Self-Supervised Inductive Logic Programming

Stassa Patsantzis

University of Surrey
s.patsantzis@surrey.ac.uk

Abstract

Inductive Logic Programming (ILP) approaches like Meta - Interpretive Learning (MIL) can learn, from few examples, recursive logic programs with invented predicates that generalise well to unseen instances. This ability relies on a background theory and negative examples, both carefully selected with expert knowledge of a learning problem and its solutions. But what if such a problem-specific background theory or negative examples are not available? We formalise this question as a new setting for Self-Supervised ILP and present a new MIL algorithm that learns in the new setting from some positive labelled, and zero or more unlabelled examples, and automatically generates, and labels, new positive and negative examples during learning. We implement this algorithm in Prolog in a new MIL system, called Poker. We compare Poker to state-of-the-art MIL system Louise on experiments learning grammars for Context-Free and L-System languages from labelled, positive example strings, no negative examples, and just the terminal vocabulary of a language, seen in examples, as a first-order background theory. We introduce a new approach for the principled selection of a second-order background theory as a Second Order Definite Normal Form (SONF), sufficiently general to learn all programs in a class, thus removing the need for a backgound theory tailored to a learning task. We find that Poker's performance improves with increasing numbers of automatically generated examples while Louise, bereft of negative examples, over-generalises.

Code — https://github.com/stassa/aaai_26_experiments/
Appendices — https://arxiv.org/abs/2507.16405

Introduction

In the Standard Setting (Nienhuys-Cheng and de Wolf 1997) for Inductive Logic Programming (Muggleton 1991) (ILP) a background theory B and positive and negative examples E^+, E^- are used as training data to learn a *correct* hypothesis H that accepts, in conjunction with B, all examples in E^+, and no examples in E^-. Typically, E^+, E^- are selected, and B programmed, manually by the user according to their background knowledge of the learning target (i.e. the logic program to be learned). B in particular is tailored to each learning target (Flener and Yilmaz 1999), and E^- hand-picked to avoid over-generalisation. Given the right B, E^+ and E^-, recent ILP systems can learn correct hypotheses with recursive clauses and invented predicates.

The ability to include background knowledge in training data is a desirable characteristic of ILP systems, but the practice of manually creating a target-specific background theory and selecting negative examples is a constant burden that limits the real-world application of ILP approaches.

In this paper we present a new way to alleviate the aforementioned burden on the user with a new algorithm for Self-Supervised ILP, more specifically, Self-Supervised Meta-Interpretive Learning(Muggleton et al. 2014) (MIL). The new algorithm, implemented in a new MIL system called Poker[1], is "self-supervised" in the sense that it learns from both labelled positive and unlabelled examples and it automatically generates new positive and negative examples during learning. Because it generates new negative examples, Poker can learn from a background theory that is not tailored to the learning target without over-generalising. Poker returns not only a hypothesis, but also a *labelling* of unlabelled, and automatically generated, examples.

We illustrate the use of Poker in Table 1 where Poker is given 3 positive, labelled examples, E^+, of the $\{1^n0^n : n > 0\}$ (1^n0^n) Context-Free Language (CFL), and 30 unlabelled examples, $E^?$ of the $\{1^n0^m : n \geq m \geq 0\}$ (1^n0^m) CFL which includes 1^n0^n as a subset. Examples are atoms in Definite Clause Grammars (DCG) notation (Pereira and Shieber 1987) representing strings of 1^n0^n and 1^n0^m. The first-order background theory, B, consists of just the terminal vocabulary of both languages, $\{1, 0, \epsilon\}$, defined as a set of DCG pre-terminals. 1^n0^n and 1^n0^m are more often denoted as a^nb^n and a^nb^m respectively, but replacing a, b with $1, 0$ makes it possible for B to express any grammar with two terminal symbols suitably mapped to 1 and 0. B can also be constructed *automatically* from $E^+, E^?$. A second-order background theory, $\mathcal{M}$, includes two *metarules*, *Chain* and *Identity*, Second-Order definite clauses with a set of constraints encoding a *Second Order Definite Normal Form* (SONF)[2]. The background theory $B \cup \mathcal{M}$ is thus sufficiently general to express, as a DCG, a Context-

[1] Named after, not the game, but *Wittgenstein's Poker*; a friendly dig at Popper (Cropper and Morel 2021).

[2] SONFs are formalised in the Framework Section.

Self-supervised ILP with Poker

Labelled examples $\mathbf{E^+}$:$\mathbf{1^n0^n(n>0)}$	**Unlabelled examples $\mathbf{E^?}$:$\mathbf{1^n0^m(n \geq m \geq 0)}$** (21 of 100 for $n \in [0,18]$)						
$s(1^1,0^1)$.	$s(1^0,0^0)$.	$s(1^3,0^0)$.	$s(1^6,0^0)$.	$s(1^9,0^9)$.	$s(1^7,0^7)$.	$s(1^6,0^6)$.	$s(1^5,0^3)$.
$s(1^2,0^2)$.	$s(1^1,0^0)$.	$s(1^4,0^0)$.	$s(1^7,0^0)$.	$s(1^8,0^8)$.	$s(1^6,0^1)$.	$s(1^5,0^1)$.	$s(1^5,0^5)$.
$s(1^3,0^3)$.	$s(1^2,0^0)$.	$s(1^5,0^0)$.	$s(1^8,0^0)$.	$s(1^7,0^1)$.	$s(1^6,0^2)$.	$s(1^5,0^2)$.	$s(1^4,0^1)$.

$\mathbf{B}$: First-order background theory

$one([1|X],X)$. $zero([0|X],X)$. $empty(X,X)$.

$\mathcal{M}$: Second-Order Background Theory

(Identity) $P(x,y) \leftarrow Q(x,y)$: $target(P) \wedge background(Q) \vee empty(Q)$

(Chain) $P(x,y) \leftarrow Q(x,z),R(z,y)$: $P \neq Q \wedge (target(P) \vee invented(P)) \wedge not(target(Q)) \wedge not(empty(Q,R)) \wedge invented(P,Q) \rightarrow P \neq Q$

Learned Hypothesis

With invented predicates ($s_1/2$):	$s(A,B) \leftarrow one(A,C),zero(C,B)$.	$s(A,B) \leftarrow s_1(A,C),zero(C,B)$. $s_1(A,B) \leftarrow one(A,C),s(C,B))$.
Unfolded to remove invented predicates:	$s(A,B) \leftarrow one(A,C),zero(C,B)$.	$s(A,B) \leftarrow one(A,C),s(C,D),zero(D,B)$.

Labelling (includes both automatically generated and user-given unlabelled examples)

Labelled Positive			**Labelled Negative** (21 of 175)						
$s(1^1,0^1)$.	$s(1^4,0^4)$.	$s(1^7,0^7)$.	$s(1^0,0^0)$.	$s(1^2,0^1)$.	$s(1^3,0^2)$.	$s(1^4,0^2)$.	$s(1^5,0^1)$.	$s(1^6,0^0)$.	$s(1^7,0^0)$.
$s(1^2,0^2)$.	$s(1^5,0^5)$.	$s(1^8,0^8)$.	$s(1^1,0^0)$.	$s(1^3,0^0)$.	$s(1^4,0^0)$.	$s(1^4,0^3)$.	$s(1^5,0^2)$.	$s(1^6,0^1)$.	$s(1^7,0^1)$.
$s(1^3,0^3)$.	$s(1^6,0^6)$.	$s(1^9,0^9)$.	$s(1^2,0^0)$.	$s(1^3,0^1)$.	$s(1^4,0^1)$.	$s(1^5,0^0)$.	$s(1^5,0^3)$.	$s(1^6,0^2)$.	$s(1^8,0^0)$.

Table 1: Poker inputs and outputs. Example strings pretty-printed from DCG notation (e.g. $s([1,1,0,0],[]) \rightarrow s(1^2,0^2)$).

Free Grammar (CFG) of any *bit-string* CFL or Regular language, i.e. one with a vocabulary of at most two characters and ϵ.

The maximal generality of $B \cup \mathcal{M}$ in Table 1 achieves two purposes. On the one hand it guarantees that $B \cup \mathcal{M}$ is general enough to learn the target grammar, i.e. a CFG of 1^n0^n. On the other hand, $B \cup \mathcal{M}$ is no longer target specific: instead of being tailored to one learning target, 1^n0^n, it can be reused to learn any bit-string CFG. At the same time, the generality of $B \cup \mathcal{M}$ introduces a problem: in the absence of negative examples, it is impossible to distinguish 1^n0^n from 1^n0^m (the language of $E^?$). Indeed, without negative examples it is impossible to distinguish any bit-string language from $\{0,1\}^*$ the maximally general language of all bit-strings. In order to learn 1^n0^n without over-generalising, therefore, negative examples are necessary. Poker can generate negative examples automatically, thus avoiding over-generalisation.

It is also possible to avoid over-generalisation by learning a hypothesis that only accepts E^+, i.e. over-fitting E^+. Poker returns a labelling of $E^?$ which, in Table 1, include 1^n0^n strings that are not in E^+. In order to correctly label examples in $E^?$ Poker must therefore learn a hypothesis that generalises at least to the 1^n0^n strings in $E^?$.

In summary, Poker's ability to automatically generate negative examples makes it possible to use a maximally general background theory that is no longer tailored to a single learning target. This ability frees the user from having to manually select a background theory and negative examples for each new learning target.

Self-Supervised ILP by detection of contradictions

The intuition behind Poker's algorithm is that, if two atoms (in the First Order Logic sense) e_1 and e_2 are accepted by the same hypothesis H (a logic program), i.e. $H \models \{e_1,e_2\}$, then to assume that e_1 is a positive and e_2 a negative example of H is a contradiction (in the informal sense). Such a contradiction can be detected in the following manner. Suppose $T = \{H_1,...,H_m\}$ is a set of hypotheses and $T \models \{e_1,e_2\}$. And suppose that we remove from T each H_i, where $H_i \models e_2$ leaving behind the set T' such that $T' \not\models e_2$. There are now two possibilities: either $T' \models e_1$, in which case there is no contradiction; or $T' \not\models e_1$, in which case there is a contradiction: e_1 and e_2 are both accepted by some subset of T, now missing from T'; therefore, e_2 is a positive, not a negative, example of the subset of T that accepts e_1.

Accordingly, Poker begins by constructing a set T of initial hypotheses that accept the labelled examples E^+. Poker can generate new unlabelled examples, added to $E^?$, by executing T as a generator. Poker then assumes that each unlabelled example $e^? \in E^?$ is negative and removes from T each hypothesis H that accepts $e^?$ in conjunction with B. If the remaining T now rejects any examples in E^+, $e^?$ is relabelled as positive and moved to E^+. The labelling process thus iteratively *specialises* T until it is consistent with E^+. The labelling process is not without error but its accuracy increases monotonically with the cardinality of $E^?$.

Contributions

We make the following contributions:

- A new setting for Self-Supervised ILP.
- A new MIL algorithm for Self-Supervised ILP, and a new MIL system, Poker, implementing the new algorithm.
- A definition of Second-Order Definite Normal Forms (SONFs), a new kind of second-order background theory sufficiently general to learn all programs in a class.
- Two SONFs for CFGs and L-System grammars in DCG notation.

- A proof that Poker's accuracy increases monotonically with the number of unlabelled examples.
- Experiments investigating the effect of automatically generated examples on Poker's learning performance.

Related Work

Self-Supervised Learning

Self-Supervised Learning is introduced in (Raina et al. 2007), albeit under the rubric of "Self-Taught Learning", where few, labelled, positive examples are used along with a larger number of unlabelled examples to learn discriminative features for "downstream" supervised learning tasks, particularly image classification. Later approaches expect only unlabelled examples but generate negative examples automatically by means of "pretext" tasks (Gui et al. 2023), such as image translations or rotations, as in Contrastive Learning (Hu et al. 2024). Choosing pretext tasks requires domain knowledge and generated examples are not used in downstream supervised learning tasks. Poker instead automatically generates both positive and negative examples and labels them consistently with labelled examples without the need of pretext tasks. Additionally Poker does not learn features for downstream learning tasks but directly uses its automatically generated examples to learn arbitrary logic programs. Logic programs learned by Poker are not classifiers but instead can be executed as either discriminators or generators as we demonstrate in the Experiments Section.

Self-Supervised Learning in ILP

We are not aware of earlier work on self-supervised ILP. The problem of ILP's over-reliance on manually defined background theories and negative examples has been tackled before.

Learning from a single example, a.k.a. one-shot learning, is comparable to self-supervised learning, in that only a single example is given that is assumed to be positive and there are no negative examples. Numerous works in the MIL literature demonstrate learning from positive-only, or single examples, e.g. (Lin et al. 2014; Dai et al. 2018). However, those systems rely on a problem-specific background theory selected manually and ad-hoc to avoid over-generalisation. In this paper instead we introduce the use of Second Order Definite Normal Forms as a new, principled way to manually select a second-order background theory for MIL.

DeepLog (Muggleton 2023) is a recent MIL system that explicitly learns from single examples, and a standard library of low-level primitives that apply to entire classes of problems, instead of a problem-specific background theory. DeePlog is limited to second-order background theories in dyadic logic (with literals of arity 2). Instead Poker's SONFs are not limited by arity, or other syntactic property.

Popper (Cropper and Morel 2021) is a system of the recent Learning from Failures setting for ILP that aims to learn logic programs without recourse to a user-provided higher-order background theory, such as required by Poker and other MIL systems. However, Popper still needs a problem-specific first-order background theory, extra-logical syntactic bias in the form of mode declarations, and negative examples. The need for negative examples is a limitation addressed in (Yang and Sergey 2025)[3]. The approach in (Yang and Sergey 2025) only applies to recursive data structures like lists or heaps. Instead Poker's algorithm learns arbitrary logic programs with recursion and invented predicates.

Name	Metarule
Identity	$\exists P, Q\ \forall x, y\colon P(x, y) \leftarrow Q(x, y)$
Inverse	$\exists P, Q\ \forall x, y\colon P(x, y) \leftarrow Q(y, x)$
Precon	$\exists P, Q\ \forall x, y\colon P(x, y) \leftarrow Q(x), R(x, y)$
Chain	$\exists P, Q, R\ \forall x, y, z\colon P(x, y) \leftarrow Q(x, z), R(z, y)$

Table 2: Metarules commonly found in the MIL literature.

Meta$_{Abd}$ (Dai and Muggleton 2021) is a neuro-symbolic MIL system that trains a deep neural net in tandem with a MIL system modified for abductive learning, on unlabelled, sub-symbolic examples. Meta$_{Abd}$ relies on a manually defined, problem-specific first-order background theory in the form of definitions of adbucible predicates. Instead Poker does not require a problem-specific background theory.

Framework

In the following, we assume familiarity with the logic programming terminology introduced e.g. in (Lloyd 1987).

Meta-Interpretive Learning

Meta-Interpretive Learning(Muggleton et al. 2014; Muggleton and Lin 2015) (MIL) is a setting for ILP where first-order logic theories are learned by SLD-Refutation of ground atoms E^+, E^- given as positive and negative training examples, respectively. E^+, E^- are resolved with a higher-order background theory that includes both first- and secod-order definite clauses, B and $\mathcal{M}$, respectively, the latter known as "metarules"(Muggleton and Lin 2015). A first SLD-Refutation proof of E^+ produces an initial hypothesis that is then specialised by a second SLD-Refutation step, this time proving E^- to identify, and discard, over-general hypotheses. The "metarules", are second-order definite clauses with second-order variables existentially quantified over the set of predicate symbols, which can optionally include automatically generated invented predicate symbols, and with first-order variables existentially or universally quantified over the set of first-order terms, in B, E^+. During SLD-Refutation of E^+ metarules' second-order variables are unified to predicate symbols, constructing the clauses of a first-order hypothesis, H. Table 2 lists metarules commonly found in the MIL literature.

A Self-Supervised Learning setting for ILP

In the Normal Setting for ILP(Nienhuys-Cheng and de Wolf 1997), we are given a first-order background theory B, positive and negative examples E^+, E^-, and must find a *correct hypothesis*, a logic program H, such that $B \cup H \models E^+$ and $\forall e^- \in E^- : B \cup H \not\models e^-$. We now formally define

[3]The authors consider the need for negative examples a limitation of "classic" or "standard" ILP by which they seem to refer specifically to Popper; that is an over-generalisation.

our new, Self-Supervised Learning setting for ILP, *SS-ILP*, where there are no negative examples, only labelled positive, or unlabelled examples, and B is replaced by a higher-order background theory that is maximally general. We model our definition after the definition of the Normal Setting for ILP formalised in (Nienhuys-Cheng and de Wolf 1997).

Definition 1 ***Inductive Logic Programming: Self-Supervised Setting***

- ***Not Given:***
  $\mathbf{\Theta} = P/n$*, a predicate* P *of arity* n *that we call the* target predicate*. Let* B_H *be the Herbrand Base of* Θ*. Recall that the Herbrand Base of a predicate* P/n *is the set of all ground atoms* P/n *with variables substituted for ground terms in the Domain of Discourse,* $\mathcal{U}$*, (or a new constant* α *if there are no ground terms in* $\mathcal{U}$*) (Nienhuys-Cheng and de Wolf 1997).*
  $\mathbf{I}$*, a Herbrand interpretation over* B_H *that we call the* intended interpretation*. Let* $I^+, I^- \subseteq B_H$ *be the sets of atoms in* B_H *that are true and false under* I*, respectively, such that* $I^+ \cup I^- = B_H$ *and* $I^+ \cap I^- = \emptyset$*.*
- ***Given:***
  $\mathbf{E} = \{\mathbf{E}^+, \mathbf{E}^?\} \subseteq \mathbf{B_H}$*, a finite set of ground atoms* P/n*. Let* $E^+ \subseteq I^+$*, and* $E^? \subseteq I^+ \cup I^-$*. Either* E^+ *or* $E^?$ *may be empty, but not both. We say that examples in* E^+ *are* labelled *and examples in* $E^?$ *are* unlabelled*, denoting an assumption that* E^+ *are all true under* I *and uncertainty of which atoms in* $E^?$ *are true under* I*.*
  $\mathcal{T} = \mathbf{B} \cup \mathcal{M}$*, a higher-order background theory, where* $B, \mathcal{M}$ *are finite sets of first- and second-order definite clauses, respectively, and such that* $\mathcal{M} \cup B \cup E \models I^+ \cup I^-$*, i.e.* $\mathcal{T}$ *is maximally general with respect to* Θ*.*
- ***Find:***
  $\mathbf{H}$*, a logic program that we call a* hypothesis*, a finite set of first-order definite program clauses such that:*
  1. $B \cup \mathcal{M} \models H$
  2. $B \cup H \models I^+$
  3. $\forall a \in I^- : B \cup H \not\models a$

  $\mathbf{L} = \mathbf{L}^+ \cup \mathbf{L}^-$*, a set of two finite sets of atoms that we call a* labelling*, such that:*
  1. $L^+ \subseteq I^+$
  2. $L^- \subseteq I^-$

  If H, L^+, L^- *satisfy the above criteria we call* H *a* correct hypothesis *and* L *a* correct labelling *under the Self-Supervised setting for ILP.*

Poker: an algorithm for Self-Supervised ILP

Poker assumes that the higher-order background theory, $\mathcal{T}$, is a *Second-Order Definite Normal Form*, abbreviated *SONF* for ease of pronounciation. Informally, a SONF is a set of *constrained metarules* that generalise the set of first-order logic program definitions of *all possible interpretations* I over the Herbrand Base B_H of Θ. A formal definition of constrained metarules and SONFs, follows.

First, we define constrained metarules extending the definition of metarules in (Muggleton and Lin 2015) with a set of constraints on the substitution of second-order variables.

Definition 2 (Constrained Metarules) *Let* $B, \mathcal{M}, E^+$ *be as in Definition 1, and let* $\mathcal{P}$ *be the set of all predicate symbols in* B, E^+*, and 0 or more automatically generated invented predicate symbols. Let* $S = \{S_E, S_B, S_I, S_\epsilon\}$ *be a set of disjoint subsets of* $\mathcal{P}$*, the sets of symbols in* E^+*,* B*, invented predicates, and the set* $\{\epsilon\} \subseteq S_B$*, respectively, and let* P, Q *be two distinct second-order variables, and* $P_1, ..., P_n$ *a list of* n *such, in a metarule* $M \in \mathcal{M}$*. A* metarule constraint *on* M *is one of the following atomic formulae, with the associated interpretation:*

- $P = Q$*:* P, Q must *be instanted to the same symbol in* $\mathcal{P}$*.*
- $P \neq Q$*:* P, Q must not *be instantiated to the same symbol in* $\mathcal{P}$*.*
- $target(P_1, ..., P_n)$*: each* P_i *must be instantiated to a symbol in* S_E*.*
- $invented(P_1, ..., P_n)$*: each* P_i *must be instiantiated to a symbol in* S_I*.*
- $background(P_1, ..., P_n)$*: each* P_i *must be instantiated to a symbol in* S_B*.*
- $empty(P_1, ..., P_n)$ *:* P *must be instantiated to* ϵ*.*
- $invented(P, Q) \rightarrow P \neq Q$*: If* P, Q *are instantiated to symbols in* S_I*,* P *must not be the same symbol as* Q*.*
- $invented(P, Q) \rightarrow P \geq Q$*: If* P, Q *are instantiated to symbols in* S_I*,* P *must be above than or equal to* Q *in the standard alphabetic ordering.*
- $invented(P, Q) \rightarrow P \leq Q$*: If* P, Q *are instantiated to symbols in* S_I*,* P *must be below than or equal to* Q *in the standard alphabetic ordering.*

If C^1_M, C^2_M *are two constraints on metarule* M*, then* $C^1_M \wedge C^2_M$ *is a constraint on* M*, interpreted as "*C^1_M *and* C^2_M *must both be true".*

If C^1_M, C^2_M *are two constraints on metarule* M*, then* $C^1_M \vee C^2_M$ *is a constraint on* M*, interpreted as "one or both of* C^1_M*,* C^2_M *must be true".*

If C_M *is a constraint on metarule* M*, then* $\neg C_M$ *is a constraint on* M*, interpreted as the opposite of* C_M*. E.g.* $\neg target(P)$ *is interpreted as "*P *must not be instantiated to a symbol in* S_E*".*

A constrained metarule *is a formula* $M : C_M$*, where* $M \in \mathcal{M}$ *and* C_M *is a constraint on* M*.*

Constraints on a metarule M *are satisfied by a first order definite clause* C *if, and only if,* C *is an instance of* M *with second-order variables instantiated according to the constraints on* M*.*

Metarule constraints are primarily meant to control recursion, particularly to eliminate unnecessary recursion, including left recursion, and to reduce redundancy in the construction of Poker's initial hypotheses. Metarule constraints can thus improve Poker's efficiency, and in that sense serve much the same function as heuristics exploiting the common structure of problems in a class, as e.g. in Planning (Geffner and Bonet 2013) and SAT-Solving (Biere et al. 2021).

We now formalise the definition of SONFs.

Definition 3 (Second-Order Definite Normal Form) *Let* Θ*,* B_H*,* I*,* I^+*,* I^-*, and* $\mathcal{M}$ *be as in Definition 1, with the additional stipulation that* $\mathcal{M}$ *is a set of* constrained *metarules as in Definition 2. Let* $\mathcal{I}$ *be the set of all Herbrand*

Chomsky-Greibach Second-Order Definite Normal Form
(Identity) $P(x,y) \leftarrow Q(x,y)$: $target(P) \wedge background(Q) \vee empty(Q)$
(Chain) $P(x,y) \leftarrow Q(x,z), R(z,y)$: $P \neq Q \wedge (target(P) \vee invented(P)) \wedge not(target(Q))$ $\wedge\ not(empty(Q,R)) \wedge invented(P,Q) \rightarrow P \neq Q$
(Tri-Chain) $P(x,y) \leftarrow Q(x,z), R(z,u), S(u,y)$: $P \neq Q \wedge Q \neq R \wedge R \neq S$ $\wedge\ (target(P) \vee invented(P)\ ) \wedge not(target(Q))$ $\wedge\ not(empty(Q,R,S)) \wedge invented(P,Q) \rightarrow P \neq Q$

Table 3: Chomsky-Greibach SONF for CFL DCGs.

interpretations I over B_H and, for each $I \in \mathcal{I}$ let Π_I be the set of all first-order definite programs $\Sigma_1, ..., \Sigma_n$, such that $\forall \Sigma_i \in \Pi_I : \Sigma_i \models I^+$ and $\forall \Sigma_i \in \Pi_I, \forall e^- \in I^- : \Sigma_i \not\models e^-$. Let Π^* be the set of all Π_I for all I over B_H.

$\mathcal{M}$ is a Second-order Definite Normal Form, abbreviated to SONF*, for Θ, if, and only if, $\mathcal{M} \models \Pi'$ for some subset Π' of Π^*. $\mathcal{M}$ is a* strong SONF *for Θ if $\Pi' = \Pi^*$. $\mathcal{M}$ is a* weak SONF *for Θ if $\Pi' \subset \Pi^*$ (i.e. Π' is a subset of, but not equal to, Π^*).*

The purpose of SONFs is to replace the use of problem-specific sets of metarules, previously common in MIL practice, with a maximally general second-order background theory sufficient to learn any logic program definition of a target predicate Θ, for any interpretation I of Θ.

Two Normal Forms We illustrate the concept of Second Order Definite Normal Forms with two SONFs used in the Experiments Section: Chomsky-Greibach Normal Form (C-GNF) used to learn CFGs, and Lindenmayer Normal Form (LNF) used to learn L-System grammars. C-GNF is listed in Table 3. We list LNF and prove the completeness of C-GNF for CFLs, and LNF for L-systems, in the Appendix.

Derivation of a SONF is a process of abstraction that we do not currently know how to automate. Both C-GNF and LNF are therefore derived manually by the author. C-GNF is derived by an encoding of the rules of Chomsky and Greibach Normal Forms in our Second-Order Normal Form notation. LNF is derived from the common structure observed in manual definitions of L-System grammars as DCGs, coded by the author. Metarule constraints are obtained from experimentation to improve learning efficiency.

A guide for the construction of Second-Order Normal Forms is beyond the scope of the paper. However, we include Table 4 as an illustration of the derivation of C-GNF metarules (without constraints) from Chomsky Normal Form.

Poker's algorithm We typeset Poker's algorithm for SS-ILP as Algorithm 1. We state our main theoretical result as Theorem 1. We include a proof of the Theorem in the Appendix.

Theorem 1 (Hypothesis Correctness) *The probability that Algorithm 1 returns a correct hypothesis increases monotonically with the number of unlabelled examples.*

Algorithm 1 Poker: SS-ILP by detection of contradictions

Input: $E^+, E^?, B, \mathcal{M}$ as in Definition 1; $\mathcal{M}$ is a SONF.
V, finite set of invented predicate symbols.
Integer $k \geq 0$, max. number of automatically generated examples.
Integer $l \geq 0$, max. number of clauses in initial hypotheses H.
Initialise: Hypothesis $T \leftarrow \emptyset$. Labelling $L = \{E^+, E^- \leftarrow \emptyset\}$.

procedure GENERALISE($B, \mathcal{M}, E^+$)
  $T \leftarrow \{H_{|H| \leq l} : \mathcal{M} \cup B \cup V \models H \wedge B \cup H \models e^+ \in E^+\}$
end procedure
procedure GENERATE($B, T, E^?$)
  $T' \leftarrow \bigcup T$ ▷ This step is optional. Alternatively $T' \leftarrow T$
  $E^- \leftarrow E^? \cup \{e_{i=1}^k :\ B \cup T' \models e_i\}$
end procedure
procedure LABEL(B, T, E^+, E^-)
  for all $e \in E^-$ **do**
    $T' \leftarrow T \setminus \{H :\ H \in T \wedge B \cup H \models e\}$
    if $B \cup T' \models E^+$ **then**
      $T \leftarrow T'$
    else
      $E^- \leftarrow E^- \setminus \{e\}$
      $E^+ \leftarrow E^+ \cup \{e\}$
    end if
  end for
end procedure
$T \leftarrow \bigcup T$ ▷ This step is optional.
$T \leftarrow T \setminus \{C \in T : T \models C\}$ ▷ This step is optional.
Return $T, L = \{E^+, E^-\}$

Implementation

We implement Algorithm 1 in a new system, also called Poker, written in Prolog. Poker extends the Top Program Construction algorithm (Patsantzis and Muggleton 2021) (TPC). Procedure GENERALISE in Algorithm 1, taken from TPC, is implemented in Poker by a call to Vanilla, an inductive second-order Prolog meta-interpreter for MIL, introduced in (Trewern, Patsantzis, and Tamaddoni-Nezhad 2024). Vanilla is developed with SWI-Prolog (Wielemaker et al. 2012) and uses SWI-Prolog's tabled execution, (Tamaki and Sato 1986), a.k.a. SLG-Resolution, to control recursion and improve performance.

Experiments

Poker learns from three kinds of examples: user-given a) labelled, and b) unlabelled, examples, and, c) automatically generated examples. In preliminary experiments we find that the effect of (b), unlabelled examples, on Pokers' learning performance depends on the generality of the unlabelled examples. This merits more thorough investigation. We limit the current investigation on the effect of automaticaly generated examples on Poker's performance, formalised in the following research question:

Research Question 1 *What is the effect of automatically generated examples on Poker's learning performance?*

We address this question with two sets of experiments, learning grammars for two sets of languages: Context-Free Languages (CFLs); and L-Systems. L-System grammars are meant to be run as *generators*, producing strings interpreted

Chomsky Normal Form	DCG	Definite Clauses	C-GNF metarules
$s \rightarrow \epsilon$	$s \rightarrow empty.$	$s(x,y) \leftarrow empty(x,y).$	**Identity** $P(x,y) \leftarrow Q(x,y)$
$N_0 \rightarrow N_1 N_2$	$n_0 \rightarrow n_1, n_2.$	$n_0(x,y) \leftarrow n_1(x,z), n_2(z,y).$	**Chain** $P(x,y) \leftarrow Q(x,z), R(z,y)$
$N_i \rightarrow t$	$n_i \rightarrow t.$	$n_i(x,y) \leftarrow t(x,y).$	**Identity** $P(x,y) \leftarrow Q(x,y)$
	$empty \rightarrow [\,].$	$empty(x,x)$	None (included in B)
	$t \rightarrow [t].$	$t([t\|x],x)$	None (included in B)

Table 4: Chomsky Normal Form mapping to DCGs and C-GNF constrained metarules. N_i, n_i: non-terminals; t : terminals.

Background theories			
Low Uncertainty (CFLs)		**Mod. Uncertainty (L-Systems)**	
DCG Rule	**Definite Clause**	**DCG Rule**	**Definite Clause**
$one \rightarrow [1]$	$one([1\|X],X)$	$plus \rightarrow [+]$	$plus([+\|X],X)$
$zero \rightarrow [0]$	$zero([0\|X],X)$	$min \rightarrow [-]$	$min([-\|X],X)$
$empty \rightarrow [\,]$	$empty(X,X)$	$f \rightarrow [f]$	$f([f\|X],X)$
		$g \rightarrow [g]$	$g([g\|X],X)$
		$x \rightarrow [x]$	$x([x\|X],X)$
		$y \rightarrow [y]$	$y([y\|X],X)$

Table 5: Background theories used in experiments.

as movement and drawing commands sent to a (now usually simulated) robot. Accordingly, we evaluate learned L-System grammars as *generators*.

Baselines There are, to our knowledge, no ILP systems that generate new examples during learning, other than Poker. We consider comparing Poker to ILP systems Aleph (Srinivasan 2001) and Popper (Cropper and Morel 2021). In preliminary experiments both perform poorly without negative examples. In private communication Popper's authors confirm Popper should not be expected to learn CFGs without negative examples. We alight on the state-of-the-art MIL system Vanilla-Louise (Trewern, Patsantzis, and Tamaddoni-Nezhad 2024) (hereby denoted as Louise, for brevity), as a simple baseline. Louise is also based on Vanilla and accepts constrained metarules, simplifying comparison.

Experimental protocol

In each set of experiments, we generate positive training examples and positive and negative testing examples from a manual definition of the learning target (the grammar to be learned) as a DCG.

We train Poker and Louise on increasing numbers, $l > 0$, of positive labelled examples. We vary the number of Poker's automatically generated examples, $k \geq 0$, and record, a) for CFLs, the True Positive Rate (TPR) and True Negative Rate (TNR) of i) hypotheses learned by both systems and ii) labellings returned by Poker; and b) for L-Systems i) the accuracy of learned hypotheses as generators ("Generative Accuracy") and ii) the cardinality of learned hypotheses, i.e. their number of clauses ("Hypothesis Size"). Testing examples are used to evaluate CFL hypotheses as acceptors: for each hypothesis, H, we test how many positive testing examples are accepted, and how many negative ones rejected, by H. We do not use testing examples to evaluate labellings or L-System hypotheses as generators. Instead, we count the number of examples labelled positive or negative, in a labelling, or generated by a hypothesis executed as a generator, that are accepted or rejected, respectively for positive and negative examples, by the manually defined learning target.

In CFL experiments, we measure TPR and TNR to avoid confusing measurements when $k = 0$.

Experiment results are listed in Figures 1 and 2 for L-Systems and CFLs, respectively. In each Figure, the x-axis plots the number of labelled positive examples, the y-axis plots mean TPR, TNR, Generative Accuracy or Hypothesis Size, while each line plots the change in that metric with each increment of k, the number of automatically generated examples. For Louise, $k = 0$ always. Error bars are standard error over 100 randomly sampled sets of training and testing examples in each experiment.

Experiment 1: L-Systems

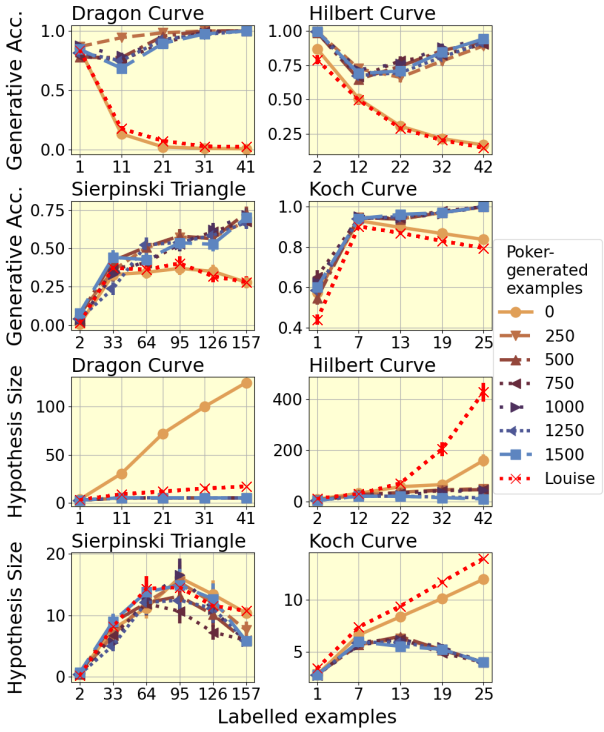

Figure 1: Learning L-Systems: generative accuracy.

We train Poker and Louise on example strings of the L-Systems for the Dragon Curve, Hilbert Curve, Koch Curve, and Sierpinski Triangle fractals, taken from (Prusinkiewicz

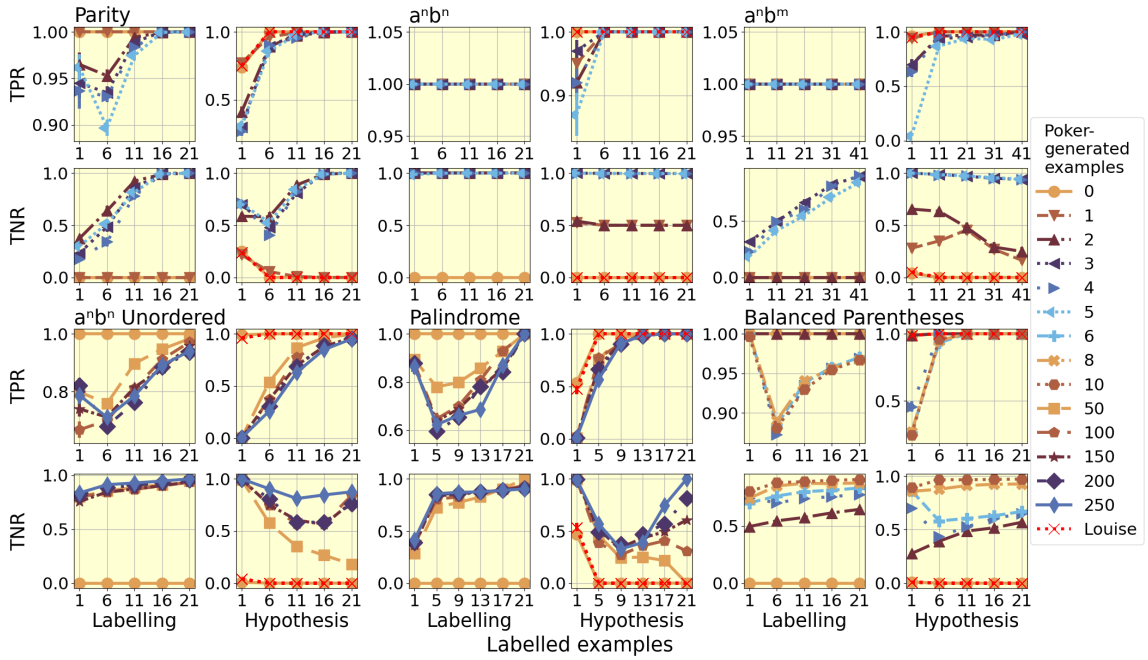

Figure 2: Learning CFGs. TPR: True Positive Rate. TNR: True Negative Rate.

and Lindenmayer 1996). The first-order background theory consists of a set of symbols, common for all L-Systems, but not necessarily with the same meaning for each, e.g. the symbol f is a *variable* (nonterminal) in Dragon and Koch Curve, but a *constant* (terminal) in Hilbert Curve. This serves as a test of a system's robustness to noise. The second-order background theory is LNF, for all L-systems.

Results are shown in Figure 1. We observe that Poker's Generative Accuracy increases, and Hypothesis Size decreases, with the number of automatically generated examples. Louise's Generative Accuracy decreases and Hypothesis Size increases with the number of labelled examples, indicating increasing over-generalisation. This indicates that increasing numbers of automatically generated examples improve Poker's performance and avoid over-generalisation in language generation tasks.

Experiment 2: Binary CFLs

We train Poker and Louise on examples of six CFLs: a) The language of bit-strings with an even number of 1's (even parity), b) $\{a^n b^n : n > 0\}$, c) $\{w \in \{a,b\}^* : n_a(w) = n_b(w), n \geq 0\}$ (n a's and n b's in any order), d) $\{a^n b^m | n \geq m \geq 0\}$, e) the language of palindromic bit-strings and f) the language of balanced parentheses. Symbols $a, b,)$, and $($ are suitably replaced by 1 and 0, so that all experiments can share the same first- and second-order background theory, consisting of the alphabet $\{1, 0, \epsilon\}$, in DCG form, as listed in Table 5, and C-GNF[4], respectively.

Results are listed in Figure 2. When $k = 0$, Poker's TPR is maximal while its TNR is minimal, because there are no negative examples. When $k > 0$ both TPR and TNR increase with k, until they are both maximised with the highest increments of k. From this we conclude that when k is low, Poker's labelling and hypothesis over-generalise, and when k is sufficiently large, Poker learns a correct labelling and hypothesis. Thus, Poker's performance improves with the number of automatically generated examples. Louise over-generalises consistently over all experiments.

[4]Note that the Tri-Chain metarule is only used with Palindrome.

Conclusions and Future Work

ILP systems can generalise well from few examples given a problem-specific background theory and negative examples, both manually selected with domain knowledge. We have endeavoured to address this onerous requirment with a new formal setting for Self-Supervised ILP, SS-ILP, and presented Poker, a new MIL system that learns in the new setting. Poker's algorithm learns from labelled and unlabelled examples, and automatically generates new positive and negative examples during learning. Instead of a problem-specific background theory Poker only needs a maximally general, Second Order Normal Form (SONF). We have presented two SONFs, for Context-Free and L-System Definite Clause Grammars. We have given a theoretical proof and empirical evidence that Poker's accuracy improves with increasing numbers of unlabelled examples, and showed how a baseline MIL system lacking Poker's ability to automatically generate negative examples over-generalises.

Our theoretical results can be extended with further proofs of the correctness of Poker's algorithm, including with respect to varying amounts of labelled and unlabelled examples; and of its computational efficiency. Our empirical results are restricted to grammar learning. Future work should investigate Poker's application to more diverse domains.

Acknowledgments

We thank Em. Professor Stephen Muggleton and Drs Lun Ai and Céline Hocquette for their kind feedback on this work.

References

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. ISBN 978-1-64368-160-3.

Cropper, A.; and Morel, R. 2021. Learning programs by learning from failures. *Machine Learning*.

Dai, W.-Z.; and Muggleton, S. 2021. Abductive Knowledge Induction from Raw Data. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 1845–1851. IJCAI Organization. Main Track.

Dai, W.-Z.; Muggleton, S.; Wen, J.; Tamaddoni-Nezhad, A.; and Zhou, Z.-H. 2018. Logical Vision: One-Shot Meta-Interpretive Learning from Real Images. In Lachiche, N.; and Vrain, C., eds., *Inductive Logic Programming*, 46–62. Cham: Springer International Publishing. ISBN 978-3-319-78090-0.

Flener, P.; and Yilmaz, S. 1999. Inductive synthesis of recursive logic programs: achievements and prospects. *The Journal of Logic Programming*, 41(2): 141 – 195.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning: Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers, 1st edition. ISBN 1608459691.

Gui, J.; Chen, T.; Zhang, J.; Cao, Q.; Sun, Z.; Luo, H.; and Tao, D. 2023. A Survey on Self-Supervised Learning: Algorithms, Applications, and Future Trends. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46: 9052–9071.

Hu, H.; Wang, X.; Zhang, Y.; Chen, Q.; and Guan, Q. 2024. A comprehensive survey on contrastive learning. *Neurocomputing*, 610: 128645.

Lin, D.; Dechter, E.; Ellis, K.; Tenenbaum, J.; Muggleton, S.; and Dwight, M. 2014. Bias reformulation for one-shot function induction. In *Proceedings of the 23rd European Conference on Artificial Intelligence*, 525–530. ISBN 9781614994190.

Lloyd, J. W. 1987. *Foundations of Logic Programming*. Symbolic Computation. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-83191-1.

Muggleton, S. 1991. Inductive Logic Programming. *New Generation Computing*, 8(4): 295–318.

Muggleton, S.; and Lin, D. 2015. Meta-Interpretive Learning of Higher-Order Dyadic Datalog : Predicate Invention Revisited. *Machine Learning*, 100(1): 49–73.

Muggleton, S. H. 2023. Hypothesizing an algorithm from one example: the role of specificity. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2251): 20220046.

Muggleton, S. H.; Lin, D.; Pahlavi, N.; and Tamaddoni-Nezhad, A. 2014. Meta-interpretive learning: Application to grammatical inference. *Machine Learning*, 94(1): 25–49.

Nienhuys-Cheng, S.-H.; and de Wolf, R. 1997. *Foundations of Inductive Logic programming*. Berlin: Springer-Verlag.

Patsantzis, S.; and Muggleton, S. H. 2021. Top program construction and reduction for polynomial time Meta-Interpretive learning. *Machine Learning*.

Pereira, F.; and Shieber, S. M. 1987. *Prolog and Natural-Language Analysis*, volume 10. Center for the Study of Language and Information.

Prusinkiewicz, P.; and Lindenmayer, A. 1996. *The algorithmic beauty of plants*. Berlin, Heidelberg: Springer-Verlag. ISBN 0387946764.

Raina, R.; Battle, A.; Lee, H.; Packer, B.; and Ng, A. 2007. Self-taught learning: transfer learning from unlabeled data. In *International Conference on Machine Learning*.

Srinivasan, A. 2001. Aleph version 4 and above.

Tamaki, H.; and Sato, T. 1986. OLD resolution with tabulation. In Shapiro, E., ed., *Third International Conference on Logic Programming*, 84–98. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-39831-8.

Trewern, J.; Patsantzis, S.; and Tamaddoni-Nezhad, A. 2024. Meta-Interpretive Learning as Second-Order Resolution. In Muggleton, S. H.; and Tamaddoni-Nezhad, A., eds., *Proceedings of the Fourth International Joint Conferences on Learning and Reasoning IJCLR 24. In print.*

Wielemaker, J.; Schrijvers, T.; Triska, M.; and Lager, T. 2012. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2): 67–96.

Yang, Z.; and Sergey, I. 2025. Inductive Synthesis of Inductive Heap Predicates – Extended Version. arXiv:2502.14478.