Differentiable Inductive Logic Programming for Structured Examples

(published: AAAI 2021)

Hikaru Shindo, Masaaki Nishino, Akihiro Yamamoto

Presented by:
Jesse Jing, Divyagna Bavikadi, Kaustuv Mukherji
For a student led discussion on:
Neuro-symbolic Reasoning
At ***Arizona State University***
Oct 24, 2022

Logic Programming Basics

$$Language\ \mathcal{L} = (\mathcal{P}, \mathcal{F}, \mathcal{A}, \mathcal{V})$$

$\mathcal{P}$: Set of Predicates (p/n)

$\mathcal{F}$: Set of Function Symbols (f/n)

$\mathcal{A}$: Set of Constants

$\mathcal{V}$: Set of Variables

Logic Programming Basics

A $term$ (t) could be a $constant$, a $variable$, or $f(t_1, t_2, \ldots, t_n)$.

An $atom$ is a formula $p(t_1, t_2, \ldots, t_n)$.

A $ground\ atom$ has no $variables$.

A $clause$ is a $disjunction$ $(\vee)$ of $literals$.

$V(C)$: Set of variables in clause C.

Logic Programming Basics

θ : Substitution operation $[x_i \leftarrow t_i]$

θ applied to head of a rule A, is written as $A\theta$.

A $unifier$ for a set $\{A_1, A_2, \ldots, A_n\}$ is a substitution θ such that, $A_1\theta = A_2\theta = \cdots = A_n\theta$, written as: $\theta = \sigma(\{A_1, A_2, \ldots, A_n\})$

σ : Unification function

$\underline{\bar{\sigma}(\{A_1, A_2, \ldots, A_n\})}$ $= \top$ if set is unifiable, else $\perp$

Decision function

The ILP problem

Given:

$$\mathcal{Q} = (\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}, \mathcal{L})$$

Find a set of definite clauses (one atom in the head) $\mathcal{H} \subseteq \mathcal{L}$, such that,

$$\forall A \in \mathcal{E}^+ \; \mathcal{H} \cup \mathcal{B} \models A.$$

$$\forall A \in \mathcal{E}^- \; \mathcal{H} \cup \mathcal{B} \not\models A.$$

Why ∂ILP?

- Unlike ILP, robust to noise and training data[1]

- Formulates ILP problems as an optimization problem, which can be differentiated and solved much like traditional machine learning methods. Hence, it's easy to combine these approaches to neural systems.

[1]Evans, R., & Grefenstette, E. (2018). Learning explanatory rules from noisy data. Journal of Artificial Intelligence Research, 61, 1-64.

Drawbacks of previous ∂ILP systems

- No function symbols are allowed.
- The arity of predicates must be less than 2.
- The number of atoms in the clause body must not exceed 2.
- Every program must be comprised of pairs of rules for each predicate.

Drawbacks of previous ∂ILP systems

<u>Thus,</u>

- Unsuitable for complex structured data, such as sequences or trees.

- Unsuitable for complex programs that are comprised of several clauses for a predicate.

Today’s paper -

Differentiable Inductive Logic Programming for Structured Examples

Hikaru Shindo, Masaaki Nishino, Akihiro Yamamoto

What are Structured Examples?

- Examples of data which has a ‘structure’ to it.

$$[a, b, c], [d], [a, b, c, d]$$

- Sequences and Trees

$$[1, 2, 4, 8, \dots]$$

- Data with function symbols

$$f^4(x) * f^8(x) = f^{12}(x)$$

Challenges

- Consider a large number of clauses.

- Number of possible ground atoms are infinite.

$$c_1, c_2, \ldots, c_n, f(c_1), g(c_1), h(c_1, c_2) \ldots \ldots \ldots$$

- Keep memory and computation costs manageable.

What should the result look like?

$$\mathcal{E}^+ = \{p(a,a), p(b,b), p(b,c), p(c,b)\}$$
$$\mathcal{E}^- = \{p(a,b)\}$$

$$\mathcal{B} = \{q(b,c), q(c,b)\}$$

What should the result look like?

$\mathcal{E}^+ = \{p(a,a), p(b,b), p(b,c), p(c,b)\}$
$\mathcal{E}^- = \{p(a,b)\}$

$\mathcal{B} = \{q(b,c), q(c,b)\}$

One possible solution:
$p(x,x)$

What should the result look like?

$\mathcal{E}^+ = \{p(a,a), p(b,b), p(b,c), p(c,b)\}$
$\mathcal{E}^- = \{p(a,b)\}$

$\mathcal{B} = \{q(b,c), q(c,b)\}$

One possible solution:
$p(x,x)$
$p(x,y) \leftarrow q(x,y)$

The Building Blocks

1. Get Clauses
2. Determine Ground Atoms for clauses
3. Introduce weights for clauses
4. Do Inference: Minimize error (Differentiable!)

Clause Search with Refinement

- <u>Objective:</u> Find promising clauses that entail many positive examples but few negative examples.

  ALL NONE

- <u>Start:</u> General/Strong clauses
  Might entail many examples, including negative examples.

- <u>Refine:</u> Iteratively specify/weaken clauses.
  Now, they may entail lesser number of $\mathcal{E}^+$, but will also entail lesser $\mathcal{E}^-$.

The Refinement Operator ($\rho_{\mathcal{L}}$)

$$\rho_{\mathcal{L}}(C) = \rho_{\mathcal{L}}^{func}(C) \cup \rho_{\mathcal{L}}^{subs}(C) \cup \rho_{\mathcal{L}}^{rep}(C) \cup \rho_{\mathcal{L}}^{add}(C)$$

The Refinement Operator ($\rho_{\mathcal{L}}$)

$$\rho_{\mathcal{L}}(C) = \rho_{\mathcal{L}}^{func}(C) \cup \rho_{\mathcal{L}}^{subs}(C) \cup \rho_{\mathcal{L}}^{rep}(C) \cup \rho_{\mathcal{L}}^{add}(C)$$

<u>Start:</u> $p(x, y)$

$\mathcal{V} = \{x, y, z\}$

$\mathcal{A} = \{a, b\}$

- Application of function symbols

$$\rho_{\mathcal{L}}^{\{func\}}\big(p(x, y)\big) = p(x, f(z))$$

The Refinement Operator ($\rho_{\mathcal{L}}$)

$$\rho_{\mathcal{L}}(C) = \rho_{\mathcal{L}}^{func}(C) \cup \rho_{\mathcal{L}}^{subs}(C) \cup \rho_{\mathcal{L}}^{rep}(C) \cup \rho_{\mathcal{L}}^{add}(C)$$

<u>Start:</u> $p(x, y)$

$\mathcal{V} = \{x, y, z\}$

$\mathcal{A} = \{a, b\}$

- Application of function symbols

$$\rho_{\mathcal{L}}^{\{func\}}\big(p(x, y)\big) = p(x, f(z))$$

- Substitution of constants

$$\rho_{\mathcal{L}}^{\{subs\}}\big(p(x, y)\big) = p(x, a)$$

The Refinement Operator ($\rho_{\mathcal{L}}$)

$$\rho_{\mathcal{L}}(C) = \rho_{\mathcal{L}}^{func}(C) \cup \rho_{\mathcal{L}}^{subs}(C) \cup \rho_{\mathcal{L}}^{rep}(C) \cup \rho_{\mathcal{L}}^{add}(C)$$

<u>Start:</u> $p(x, y)$

$\mathcal{V} = \{x, y, z\}$

$\mathcal{A} = \{a, b\}$

- Replacement of variables

$$\rho_{\mathcal{L}}^{\{rep\}}\big(p(x, y)\big) = p(x, z)$$

The Refinement Operator ($\rho_{\mathcal{L}}$)

$$\rho_{\mathcal{L}}(C) = \rho_{\mathcal{L}}^{func}(C) \cup \rho_{\mathcal{L}}^{subs}(C) \cup \rho_{\mathcal{L}}^{rep}(C) \cup \rho_{\mathcal{L}}^{add}(C)$$

<u>Start:</u> $p(x, y)$

$\mathcal{V} = \{x, y, z\}$

$\mathcal{A} = \{a, b\}$

- Replacement of variables

$$\rho_{\mathcal{L}}^{\{rep\}}\big(p(x, y)\big) = p(x, z)$$

- Addition of atoms

$$\rho_{\mathcal{L}}^{\{add\}}\big(p(x, y)\big) = p(x, y) \leftarrow q(x, y)$$

What should the result look like?

$$\mathcal{E}^+ = \{p(a,a), p(b,b), p(b,c), p(c,b)\}$$
$$\mathcal{E}^- = \{p(a,b)\}$$

$$\mathcal{B} = \{q(b,c), q(c,b)\}$$

Refinement with beam search

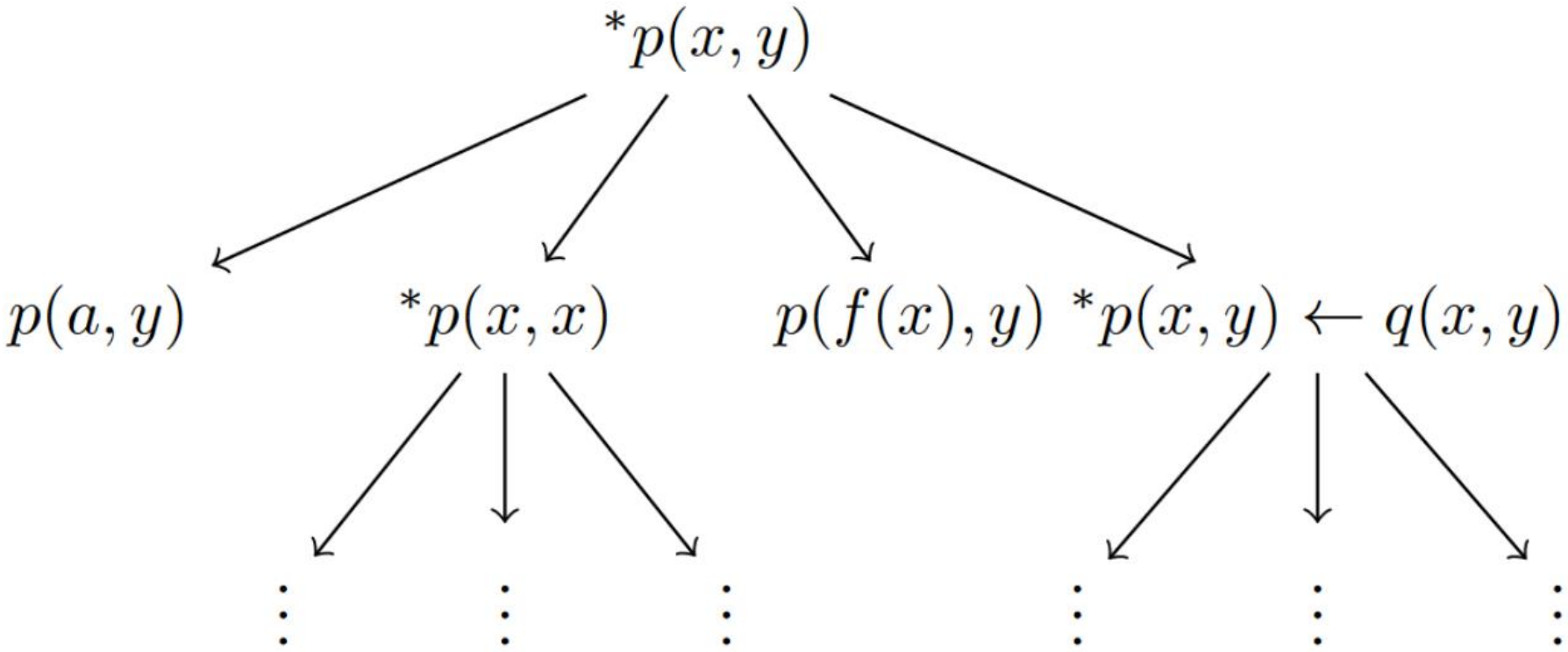
$$\mathcal{E}^+ = \{p(a,a), p(b,b), p(b,c), p(c,b)\}$$
$$\mathcal{E}^- = \{p(a,b)\}$$

$$\mathcal{B} = \{q(b,c), q(c,b)\}$$

$$C_0 = \{p(x,y\}$$

$$N_{beam} = 2$$
$$T_{beam} = 2$$

Refinement with beam search



$$eval(R, \mathcal{Q}) = |\{E \mid E \in \mathcal{E}^+ \wedge \mathcal{B} \cup \{R\} \models E\}|$$

Adaptive Fact Enumeration

Soft Program Composition

Putting it all together – solving the ILP!

Experiments

- Enumeration Algorithm
- Beam Searching Approach for Clause Generation
- Memory and Computational efficiency
- Does the method efficiently learn from noisy and structured examples?
- Used standard 5 ILP tasks with structured examples

Experiments

V = {x, y, z, v, w} $\gamma = 10^{-5}$

- Member:

$$\mathcal{P} = \{mem/2\} \quad \mathcal{F} = \{f/2\}$$

$$\mathcal{C}_0 = \{mem(x, y)\} \qquad \mathcal{A} = \{a, b, c, *\}$$

$$(N_{beam}, T_{beam}) = (3, 3) \qquad m = 2 \qquad T = 4$$

$$\mathcal{E}^+ = \{mem(a, [a, c]), mem(a, [b, a]), \ldots\},$$

$$\mathcal{E}^- = \{mem(c, [b, a]), mem(c, [a]), \ldots\},$$

$$\mathcal{B} = \{mem(a, [a]), mem(b, [b]), mem(c, [c])\}.$$

Experiments

- Plus:

$$\mathcal{P} = \{plus/3\}, \mathcal{F} = \{s/1\}$$

$$\mathcal{C}_0 = \{plus(x, y, z)\} \qquad \mathcal{A} = \{0\}$$

$$(N_{beam}, T_{beam}) = (10, 5) \qquad m = 3 \qquad T = 8$$

$$\mathcal{E}^+ = \{plus(s(0), 0, s(0)), plus(s^5(0), s^3(0), s^8(0)), \ldots\},$$
$$\mathcal{E}^- = \{plus(s(0), s^2(0), 0), plus(0, s^2(0), s^4(0)), \ldots\},$$
$$\mathcal{B} = \{plus(0, 0, 0)\}.$$

Experiments

- Append:

$$\mathcal{P} = \{app/3\}, \mathcal{F} = \{f/2\}$$

$$\mathcal{A} = \{a, b, c, *\} \qquad \mathcal{C}_0 = \{app(x, y, z)\}$$

$$(N_{beam}, T_{beam}) = (10, 5) \qquad m = 3 \qquad T = 4$$

$$\mathcal{E}^+ = \{app([c], [], [c]), app([a, a, b], [b, c], [a, a, b, b, c]), \ldots\},$$
$$\mathcal{E}^- = \{app([], [a, a], [a, a, b]), app([b], [], [c]), \ldots\},$$
$$\mathcal{B} = \{app([], [], [])\}.$$

Experiments

- Delete:

$$\mathcal{P} = \{del/3\},\ \mathcal{F} = \{f/2\}$$

$$\mathcal{C}_0 = \{del(x, y, z)\} \qquad \mathcal{A} = \{a, b, c, *\}$$

$$(N_{beam}, T_{beam}) = (10, 5) \qquad m = 2 \qquad T = 4$$

$$\mathcal{E}^+ = \{del(b, [a, c, b], [a, c]), del(a, [b, a, a], [b, a]), \ldots\},$$
$$\mathcal{E}^- = \{del(c, [c, a, a], [a, b]), del(b, [b], [a]), \ldots\},$$
$$\mathcal{B} = \{del(a, [a], []), del(b, [b], []), del(c, [c], [])\}.$$

Experiments

- Subtree:

$$\mathcal{P} = \{sub/2\} \qquad \mathcal{F} = \{f/2\}$$

$$\mathcal{C}_0 = \{sub(x, y)\} \qquad \mathcal{A} = \{a, b, c\}$$

$$(N_{beam}, T_{beam}) = (15, 3) \qquad m = 4 \qquad T = 4$$

$$\mathcal{E}^+ = \{sub(f(b,b), f(f(f(b,b), f(a,c)), f(a,c))), \ldots\},$$
$$\mathcal{E}^- = \{sub(f(a,a), f(f(c,a), f(a,c))), \ldots\},$$
$$\mathcal{B} = \{sub(a,a), sub(b,b), sub(c,c)\}.$$

Experiment 1

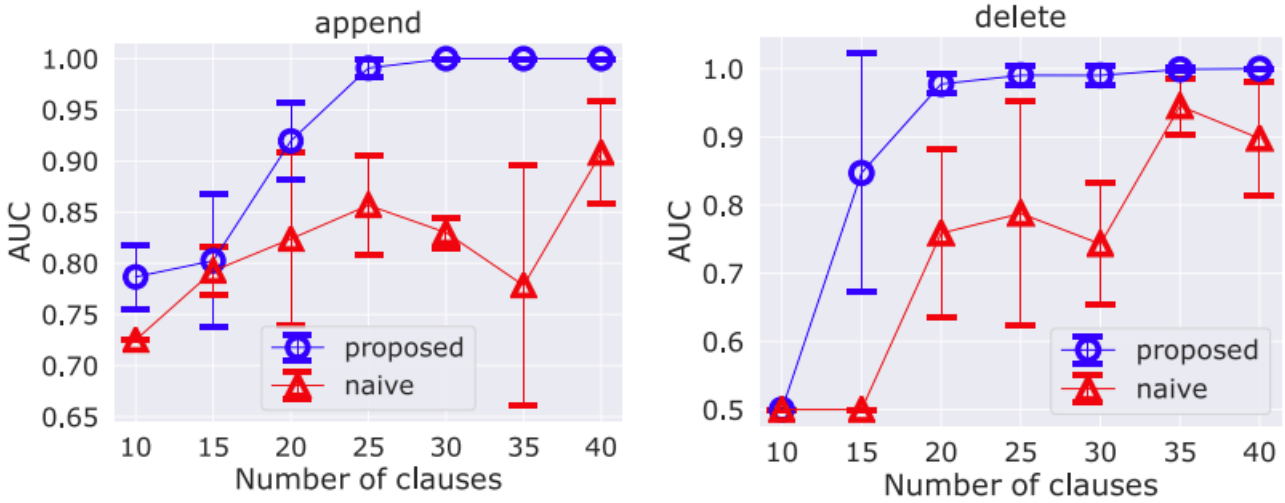
- The dILP method won’t be feasible here.
- Enumeration algorithm yields reasonable number of ground atoms

Member	Plus	Append	Delete	Subtree
228	1857	2899	2513	2172

Table 1: Number of enumerated ground atoms

Experiment 2

- Two Clause Generation Algorithms:
- 1.Beam Searching + Refinement
- 2.Naive generation without beam searching



AUC for number of generated clauses

Experiment 3

- 1.Multiple weights and softor approach $\mathbf{W} \in \mathbb{R}^{|\mathcal{C}|}$
- 2. 2-d weights for pairs of clauses
- assigned weights in the form of a 2-d matrix

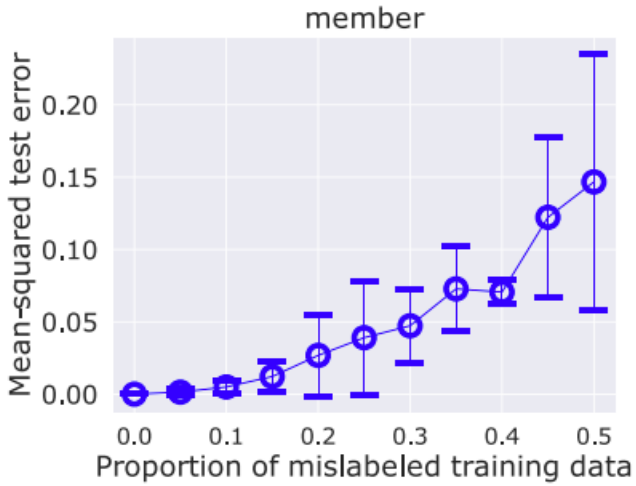
$$\mathbf{W} \in \mathbb{R}^{|\mathcal{C}| \times |\mathcal{C}|}$$

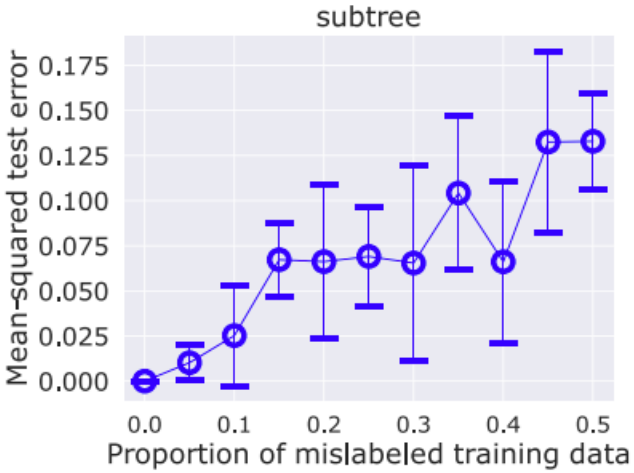
	Parameters		Runtime [s]	
	Proposed	Pair	Proposed	Pair
Member	24	144	**0.015**	0.12
Plus	120	1600	**0.03**	6.91
Append	150	2500	**0.09**	5.18
Delete	150	2500	**0.06**	5.4
Subtree	80	400	**0.039**	1.11

Table 2: Number of parameters and mean runtime in learning steps

Experiment 4

- Change the proportion of the mislabeled training data.
- Mean-squared test error as proportion of mislabeled training data: (gets a solution for mislabeled data as well)





Result

Problem	Learned logic program
Member	$mem(x, [y\|z]) \leftarrow mem(x, z)$ $mem(x, [x\|y])$
Plus	$plus(0, x, x)$ $plus(x, s(y), s(z)) \leftarrow plus(x, y, z)$ $plus(s(x), y, s(z)) \leftarrow plus(y, x, z)$
Append	$app([], x, x)$ $app(x, [], x)$ $app([x\|y], z, [x\|v]) \leftarrow app(y, z, v)$
Delete	$del(x, [x\|y], y)$ $del(x, [y\|z], [y\|v]) \leftarrow del(x, z, v)$
Subtree	$sub(f(x, y), f(x, y))$ $sub(x, f(y, z)) \leftarrow sub(x, z)$ $sub(x, f(y, z)) \leftarrow sub(x, y)$ $sub(x, f(y, x))$

Table 3: Learned logic programs in standard ILP tasks

Conclusion

- Differentiable inductive logic programming framework that deals with complex logic programs that have several clauses with function symbols that yield readable outputs for *noisy structured* data.
- Clause generation algorithm uses beam searching with refinement that improves differential program searching
- Enumeration algorithm for sufficient ground atoms from given examples and clauses
- Soft program composition approach using multiple weights and the softor function
- Proposed method estimates logic programs efficiently in terms of memory and computational costs

Conclusion

- Filling the gaps for both *∂ILP* and *standard ILP* approaches, the framework learns logic programs successfully from noisy and structured examples
- Incorporates symbolic methods, such as refinement, with a differentiable ILP approach

Limitations

- Scalability for large-scale programs.
- Learning more expressive programs for more complex tasks like learning sorting from captured images.
  (Incorporating biases[2] to manage the search space can help solve this)

[2]Claire, N.; Celine, R.; Hilde, A.; Francesco, B.; and Birgit, ´ T. 1996. Declarative bias in ILP. Advances in inductive logic programming 32: 82—103

Ira A. Fulton Schools of Engineering

Arizona State University

Algorithm 1 Clause generation by beam searching

Input: $\mathcal{C}_0, \mathcal{Q}, N_{beam}, T_{beam}$

$\mathcal{C}_{to\_open} \leftarrow \mathcal{C}_0$
$\mathcal{C} \leftarrow \emptyset$
$t = 0$
while $t < T_{beam}$ **do**
  $\mathcal{C}_{beam} \leftarrow \emptyset$
  for $C_i \in \mathcal{C}_{to\_open}$ **do**
    $\mathcal{C} = \mathcal{C} \cup \{C_i\}$
    for $R \in \rho_{\mathcal{L}}(C_i)$ **do**
      $score = eval(R, \mathcal{Q})$ //Evaluate each clause
      $\mathcal{C}_{beam} = insert(\mathcal{C}_{beam}, R, score)$ //Insert refined clause in order of scores possibly discarding it
  $\mathcal{C}_{to\_open} = \mathcal{C}_{beam}$ //top-N_{beam} clauses are refined in the next loop
  $t = t + 1$
return $\mathcal{C}$

Algorithm 2 Enumeration of ground atoms

Input: $\mathcal{Q}, \mathcal{C}, T$

$\mathcal{G} \leftarrow \{\bot, \top\} \cup \mathcal{E}^+ \cup \mathcal{E}^- \cup \mathcal{B}$
for $i = 0$ to $T - 1$ **do**
    $\mathcal{S} \leftarrow \emptyset$
    for $A \leftarrow B_1, \ldots, B_n$ in $\mathcal{C}$ **do**
        for $G \in \mathcal{G}$ **do**
            if $\bar{\sigma}(A, G)$ **then**
                $\theta \leftarrow \sigma(A, G)$
                $\mathcal{S} \leftarrow \mathcal{S} \cup \{B_1\theta, \ldots, B_n\theta\}$
    $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{S}$
return $\mathcal{G}$