y)$	X	✓	✓
Constraint on $Y$	$1 \leq Y \leq X$		$Y$ divides any example

$leapyear(A) \leftarrow leapyear\_1(A), leapyear\_2(A).$   
 $leapyear(A) \leftarrow divisible(A, 16).$   
 $leapyear\_1(A) \leftarrow divisible(A, 4).$   
 $leapyear\_2(A) \leftarrow not\_divisible(A, 25).$

**Fig. 6.** Leap year theory induced by  $Metagol_{AI}$  for  $poc-desc$  using  $B_{max}$ .

Using  $B_{ndiv}$   $Metagol_{AI}$  failed on  $poc-asc$  within short time (0.7 s). This is due to the ordering of the examples and the imposed constraint on the numeric constants. In  $poc-asc$  the number 4 is the first positive example. Since 25 and 400 are greater than 4, neither  $not\_divisible(X, 25)$  nor  $divisible(X, 400)$  can be abduced from this example. Learning fails as no theory can be found which is consistent with the negative examples. For descendent ordering this does not hold. Either  $not\_divisible(X, 25)$  and  $divisible(X, 400)$  could be abduced from the first example (400).

However, reordering the data set seems to increase the search space massively. This hypothesis is supported by the observations that  $Metagol_{AI}$  timed out on this data set and that  $Metagol_{SN}$  timed out or took much longer to induce a theory. Even in failed attempts to induce a theory, both  $Metagol_{AI}$  and  $Metagol_{SN}$  took longer for  $poc-desc$  than  $poc-asc$ . We conclude that  $Metagol_{AI}$  and  $Metagol_{SN}$  are susceptible to the ordering of examples.

Finally,  $B_{max}$  allowed both  $Metagol$  systems to induce theories. There are no differences in results between  $poc-asc$  and  $poc-desc$ . However,  $Metagol_{SN}$  always found a theory with 3 clauses whereas  $Metagol_{AI}$  required 4 clauses. Regarding learning time, differences between the systems are negligible.

To induce a theory  $Metagol_{SN}$  required a SNT depth of two for  $B_{div}$ . And a depth of one was sufficient for  $B_{ndiv}$  and  $B_{max}$ , but a depth of two produced smaller theories in shorter time for  $B_{ndiv}$ . Increasing the depth limit further did not change performance. A maximal false positive fraction of 67%, or two of the initial negative examples, was required for  $B_{div}$  and  $B_{ndiv}$ . A higher value did not improve performance further. For  $B_{max}$ , a maximal false positive fraction of 34%, or one of the initial negative examples, delivered good results. No improvement with increasing value can be deduced.

```

leapyear(A) ← divisible(A,4), not(_leapyear(A)).
_leapyear(A) ← divisible(A,100), not(_leapyear(A)).
_leapyear(A) ← divisible(A,400).

```

**Fig. 7.** Leap year theory induced by  $\text{Metagol}_{SN}$  for *poc-asc* using  $B_{div}$ . Maximal allowed false positive fraction was 0.67, SNT depth was limited to 2.

**Table 3.** Results for single runs on *poc-asc* data set. Shown are the maximal allowed false positive fraction (max. FP), the limit on the SNT depth (limit), the number of clauses in the learned theory (#c) where – denotes a failed run, the SNT depth (d), and the used time in seconds (t).

	max. FP	limit	$B_{div}$			$B_{ndiv}$			$B_{max}$		
			#c	d	t	#c	d	t	#c	d	t
Metagol <sub>AI</sub>			–	–	0.4	–	–	0.7	4	0	0.2
Metagol <sub>SN</sub>	0.34	1	–	–	0.5	–	–	1.0	3	1	0.0
	0.50	1	–	–	0.5	–	–	1.0	3	1	0.0
	0.67	1	–	–	1.7	4	1	0.3	3	1	0.0
		2	3	2	0.0	3	2	0.0	3	2	0.0
0.75	1	–	–	1.7	4	1	0.3	3	1	0.0	
	2	3	2	0.0	3	2	0.0	3	2	0.0	

## 4.2 Performance Evaluation

Based on the results of the proof of concept we conducted a larger experiment to study the influence of SNL parameters on performance. Meta-rules are as for the small data sets. To have a fair comparison for both systems, we choose to use only  $B_{max}$  in our experiments.

**Examples.** As examples we use all years from 1582, where leap years following the current rule were introduced with the Gregorian calendar [1], to the current year. Years were separated in leap years and not leap years using the rule from Figure 5. For the experiments we randomly sampled 20%, 40%, 60%, 80%, or 100% of the positive and negative examples. Metagol<sub>AI</sub> and Metagol<sub>SN</sub> were evaluated on the same samples.

**Parameters.** We varied the maximal allowed false positive fraction with possible values 0.05, 0.10, 0.25, 0.50, 0.75, and 0.95, and the depth limit with possible values 1, 2, or 5. Every parameter combination and Metagol<sub>AI</sub> were evaluated 10 times, each time with a different sampling. Learning was cut off after 30 minutes.

**Evaluation.** We evaluated the systems on theory complexity (number of clauses and SNT depth), learning time, and predictive accuracy. Therefore, we applied the induced theories on the next thousand years (2019–3018).

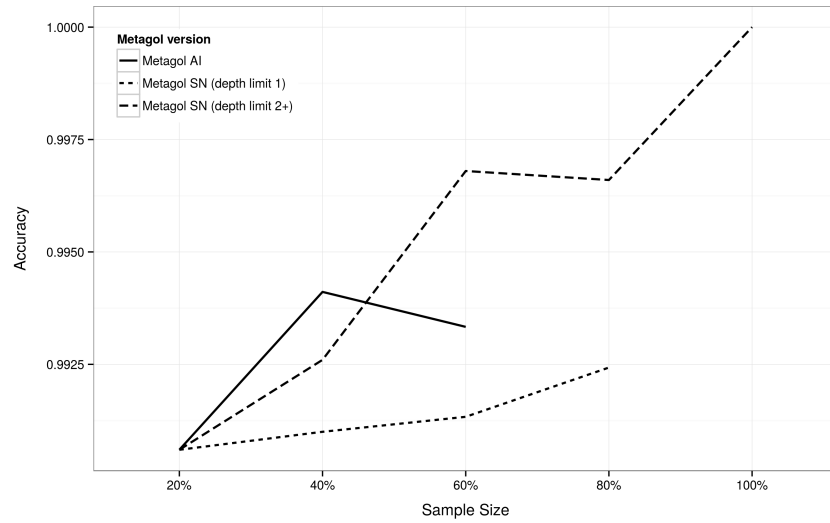
**Table 4.** Results for single runs on *poc-desc* data set. Shown are the maximal allowed false positive fraction (max. FP), the limit on the SNT depth (limit), the number of clauses in the learned theory ( $\#c$ ) where  $-$  denotes a failed run, the SNT depth (d), and the used time in seconds (t).  $\dagger$  denotes that the run did not finish within 30 minutes.

	max. FP	limit	$B_{div}$			$B_{ndiv}$			$B_{max}$		
			$\#c$	d	t	$\#c$	d	t	$\#c$	d	t
Metagol <sub>AI</sub>			-	-	14.6	-	-	$\dagger$	4	0	0.1
Metagol <sub>SN</sub>	0.34	1	-	-	17.1	-	-	$\dagger$	3	1	0.0
	0.5	1	-	-	17.3	-	-	$\dagger$	3	1	0.0
	0.67	1	-	-	34.0	4	1	25.8	3	1	0.0
		2	3	2	0.0	3	2	0.1	3	2	0.0
	0.75	1	-	-	33.7	4	1	25.8	3	1	0.0
2		3	2	0.0	3	2	0.1	3	2	0.0	

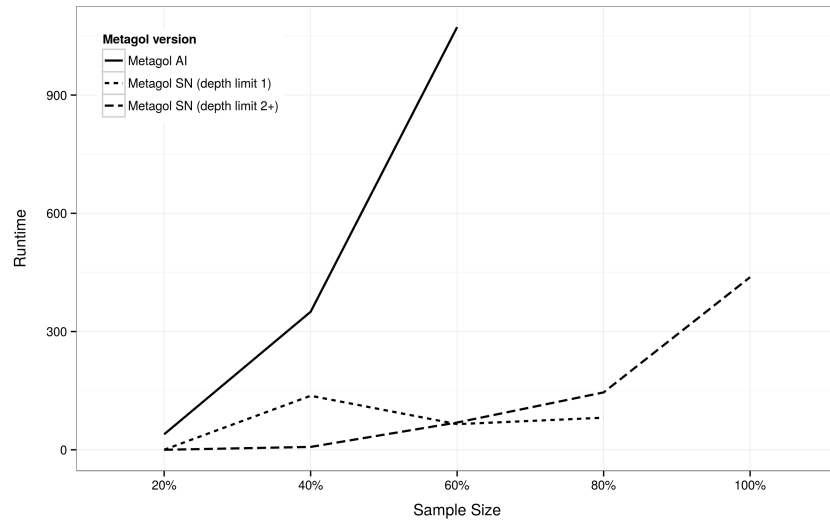
**Results.** In general, accuracies are very high (greater than or equal to 97.6%) for all sample sizes and all SNT depth limits (see Figure 8). However, Metagol<sub>AI</sub> was not able to learn for sample sizes larger than 60% with the given time limit. Metagol<sub>SN</sub> was able to learn for all sample sizes when allowing up to 25% false positives and a SNT depth of two or five. For SNT depth one, Metagol<sub>SN</sub> was able to learn from 80% of the data but not from 100% within the time limit. When training on the complete data set succeeded, Metagol<sub>SN</sub> has an accuracy of 100%. For both systems runtime increases with sample size (see Figure 9). In general, Metagol<sub>AI</sub> has higher run times than Metagol<sub>SN</sub>. For both systems, the number of time-outs increases with sample size. For Metagol<sub>SN</sub> time-outs increase with maximal allowed false positive fraction.

## 5 Conclusion

We introduced the concept of step-wise narrowed theories to tackle domains which can best be characterized by an over-general rule and cascading exceptions. We could show that the ILP system Metagol which allows predicate invention but cannot deal with negation can be extended to learn such theories. Evaluation has been performed on the leap year domain. As a next step, we plan to compare Metagol<sub>SN</sub> with other systems, like ATRE and XHAIL, on the leap year and additional domains. For example, we want to investigate rule plus exception concepts in the Michalski train domain [15]. Learning a rule such as, *a train is eastbound if it contains a rectangle in any carriage but not in the first one*. Furthermore, we expect to gain better performances for learning grammars for facial expressions of pain than with a previously explored grammar induction approach [16]. Relaxing the MIL framework allows learning of imperfect theories. This could be exploited in an iterative deepening approach on the false positive limit, producing a stream of improving theories, that is an anytime algorithm.



**Fig. 8.** Average accuracies (standard deviations are similar for all cases and approximately 0.05) for different sample sizes given different SNT depth limits. Results for depth limits 2 and 5 were identical and thus are show in a single line. Metagol<sub>SN</sub> was run with 0.05 maximal allowed false positives.



**Fig. 9.** Average run times for different sample sizes given for different SNT depth limits. Results for depth limits 2 and 5 were identical and thus are show in a single line. Metagol<sub>SN</sub> was run with 0.05 maximal allowed false positives.

**Acknowledgements.** We like to thank Andrew Cropper for valuable discussions on negation in Metagol. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SCHM 1239/10-1.

## References

1. Richards, E.G. *Calendars* in *Explanatory Supplement to the Astronomical Almanac* (eds Urban, S.E. & Seidelmann, P.K.) (2013), 585–624.
2. Quinlan, J.R. Induction of Decision Trees. *Machine Learning* **1**, 81–106 (1986).
3. Mitchell, T.M. *Machine Learning* (McGraw-Hill, 1997).
4. Muggleton, S. & Buntine, W. *Machine Invention of First-order Predicates by Inverting Resolution* in *Machine Learning Proceedings* (1988), 339–352. doi:10.1016/B978-0-934613-64-4.50040-2.
5. Bain, M. & Muggleton, S. *Non-monotonic Learning* in *Machine Intelligence 12 - Towards an Automated Logic of Human Thought* (1991), 105–120.
6. Malerba, D., Esposito, F. & Lisi, F.A. *Learning Recursive Theories with ATRE* in *ECAI* (1998), 435–439.
7. Malerba, D. Learning Recursive Theories in the Normal ILP Setting. *Fundamenta Informaticae* **57**, 39–77 (Oct. 16, 2003).
8. Ray, O. Nonmonotonic Abductive Inductive Learning. *Journal of Applied Logic* **7**, 329–340 (2009).
9. Stahl, I. *Predicate Invention in ILP – an Overview* in *Machine Learning: ECML-93* (1993), 313–322.
10. Muggleton, S.H., Lin, D., Pahlavi, N. & Tam. Meta-interpretive Learning: Application to Grammatical Inference. *Machine Learning* **94**, 25–49 (2014).
11. Lin, D., Dechter, E., Ellis, K., Tenenbaum, J. & Muggleton, S.H. *Bias Reformulation for One-shot Function Induction* in *ECAI* (2014), 525–530. doi:10.3233/978-1-61499-419-0-525.
12. Cropper, A. & Muggleton, S.H. *Learning Higher-order Logic Programs through Abstraction and Invention* in *IJCAI* (2016), 1418–1424.
13. Muggleton, S.H., Lin, D. & Tamaddoni-Nezhad, A. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning* **100**, 49–73 (2015).
14. Cropper, A. & Muggleton, S.H. *Metagol System* <https://github.com/metagol/metagol>. 2016.
15. Larson, J. & Michalski, R.S. Inductive Inference of VL Decision Rules. *ACM SIGART Bulletin*, 38–44 (June 1977).
16. Siebers, M., Schmid, U., Seuß, D., Kunz, M. & Lautenbacher, S. Characterizing facial expressions by grammars of action unit sequences – A first investigation using ABL. *Information Sciences* **329**, 866–875 (Feb. 2016).