
Embeddings as Probabilistic Equivalence in Logic Programs

Jaron Maene
KU Leuven
Leuven, Belgium
jaron.maene@kuleuven.be

Efthymia Tsamoura
Huawei Labs*
Cambridge, United Kingdom
efthymia.tsamoura@huawei.com

Abstract

The integration of logic programs with embedding models resulted in a class of neurosymbolic frameworks that jointly learn symbolic rules and representations for the symbols in the logic (constant or predicate). The key idea that enabled this integration was the *differentiable relaxation of unification*, the algorithm for variable instantiation during inference in logic programs. Unlike unification, its relaxed counterpart exploits the similarity between symbols in the embedding space to decide when two symbols are semantically equivalent. We show that this similarity between symbols violates the transitive law of equivalence, leading to undesirable side effects in learning and inference. To alleviate those side effects, we are the first to revamp the well-known possible world semantics of probabilistic logic programs into new semantics called *equivalence semantics*. In our semantics, a probabilistic logic program induces a probability distribution over all possible equivalence relations between symbols, instead of a probability distribution over all possible subsets of probabilistic facts. We propose a factorization of the equivalence distribution using latent random variables and characterize its expressivity. Additionally, we propose both exact and approximate techniques for reasoning in our semantics. Experiments on well-known benchmarks show that the equivalence semantics leads to neurosymbolic models with up to 42% higher results than state-of-the-art baselines.

1 Introduction

Probabilistic logic programs offer a principled way to reason in the presence of uncertain knowledge [De Raedt and Kimmig, 2015]. In neurosymbolic AI, this uncertain knowledge is provided by neural models [Marra et al., 2024, Feldstein et al., 2024]. Probability can hence be the “glue” between symbolic background knowledge and neural networks, leading to end-to-end differentiable architectures [Rocktäschel and Riedel, 2017, Manhaeve et al., 2021].

The most straightforward way to reason symbolically over the outputs of neural networks is by treating them as probabilistic facts [Manhaeve et al., 2021]. A second alternative is to map the constants and predicates of the logic program into a representation space, achieving a tighter integration. The question hence arises, *how can we apply a symbolic rule over elements in an embedding space?* The answer is via adopting the simple, yet powerful notion of *soft unification* [Rocktäschel and Riedel, 2017], which relaxes the *unification* algorithm in logic programming [Sessa, 2002]. Instead of requiring two constants and predicates to be syntactically the same, they need to be “semantically equivalent” to some extent. We demonstrate the notion of unification and its soft counterpart below.

Example 1. Consider the rule $\text{isIn}(X, Y) \leftarrow \text{isIn}(X, Z) \wedge \text{isIn}(Z, Y)$ and the following three facts: $\text{locatedIn}(\text{eiffel_tower}, \text{paris})$, $\text{isIn}(\text{paris}, \text{france})$, $\text{isIn}(\text{france}, \text{europe})$. In the context of logic pro-

*Work started before Efthymia Tsamoura joined Huawei Labs.

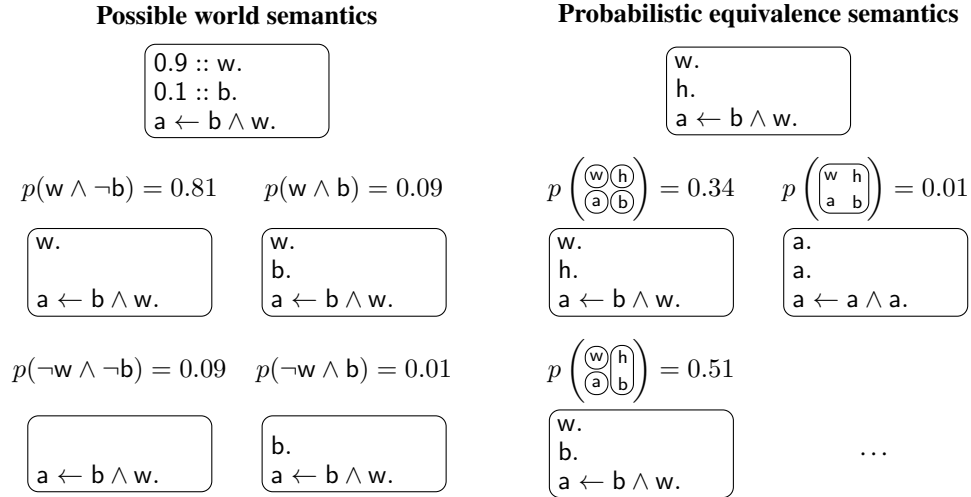


Figure 1: Example probabilistic logic programs, under the possible world (left) and the probabilistic equivalence semantics (right). The top row displays the probabilistic programs, while the rows below are the discrete programs in the corresponding program distributions. Above each different program, we write its probability according to the distribution.

gramming, *unification* is the mechanism to derive new facts by establishing a mapping σ from variables to constants. This mapping aims to instantiate the premise of a rule using the existing facts. In our example, applying the mapping $\sigma = \{X \mapsto \text{paris}, Y \mapsto \text{europe}, Z \mapsto \text{france}\}$ to our rule, we can derive the new fact $\text{isIn}(\text{paris}, \text{europe})$ using the existing facts $\text{isIn}(\text{paris}, \text{france})$ and $\text{isIn}(\text{france}, \text{europe})$. Establishing σ is only possible because $\text{isIn}(\text{paris}, \text{france})$ shares the same predicate isIn with the atom $\text{isIn}(X, Z)$ and the fact $\text{isIn}(\text{france}, \text{europe})$ shares the same predicate with the atom $\text{isIn}(Z, Y)$. The fact $\text{isIn}(\text{eiffel_tower}, \text{france})$ cannot be derived: unification fails as the predicate locatedIn is distinct from isIn . To support such cases, soft unification extends unification by allowing us to consider as being the same symbols (constants or predicates) that are different from each other, yet semantically equivalent. In our example, soft unification does identify σ , subject to isIn being equivalent to locatedIn . The probability that two symbols are equivalent is determined by their proximity in the underlying representation space.

Motivation. Soft unification has been particularly appealing, as it enables simultaneously learning symbolic rules and symbol embeddings in an end-to-end differentiable fashion [Campero et al., 2018, Weber et al., 2019, Minervini et al., 2020a,b, Maene and De Raedt, 2023]. The common characteristic of all these techniques is that they all use a *similarity* between symbols, disregarding that equivalence is transitive. However, abusing the semantics of equivalence in probabilistic logic programming has problematic consequences in learning and inference. More precisely, we show that soft unification leaks probability mass and suffers from local optima.

Contributions. To overcome the above consequences, we revamp the semantics of probabilistic logic programs, so that they induce a distribution $P_{\mathcal{E}}$ over *all possible equivalence relations* rather than a distribution over *all possible worlds* (all possible subsets of facts in the program). Figure 1 visualizes how placing uncertainty on equivalence between constants compares to the usual possible world semantics. Our semantics, referred to as *probabilistic equivalence*, allows us to treat symbol embeddings as a factorization of $P_{\mathcal{E}}$ and to reason over them. Additionally, we re-purpose existing inference techniques developed for reasoning under equivalence for the probabilistic setting. We propose an exact inference pipeline based on singularization [Marnette, 2009] and the magic sets transformation [Beeri and Ramakrishnan, 1987], as well as a Monte-Carlo estimator for approximate inference. Our empirical analysis against state-of-the-art techniques that learn and reason over symbol embeddings relying on soft unification [Rocktäschel and Riedel, 2017, Minervini et al., 2020a, Maene and De Raedt, 2023] shows that our semantics leads to models with up to 42% higher accuracy over the state-of-the-art. In summary, we make the following contributions:

- In Section 3.1, we show that soft unification violates the semantics of equivalence in probabilistic logic programming and investigate its side-effects for learning and inference.
- We propose a sound distribution semantics over probabilistic logic programs under equivalence relations called equivalence semantics in Section 4. Our semantics overcomes the independence assumption in probabilistic logic programming [Van Krieken et al., 2024].
- In Section 4.1, we propose a factorization of the program distribution induced by our semantics using latent random variables and characterize its expressivity. In this factorization, we realize the latent random variables as symbol embeddings enabling reasoning over them.
- We introduce exact and approximate inference techniques for our semantics, in Section 5.
- We empirically assess our proposed technique in link prediction and structure learning and compare it against state-of-the-art neurosymbolic approaches that employ soft unification. Our results show that our equivalence semantics improves performance up to 42%.

Supplementary material, including all code and proofs, is available at https://github.com/ML-KULEuven/equality_reasoning.

2 Preliminaries

We first briefly introduce (probabilistic) logic programs and equivalence relations. For a more extensive introduction to probabilistic logic programming, we refer to De Raedt and Kimmig [2015] or Riguzzi [2023]. We adopt probabilistic Datalog, which is a subset of more general probabilistic programming languages such as ProbLog [Fierens et al., 2015] or dPASP [Geh et al., 2024]. Although our technical presentation focuses on Datalog, it largely carries over to these more general languages. All notation used in the paper is summarized in Table 3 in the Appendix.

Logic programming. An *atom* is an expression of the form $r(t_1, \dots, t_n)$ where r is an n -ary *predicate* and each t_i is a *variable* or *constant*. A symbol is a predicate or a constant. A *Datalog rule*, or simply *rule*, is an expression of the form $h \leftarrow b_1 \wedge \dots \wedge b_n$, where the conclusion h is an atom and the premise $b_1 \wedge \dots \wedge b_n$ is a conjunction of n atoms. A *ground* (or non-ground) atom includes no variable (or no constant) argument. A rule is (non-)ground when each occurring atom is (non-)ground. A ground rule with an empty body is also called a *fact*. A *Datalog program*, or simply *program*, $\mathcal{P}$ is a tuple $(\mathcal{F}, \mathcal{R})$, where $\mathcal{F}$ is a set of facts and $\mathcal{R}$ is a set of non-ground rules. The *Herbrand universe* U of $\mathcal{P}$ is the set of all constants that occur in $\mathcal{F}$. The *Herbrand base* B of $\mathcal{P}$ is the set of all ground atoms constructed using the relations and constants in $\mathcal{P}$. Throughout the text, we fix a (probabilistic) logic program $\mathcal{P}$. All formal statements that will follow are w.r.t. this program.

Example 2. Consider the program $\mathcal{P}_{ex}$ including the facts $\mathcal{F}_{ex} = \{r(a, b), r(b, c)\}$ and the rules $\mathcal{R}_{ex} = \{r(X, Y) \leftarrow r(X, Z) \wedge r(Z, Y)\}$. The Herbrand universe of $\mathcal{P}_{ex}$ is $U = \{a, b, c\}$ and its Herbrand base is $B = \{r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c)\}$.

A set of ground atoms $I \subset B$ satisfies a ground rule $h \leftarrow b_1, \dots, b_n$ if $h \in I$ holds when each b_i is in I . The grounding of a rule in $\mathcal{P}$ is the rule that results after consistently replacing each variable with a constant from U . The *least model* $M(\mathcal{P})$ of $\mathcal{P}$ is the smallest superset of $\mathcal{F}$ that satisfies each possible grounding of each rule in $\mathcal{R}$. Every program has a unique least model [Ceri et al., 1989]. In Example 2, we have $M(\mathcal{P}_{ex}) = \mathcal{F}_{ex} \cup \{r(a, c)\}$. The program $\mathcal{P}$ entails an atom α , written as $\mathcal{P} \models \alpha$, if α is an element of the least model $M(\mathcal{P})$.

Probabilistic logic programming. Probabilistic logic programs extend traditional logic programs by treating each fact $f \in \mathcal{F}$ as an independent Bernoulli random variable that becomes true with probability $P_f(f)$, where $P_f : \mathcal{F} \rightarrow [0, 1]$. As such, a probabilistic logic program $\mathcal{P}_p = (\mathcal{F}, \mathcal{R}, P_f)$ induces a distribution over all (non-probabilistic) logic programs $(w, \mathcal{R})$, each formed by taking a subset of facts w from $\mathcal{F}$. Each such subset $w \subseteq \mathcal{F}$ is called a *possible world*. This semantics is known as the possible world semantics [Sato, 1995]. The probability $P(\alpha)$ a fact α is true in $\mathcal{P}_p$ is defined as the probability a possible world w leads to the entailment of α :

$$P_{\mathcal{F}}(w) := \prod_{f \in w} P_f(f) \prod_{f \in \mathcal{F} \setminus w} (1 - P_f(f)) \quad \text{and} \quad P(\alpha) := \mathbb{E}_{w \sim P_{\mathcal{F}}} [(w, \mathcal{R}) \models \alpha]. \quad (1)$$

Example 3. The program from Example 2 has four possible worlds: $\{\}$, $\{r(a, b)\}$, $\{r(b, c)\}$, and $\{r(a, b), r(b, c)\}$. Only the last one derives $r(a, c)$. By setting $P_f = \{r(a, b) \mapsto 0.5, r(b, c) \mapsto 0.1\}$, we have $P(r(a, c)) = P_{\mathcal{F}}(r(a, b)) \cdot P_{\mathcal{F}}(r(b, c)) = 0.05$.

Equivalence. A relation is a set of n -ary tuples over a domain of elements. An equivalence relation e over a set S is a binary relation $e \subseteq S \times S$ that is *reflexive*, i.e., $\forall x \in S : e(x, x)$, *symmetric*, i.e., $\forall x, y \in S : e(x, y) \Rightarrow e(y, x)$, and *transitive*, i.e., $\forall x, y, z : e(x, y) \wedge e(y, z) \Rightarrow e(x, z)$. Equivalence relations are isomorphic to set partitions, and, hence, we use them interchangeably. We write $\mathcal{E}_S$ to denote the set of all equivalence relations over the set S .

We use equivalence relations both over the Herbrand universe (constants) and the Herbrand base (ground atoms) of a program. Following Maene and De Raedt [2023], we define equivalence relations over the constants of the program and not over the atoms. However, each equivalence relation e_U over the Herbrand universe U implies an equivalence relation e_B over the Herbrand base B . In particular,

$$e_B(r(a_1, \dots, a_n), r(b_1, \dots, b_n)) \text{ holds in } B \text{ if and only if } \bigwedge_{i=1}^n e_U(a_i, b_i) \text{ holds in } U. \quad (2)$$

The above definition might seem restrictive as atoms with different predicates can never be equivalent. To support such cases, we can simply take predicates as arguments. For example, $r_1(x_1, y_1)$ and $r_2(x_2, y_2)$ become $t(r_1, x_1, y_1)$ and $t(r_2, x_2, y_2)$, where t is a fresh dummy predicate.

3 Equivalence in Logic Programming

Intuitively, equivalence creates exchangeability between the facts in the program, meaning that the semantics of the program should remain the same if we replace a fact with an equivalent one, see also Figure 1 (right). For example, if under an equivalence relation b is equivalent to c , there should be no difference between the rules $a \leftarrow b$ and $a \leftarrow c$. We formalize equivalence in logic programs using the notion of *saturation* in Definition 2. Then, we formalize soft unification in Definition 3 and show that soft unification violates the semantics of equivalence in probabilistic logic programming and conclude with the impact on learning and inference in Theorems 2 and 3.

Definition 1 (Set Saturation). *Consider a set T and an equivalence relation $e \in \mathcal{E}_T$ over T . The saturation $S|_e$ of a set $S \subseteq T$ subject to e is the smallest superset of S that is a union of partitions of e .*

$$S|_e := \{y \mid (x, y) \in e, x \in S\} \quad (3)$$

When a set is its own saturation, i.e., $S = S|_e$, we also say that S is saturated subject to e . We next expand this definition of saturation from sets to programs.

Definition 2 (Program Saturation). *For an equivalence relation $e_U \in \mathcal{E}_U$, the program $\mathcal{P}|_{e_U}$ is the saturation of the program $\mathcal{P}$ subject to e_U if $M(\mathcal{P}|_{e_U}) \cap B$ is the smallest model of $\mathcal{P}$ that is saturated subject to e_B .*

Based on Definition 2, if a saturated program entails a fact α , it also entails the fact α' if the two facts are equivalent. We provide an example of saturation below.

Example 4. *Consider the equivalence relation $e_U = \{(a, a), (b, b), (c, c), (a, b), (b, a)\}$ and the program $\mathcal{P}_{ex}$ from Example 2. The program $\mathcal{P}_{ex}$ is not saturated subject to e_U , since $(r(a, a), r(a, b)) \in e_B$ and $\mathcal{P} \models r(a, b)$, but $\mathcal{P} \not\models r(a, a)$. The saturation of the set $\mathcal{F}_{ex} = \{r(a, b), r(b, c)\}$ subject to e_B is $\mathcal{F}_{ex}|_{e_B} = \mathcal{F}_{ex} \cup \{r(a, a), r(b, b), r(b, a), r(a, c)\}$.*

A standard way to saturate a program is to explicitly axiomatize the equivalence relation and add the corresponding axioms to the program [Fitting, 1996, Chapter 9]. Specifically, for a given equivalence relation e_U , the axiomatization requires introducing (i) a fresh binary predicate $\overset{e_U}{\approx}$, (ii) all facts in $F(e_U) := \{a \overset{e_U}{\approx} b \mid (a, b) \in e_U\}$ and (3) a set of congruence rules $C_{e_U}(\mathcal{R})$:

$$C_{e_U}(\mathcal{R}) := \left\{ p(X_1, \dots, X_n) \leftarrow p(X'_1, \dots, X'_n) \wedge \bigwedge_{i=1}^n X_i \overset{e_U}{\approx} X'_i \mid p \text{ is a } n\text{-ary predicate in } \mathcal{P} \right\} \quad (4)$$

In Example 2, there is one congruence rule for each equivalence relation e_U , namely $C_{e_U}(\mathcal{R}_{ex}) = \{r(X, Y) \leftarrow r(X', Y') \wedge X \overset{e_U}{\approx} X' \wedge Y \overset{e_U}{\approx} Y'\}$. We have the following result.

Theorem 1. *For any equivalence relation $e_U \in \mathcal{E}_U$, the program $(\mathcal{F} \cup F(e_U), \mathcal{R} \cup C_{e_U}(\mathcal{R}))$ is a saturation $\mathcal{P}|_{e_U}$ of the program $\mathcal{P}$ subject to e_U .*

Unlike previous work on question answering over logical theories with equality, we do not axiomatize the symmetry and transitivity properties. This is because the facts in $F(e_U)$ are already an equivalence relation – in Benedikt et al. [2018], the basic assumption is that the input facts $\mathcal{F}$ include no $\approx$ -facts.

3.1 Soft Unification

Before formalizing soft unification, we will introduce some key notions. Let $\rho := \mathcal{U} \times \mathcal{U}$ be the equivalence relation where all constants are equivalent. We use $\approx$ as a predicate to denote equivalence under ρ and define the set $F(\rho) := \{c_1 \approx c_2 \mid c_1, c_2 \in \mathcal{U}\}$.

We now introduce soft unification using the notions from Section 3. Let $C_\rho(\mathcal{R})$ be the set of rules defined as in (4); however, these rules are defined based on $\approx$. By associating a Bernoulli random variable with each fact in $F(\rho)$, we can capture the semantics of soft unification using a class of probabilistic logic programs referred to as *soft unification programs*.

Definition 3 (Soft Unification). *Let $d : \mathbb{R}^k \times \mathbb{R}^k \rightarrow [0, 1]$ be a function that returns the similarity between the embeddings $\vec{v}_a, \vec{v}_b \in \mathbb{R}^k$ of two symbols a and b in $\mathcal{U}$. A soft unification program $\mathcal{P}_s$ over a set of rules $\mathcal{R}$, a set of facts $\mathcal{F}$, and the similarity function d is defined as the tuple:*

$$\mathcal{P}_s := (\mathcal{F} \cup F(\rho), \mathcal{R} \cup C_\rho(\mathcal{R}), P_d) \quad (5)$$

The probability a fact $a \approx b \in F(\rho)$ is true in $\mathcal{P}_s$ is defined as $P_d(a \approx b) := d(\vec{v}_a, \vec{v}_b)$. The probability that a fact in $\mathcal{F}$ is true in $\mathcal{P}_s$ is simply one².

According to Definition 3, each possible world of a soft unification program is a set of $\approx$ -facts, where the probability of each such fact is computed using the constant embeddings. We give an example:

Example 5. *The soft unification program given the rules and the facts in the running example includes the facts $\{r(a, b), r(b, c), a \approx a, a \approx b, a \approx c, b \approx a, b \approx b, b \approx c, c \approx a, c \approx b, c \approx c\}$ and the rules $\{r(X, Y) \leftarrow r(X, Z) \wedge r(Z, Y), r(X, Y) \leftarrow r(X', Y') \wedge X \approx X' \wedge Y \approx Y'\}$. One possible world of this program is $\{a \approx a, a \approx b, b \approx c\}$.*

In the appendix, we discuss how Definition 3 can represent different soft unification techniques [Rocktäschel and Riedel, 2017, Minervini et al., 2020a, Maene and De Raedt, 2023].

Do soft unification programs adhere to the properties of equivalence relations? From Definition 3, it follows that to enforce a soft unification program to treat equivalence as a reflexive relation, it suffices to choose the similarity function d such that $d(\vec{v}_c, \vec{v}_c) = 1$ for every embedded symbol $\vec{v}_c$. Symmetry can similarly be enforced, e.g. by defining $\approx$ as an unordered predicate. However, soft unification programs do not treat equivalence as a transitive relation, failing to capture the true semantics of equivalence. Returning to Example 5, there are possible worlds in which the facts $a \approx b$ and $b \approx c$ hold, but the fact $a \approx c$ does not hold. Nonetheless, under a mild assumption on the similarity function d , we get a weak kind of transitivity due to the use of embeddings.

Theorem 2. *For each possible world $w \subseteq F(\rho)$ of a soft unification program $\mathcal{P}_s$ and each similarity function d satisfying*

$$d(\vec{v}_1, \vec{v}_2) = 1 \text{ if and only if } \vec{v}_1 = \vec{v}_2, \forall \vec{v}_1, \vec{v}_2 \in \mathbb{R}^k, \quad (6)$$

$P(w) = 1$ implies that the $\approx$ -facts in w satisfy the semantics of transitivity.

Theorem 2 implies that a soft unification program distribution can only place arbitrarily high probability mass on possible worlds where the $\approx$ -facts satisfy the semantics of transitivity. For example, there exists no embedding of the constants a , b , and c that satisfies the following probabilities: $P(a \approx b) = 1$, $P(b \approx c) = 1$, and $P(a \approx c) = 0$. From a Bayesian viewpoint, the probability mass on the possible worlds whose $\approx$ -facts do not satisfy the semantics of transitivity is essentially “leaked”. Indeed, after training convergences, these possible worlds need to have probability zero, see also Van Krieken et al. [2024].

Soft unification also poses challenges in learning. It is known that any distribution over Boolean variables is a multilinear polynomial function [Darwiche, 2003]. Hence, all optima are global, making it straightforward to learn neurosymbolic models computing the gradients of those functions, such as DeepProbLog [Manhaeve et al., 2021]. The above does not hold when using soft unification.

Theorem 3. *The probability $P(\alpha)$ a ground atom $\alpha \notin \mathcal{F}$ is true in a soft unification program $\mathcal{P}_s$ is not a multilinear polynomial function in the embeddings.*

²Conventionally, the facts in a soft unification program are non-probabilistic. However, it is also possible to associate a probability to them as in probabilistic logic programming, see (1).

4 Semantics of Equivalence in Probabilistic Logic Programming

This section introduces our semantics, referred to as *probabilistic equivalence semantics*. Our semantics relies on a distribution over the equivalence relations $P_{\mathcal{E}} : \mathcal{E}_{\mathcal{U}} \rightarrow [0, 1]$. Recall that possible world semantics relies on a distribution over the possible worlds $P_{\mathcal{F}} : 2^{\mathcal{F}} \rightarrow [0, 1]$, Section 2. As in soft unification, we use $\approx$ as a predicate to denote equivalence in ρ and assume no $\approx$ -atom occurs in $\mathcal{R}$ or $\mathcal{F}$.

Definition 4 (Probabilistic Equivalence Semantics). *A probabilistic equivalence program $\mathcal{P}_{\mathbf{e}}$ over a set of rules $\mathcal{R}$, a set of facts $\mathcal{F}$, and a distribution $P_{\mathcal{E}} : \mathcal{E}_{\mathcal{U}} \rightarrow [0, 1]$ is defined as the tuple $(\mathcal{R}, \mathcal{F}, P_{\mathcal{E}})$. The probability a fact $a \approx b \in \mathcal{F}(\rho)$ is true in $\mathcal{P}_{\mathbf{e}}$ is defined as the probability (a, b) belongs to some $e_{\mathcal{U}} \in \mathcal{E}_{\mathcal{U}}$:*

$$P(a \approx b) := \mathbb{E}_{e_{\mathcal{U}} \sim P_{\mathcal{E}}} [(a, b) \in e_{\mathcal{U}}] \quad (7)$$

The probability a non- $\approx$ fact α is true in $\mathcal{P}_{\mathbf{e}}$ is defined as the probability α is entailed by a saturation $\mathcal{P}_{|e_{\mathcal{U}}}$ of $(\mathcal{R}, \mathcal{F})$ subject to some $e_{\mathcal{U}} \in \mathcal{E}_{\mathcal{U}}$:

$$P(\alpha) := \mathbb{E}_{e_{\mathcal{U}} \sim P_{\mathcal{E}}} [\mathcal{P}_{|e_{\mathcal{U}}} \models \alpha] \quad (8)$$

Section 4.1 provides a definition of $P_{\mathcal{E}}$ based on an embedding model of constants. We provide an example of Definition 4 below.

Example 6. *Consider the running example with a distribution over the five possible equivalence relations on $\mathcal{U} = \{a, b, c\}$.*

$$P_{\mathcal{E}} = \left\{ \begin{array}{l} \text{Diagram 1} \mapsto 0.4, \text{Diagram 2} \mapsto 0.3, \text{Diagram 3} \mapsto 0.0, \text{Diagram 4} \mapsto 0.2, \text{Diagram 5} \mapsto 0.1 \end{array} \right\}$$

The probability of $r(b, a)$ under Definition 4 is $P(r(b, a)) = 0.3 + 0.0 + 0.1 = 0.4$, since either $a \approx b$ or $a \approx c$ must hold to derive $r(b, a)$ from the least model $M(\mathcal{P}_{ex}) = \{r(a, b), r(b, c), r(a, c)\}$.

The probabilistic equivalence semantics (Definition 4) can be seen as the possible world semantics (Definition 1) by defining $P(w) := \mathbb{E}_{e_{\mathcal{U}} \sim P_{\mathcal{E}}} [w = \mathcal{F}_{|e_{\mathcal{B}}}]$. However, unlike in the possible world semantics, the facts in each possible w are no longer independent. We further discuss this in Appendix C.

4.1 Factorizing the Distribution Over Equivalence Relations

We now discuss how to represent an equivalence distribution $P_{\mathcal{E}} : \mathcal{E}_{\mathcal{U}} \rightarrow [0, 1]$ (see Definition 4) over embeddings of constants. Similarly to representing the full joint distribution $P_{\mathcal{F}}$ over all possible worlds, we need to make additional assumptions on $P_{\mathcal{E}}$ to make the representation of $P_{\mathcal{E}}$ tractable, as the size of $\mathcal{E}_{\mathcal{U}}$, known as the Bell number, grows exponentially in the size of $\mathcal{U}$. Inspired by probabilistic clustering [Deng and Han, 2018], we associate an independent latent random variable V_c with each constant $c \in \mathcal{U}$. Hence, the probability c belongs to the i^{th} partition is $P(V_c = i)$. We can treat the probability vector of the categorical V_c as an embedding for the constant c .

$$P_{\mathcal{E}}(e_{\mathcal{U}}) := \sum_{\pi_e} \prod_{c \in \mathcal{U}} P(V_c = \pi_e(c)) \quad (9)$$

Here, $\pi_e : \mathcal{U} \rightarrow \{1, \dots, k\}$ is any function such that $\forall (a, b) \in e_{\mathcal{U}} : \pi_e(a) = \pi_e(b)$, with k being the embedding dimension. Although π_e is required for the above definition of $P_{\mathcal{E}}(e_{\mathcal{U}})$ to marginalize away the possible orderings of the partitions, it never needs to be computed in practice. An advantage of this factorization is that we can trivially calculate the expectation of two atoms being equivalent.

Corollary 1. *When $P_{\mathcal{E}}$ is defined as in (9), the probability an equivalence atom $a \approx b$ is true in a probabilistic equivalence program $\mathcal{P}_{\mathbf{e}} = (\mathcal{R}, \mathcal{F}, P_{\mathcal{E}})$ is given by:*

$$P(a \approx b) = \sum_{e_{\mathcal{U}} \in \mathcal{E}_{\mathcal{U}}} \mathbb{I}[(a, b) \in e_{\mathcal{U}}] \sum_{\pi_e} \prod_{c \in \mathcal{U}} P(V_c = \pi_e(c)) = \sum_{i \in \{1, \dots, k\}} P(V_a = i) P(V_b = i)$$

Our factorization assumes that the probability a constant is part of a partition is independent, reducing the expressivity. Specifically, we prove below that it can no longer represent any combination of marginal probabilities between constants.

Theorem 4. Consider a matrix M that contains all the marginal probabilities of constants being equivalent under the probabilistic equivalence semantics, i.e., $M_{i,j} = P(c_i \approx c_j)$ with $c_i, c_j \in \mathcal{U}$. If $P_{\mathcal{E}}$ is factorized as in (9), M is positive semi-definite.

According to Theorem 4, the marginals of equivalence facts should always form a positive semi-definite matrix M , or they cannot be expressed by our factorization (9).

5 Inference & Learning

This section discusses exact and approximate techniques for inference using probabilistic equivalence programs over a constant embedding space. We start by briefly discussing probabilistic Datalog inference. Then, we discuss the changes required to support our semantics.

Probabilistic Datalog. Datalog inference is typically based on the fixed-point semantics [Ceri et al., 1989]. Starting with a given set of facts $\mathcal{F}$, at each inference step, we apply the rules in $\mathcal{R}$ to the existing facts until no new facts can be derived. Modern Datalog engines rely on techniques such as semi-naïve evaluation [Bancilhon, 1986] and Trigger Graphs [Tsamoura et al., 2021] to reduce the number of rule applications deriving no new facts. By keeping track of the rules applied to derive a targetfact, we can compute the *lineage* of the fact, i.e., the Boolean formula representing the possible worlds that lead to its derivation. Calculating the probability of the target fact then reduces to calculating the probability of this Boolean formula [Fierens et al., 2015, Vlasselaer et al., 2016], a problem known as *weighted model counting* [Sang et al., 2005].

Singularization. Consider a single equivalence relation. The naïve approach to reason under this relation is to introduce a special predicate $\approx$ and add the congruence rules $\mathcal{C}(\mathcal{R})$ in (4) to the program, Theorem 1. More advanced techniques deal with equivalence by taking advantage of the symmetry between elements in the same partition [Fitting, 1996, Chapter 9]. This idea was implemented as *paramodulation* [Robinson and Wos, 1983] and *singularization* [Marnette, 2009]. We briefly explain the latter technique below, as it is most suited to Datalog.

Definition 5 (Singularization [Marnette, 2009]). The singularization $\text{Sg}(\mathcal{R}, p)$ of a set of rules $\mathcal{R}$ with respect to a target predicate p is the set that includes each rule $r \in \mathcal{R}$ transformed as follows: for each variable X that occurs more than once in a non- $\approx$ -atom in the premise of r , X is replaced with a unique variable X_i and the atom $X \approx X_i$ is added to the premise of r . In addition, $\text{Sg}(\mathcal{R}, p)$ includes the congruence rule $p(X'_1, \dots, X'_n) \leftarrow p(X_1, \dots, X_n) \wedge X_1 \approx X'_1 \wedge \dots \wedge X_n \approx X'_n$.

In Definition 5, the predicate p is different from $\approx$, it is defined by exactly one rule and cannot occur in the body of any rule in $\mathcal{R}$. This assumption is without loss of generality, as it can always be enforced by introducing fresh predicates. In our running example, singularization turns the transitive rule $r(X, Y) \leftarrow r(X, Z) \wedge r(Z, Y)$ into $r(X, Y) \leftarrow r(X, Z_1) \wedge r(Z_2, Y) \wedge Z_1 \approx Z_2$. Singularization alters the least model and does not preserve probability in general. However, it leads to sound inference under our semantics (see Theorem 5 below).

Magic sets transformation. A further well-known optimization in Datalog inference is goal-driven inference. Throughout the rest of this section, consider a target predicate p as in Definition 5 and a ground atom $p(c)$, where c is a tuple of constants. The goal is to find whether the program entails $p(c)$. To do that, instead of computing the least model entirely, specific atoms, called *magic atoms*, prevent the application of a rule until it is necessary for the derivation of $p(c)$. This approach is known as the *magic sets transform* [Beeri and Ramakrishnan, 1987], which we denote as $\text{Mag}(\mathcal{R}, p(c))$. Appendix D demonstrates the magic transform in the running example. As we have a regular Datalog program after singularization, we can apply magic sets out of the box, bringing us to the final program.

Theorem 5. For each distribution $P_{\mathcal{E}} : \mathcal{E}_{\mathcal{U}} \rightarrow [0, 1]$ and each target ground p -atom α , we have

$$\mathbb{E}_{e_{\mathcal{U}} \sim P_{\mathcal{E}}} [\mathcal{P}_{|e_{\mathcal{U}}} \models \alpha] = \mathbb{E}_{e_{\mathcal{U}} \sim P_{\mathcal{E}}} [(\mathcal{F} \cup \mathcal{F}(e_{\mathcal{U}}), \text{Mag}(\text{Sg}(\mathcal{R}, p), \alpha)) \models \alpha].$$

The exact inference procedure is summarized in Algorithm 1. We assume that distributions $P_{\mathcal{E}}$ are represented over constant embeddings as in Section 4.1. The function Lin computes the lineage of the target. The lineage of the Datalog program contains only $\approx$ -facts, due to the latent variable representation (see Section 4.1). These $\approx$ -facts are encoded into a Boolean formula whose variables have probabilities encoded in a vector $\mathbf{W}$ using the function BooleanEnc , see Cao et al. [2023], De Smet and Zuidberg Dos Martires [2024]. Finally, WMC computes the weighted model counting of the resulting formula subject to the weights.

Algorithm 1: Exact Inference

Input: Program $\mathcal{P} = (\mathcal{F}, \mathcal{R})$,
equivalence distribution $P_{\mathcal{E}}$ as in (9), and
target p-fact α .

Output: Probability $P(\alpha)$

Step 1: Singularization

$\mathcal{R}' := \text{Sg}(\mathcal{R}, \mathbf{p})$

Step 2: Magic Sets

$\mathcal{R}'' := \text{Mag}(\mathcal{R}', \alpha)$

Step 3: Lineage Computation

$\text{lineage} := \text{Lin}(\mathcal{F} \cup \mathcal{F}(\mathbf{U} \times \mathbf{U}), \mathcal{R}'', \alpha)$

Step 4: Boolean Encoding

$\phi_{\alpha}, \mathbf{W} := \text{BooleanEnc}(\text{lineage}, \mathcal{E}_{\mathbf{U}})$

Step 5: Weighted Model Counting

$P(\alpha) := \text{WMC}(\phi_{\alpha}, \mathbf{W})$

return $P(\alpha)$

Algorithm 2: Approximate Inference

Input: Program $\mathcal{P} = (\mathcal{F}, \mathcal{R})$,
equivalence distribution $P_{\mathcal{E}}$ as in (9),
target p-fact α , and number of samples k .

Output: Approximation of $P(\alpha)$.

Step 1: Magic Sets

$\mathcal{R}' := \text{Mag}(\mathcal{R}, \alpha)$

Step 2: Sampling the Latents

$\mathbf{V} \sim P_{\mathcal{E}}$

Step 3: Constant Abstraction

$\mathcal{F}' := \text{Inst}(\mathcal{F}, \mathbf{V})$

Step 4: Forward Inference

$x := \text{CheckEntails}(\mathcal{F}', \mathcal{R}', \text{Inst}(\alpha, \mathbf{V}))$

Step 5: Repeat steps 2–4, collecting k

samples $[x_1, x_2, \dots, x_k]$.

return $\text{mean}([x_1, x_2, \dots, x_k])$

Approximate Inference. As exact probabilistic inference is often intractable, we also include a straightforward Monte-Carlo approximation, similar to Gutmann et al. [2011]. In our case, the probability of a ground atom α is estimated as the average number of times the α is entailed (i.e., $\mathcal{P}|_{e_{\mathbf{U}}} \models \alpha$) under the sampled equivalence relations $e_{\mathbf{U}}$. In particular, let $\text{Inst}(\mathcal{F}, \mathbf{V})$ be the set of facts that results after replacing each constant $c \in \mathbf{U}$ in each fact in $\mathcal{F}$ with the value of its latent V_c in $\mathbf{V}$. Here, we write $\mathbf{V}$ for an assignment to all the latent variables of the constants in $\mathbf{U}$. After applying the instantiation Inst , we obtain a regular Datalog program. So applying singularization is no longer necessary.

In Lemma 4, we prove that such an instantiation is a valid saturation of the program. Algorithm 2 contains the pseudo-code and we prove the soundness of our sampling technique below.

Theorem 6. *For each distribution $P_{\mathcal{E}} : \mathcal{E}_{\mathbf{U}} \rightarrow [0, 1]$ and each target ground p-atom α , we have*

$$\mathbb{E}_{e_{\mathbf{U}} \sim P_{\mathcal{E}}} [\mathcal{P}|_{e_{\mathbf{U}}} \models \alpha] = \lim_{k \rightarrow \infty} \frac{1}{k} \sum_{i=1}^k \mathbb{1}[(\text{Inst}(\mathcal{F}, \mathbf{V}_i), \text{Mag}(\mathcal{R}, \alpha)) \models \text{Inst}(\alpha, \mathbf{V}_i)], \text{ where } \mathbf{V}_i \sim P_{\mathcal{E}}. \quad (10)$$

Learning. Using exact inference, the computation for $P(\alpha)$ is end-to-end differentiable, and we can hence learn the embeddings from entailment using gradient descent on the negative log-likelihood ($-\log P(\alpha)$). When sampling for approximate inference, we lose differentiability and require a gradient estimator. Our experiments use the score function estimator (also known as REINFORCE) in combination with the leave-one-out baseline for variance reduction [Kool et al., 2019].

6 Experiments

We assess the effect of our semantics on two settings where embeddings and rules need to be learnt jointly. First, we consider link prediction in knowledge graphs. Second, we consider differentiable finite state machines, where the structure of a finite state machine needs to be learned from sequences of subsymbolic data.

6.1 Link Prediction in Knowledge Graphs

Benchmarks. We use two well-known small knowledge graphs: countries [Bouchard et al., 2015] and nations [Rummel, 1992]. Countries contains the locations of countries and regions in the world and comes in three variants with increasing difficulty (S1, S2, and S3). Nations contains geopolitical relations between countries. There are no standard data splits for nations, which complicates the comparison with prior results. We adopt the same splits as Rocktäschel and Riedel [2017] and Minervini et al. [2020b].

Table 1: Link prediction results on the countries and nations knowledge graphs. We report the mean and standard deviation over 10 seeds. Baselines are taken from Minervini et al. [2020b,a], Maene and De Raedt [2023]. Tables 5 and 6 in the appendix show the used hyperparameters.

Dataset	Metric	NTP	GNT	CTP	DeepSoftLog	Ours
Countries	S1 AUC-PR	90.83 ± 15.4	99.98 ± 0.05	100.0 ± 0.00	100.0 ± 0.00	100.0 ± 0.00
	S2 AUC-PR	87.40 ± 11.7	90.82 ± 0.88	91.81 ± 1.07	97.67 ± 0.98	100.0 ± 0.00
	S3 AUC-PR	56.68 ± 17.6	87.70 ± 4.79	94.78 ± 0.00	97.90 ± 1.00	99.89 ± 0.31
Nations	MRR	0.61	0.658	0.709 ± 0.03	timeout	0.724 ± 0.01
	Hits@1	0.45	0.493	0.562 ± 0.05	timeout	0.591 ± 0.02
	Hits@3	0.73	0.781	0.813 ± 0.03	timeout	0.819 ± 0.01
	Hits@10	0.87	0.985	0.995 ± 0.00	timeout	0.988 ± 0.01

Evaluation. As is standard in the literature, we report the area under the precision-recall curve (AUC-PR) for the countries knowledge graph and use filtered ranking [Bordes et al., 2013] for nations. For each fact $p(a, b)$ in the test dataset, we take all possible corrupted facts by replacing one of the two arguments, creating $p(a', b)$ and $p(a, b')$, and filter out any corrupted facts that appear in the knowledge graph. We then rank the probability of the test fact compared to all these corrupted facts. When different facts have the same probability, previous works often rank ties in their favor. This is known to skew results [Sun et al., 2020], and we instead do random tie-breaking. Using the ranks, we report the mean reciprocal rank and hits@ k (with $k \in \{1, 3, 10\}$).

Setup. We train with the same rule templates as prior work (c.f. Appendix B) and set the embedding dimension to the vocabulary size to avoid limiting the expressivity. We use approximate inference as described in Section 5 with the GLog Datalog engine [Tsamoura et al., 2021]. The only biased aspect of optimization is that we truncate the fixed-point iteration of the Datalog engine, meaning the derivation of each fact can take at most 8 steps. This was found to speed up training with no significant performance penalty. We train for 2 epochs with the Adam optimizer. All hyperparameters are summarized in Appendix B. As baselines, we compare with prior work that reasons on embeddings in logic programs. Namely, the Neural Theorem Prover (NTP) [Rocktäschel and Riedel, 2017], the Greedy Neural Theorem Prover (GNT) [Minervini et al., 2020b], the Conditional Theorem Prover (CTP) [Minervini et al., 2020b], and DeepSoftLog [Maene and De Raedt, 2023].

Results. The results are summarized in Table 1. On every tested knowledge graph, we outperform existing soft unification models. Notably, this is still the case for CTP, which is not restricted to a specific set of rule templates, as all other tested methods are.

6.2 Differentiable Finite State Machines

Differentiable finite state machines classify an input sequence of MNIST images into accepting and non-accepting sequences. For example, in the language $(01)^*$, **01** and **0101** are accepting sequences while **1** and **0001** are non-accepting sequences. The goal is to jointly train a neural network, in this case a digit classifier, and the transition rules of the state machine.

We follow the same experimental setting as Maene and De Raedt [2023], using three different binary languages: the language of 01 repetitions, the language of sequences with exactly one 1, and the language of sequences with an even number of 1's. In each case, 20 example digits are given as concept supervision to provide a non-random baseline accuracy to the neural network. After this, the models are trained on sequences of length 4. To test generalization, the test sequences have double the length and use images disjoint from the training split. Table 2 displays the results. As a baseline, we include both DeepSoftLog and a simple RNN. On each language, the introduction of transitivity greatly improves the results of DeepSoftLog, to the point of saturating this benchmark.

7 Related Work

Equivalence & logic. Equivalence in logic has been studied extensively [Nilsson and Maluszynski, 1995, Chap. 13; Fitting, 1996, Chap. 9], going back to Jaffar et al. [1984]. Logic with equivalence relations has been used in the context of database dependencies [Marnette, 2009, Benedikt et al., 2018], the semantic web [Motik et al., 2015], compiler optimizations [Zhang et al., 2023], or second-

Table 2: AUC-PR on sequence classification with differentiable automata for three grammars. We report the mean and standard deviation over 10 seeds. Baselines are from Maene and De Raedt [2023]

Language	(01)*	0*10*	(0 10*1)*
RNN	77.63 ± 15.05	61.59 ± 10.09	50.14 ± 1.36
DeepSoftLog	83.93 ± 25.87	87.01 ± 7.18	56.12 ± 15.98
Ours	99.60 ± 1.02	97.46 ± 5.87	99.61 ± 0.54

order theories [Tsamoura and Motik, 2024]. To the best of our knowledge, we are the first to place *uncertainty* on the equivalence. Exchangeability in probability distributions was also studied under the name of lifted inference [Niepert and Van den Broeck, 2014]. In lifted inference, there is exchangeability between the variables of the probability distribution, while in our work, we have a probability distribution on the exchangeability between symbols.

Soft unification. Many extensions of logic have been proposed to handle uncertainty or vagueness. Similarity-based logics extend fuzzy logic with a similarity on symbols [Gerla and Sessa, 1999, Fontana and Formato, 2002, Godo and Rodríguez, 2008]. This fuzzy matching between symbols has been variously called soft or weak unification, as it can be seen as a relaxation of the unification algorithm that matches symbols during SLD-resolution. A key difference between the similarity used in soft unification and the equivalence we consider is the former does not respect transitivity. Further works have studied the model-based semantics of soft unification [Sessa, 2002, Medina et al., 2004] and the use of similarities or proximities [Medina et al., 2004, Julián-Iranzo and Rubio-Manzano, 2015] and introduced practical implementations such as Bousi~Prolog [Julián-Iranzo et al., 2009].

Related to soft unification, Cohen [2000a,b] relaxed database joins using similarity. Rocktäschel and Riedel [2017] took a neurosymbolic view on soft unification with the Neural Theorem Prover (NTP) by learning the embeddings in an end-to-end differentiable way. Minervini et al. [2020a] improved the scalability of the NTP using greedy approximations. Maene and De Raedt [2023] argued for using probabilistic logic instead of the fuzzy semantics, as the fuzzy inference leads to poor optimization [de Jong and Sha, 2019]. All these works still rely on similarity and do not use full equivalence. Lastly, relational graph neural network architectures may approximate the behaviour of NTPs through message passing [Barbiero et al., 2024].

Knowledge graph embeddings. Embeddings are the standard approach in knowledge graphs to handle incompleteness [Bordes et al., 2013]. From a probabilistic view, embeddings can be seen as a low-rank decomposition of the distribution over triples [Loconte et al., 2023]. In contrast to our work, the probability of an atom here only relies on its own embeddings (up to normalization). Moreover, these probabilistic semantics of embeddings do not allow rule learning. Several recent knowledge graph methods have proposed neurosymbolic rule learning [DeLong et al., 2024]. Unlike our work, these are typically not end-to-end differentiable and do not have sound probabilistic semantics.

8 Conclusion & Limitations

We investigated the semantics of soft unification for learning and reasoning over embeddings. We showed that soft unification does not adhere to the properties of logical equivalence causing issues in inference and learning. To overcome those issues, we introduced the equivalence semantics in logic programming. We linked our semantics to reasoning over constant embeddings by associating each constant with a category corresponding to the partition the constant belongs. We also adapted existing techniques for inference in the presence of equality to reason under our semantics. Lastly, we experimentally confirmed that enforcing transitivity can considerably improve performance on neurosymbolic tasks, where the rules and embeddings are learned jointly.

Several open questions and limitations remain. First, developing lower variance methods for gradient estimation is necessary for effective training with a large number of rules. Full grounding is often infeasible when learning rules, so many existing approximation strategies are not applicable. Second, our approach still relies on templates for rule learning, so either considerable language bias is incurred or inference becomes challenging. Third, the extension to other languages, such as non-monotonic logic programming or declarative probabilistic languages, is left to future work.

Acknowledgements

Jaron Maene received funding from the Flemish Government (AI Research Program). We thank Luc De Raedt for his valuable feedback.

References

- Francois Bancilhon. Naive evaluation of recursively defined relations. In *On knowledge base management systems: integrating artificial intelligence and database technologies*, pages 165–178. Springer, 1986.
- Pietro Barbiero, Francesco Giannini, Gabriele Ciravegna, Michelangelo Diligenti, and Giuseppe Marra. Relational concept bottleneck models. *Advances in Neural Information Processing Systems*, 37:77663–77685, 2024.
- Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 269–284, 1987.
- Michael Benedikt, Boris Motik, and Efthymia Tsamoura. Goal-driven query answering for existential rules with equality. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.
- Guillaume Bouchard, Sameer Singh, and Theo Trouillon. On approximate reasoning capabilities of low-rank vector spaces. In *AAAI spring symposia*, 2015.
- Andres Campero, Aldo Pareja, Tim Klinger, Josh Tenenbaum, and Sebastian Riedel. Logical rule induction and theory learning using neural theorem proving. *arXiv preprint arXiv:1809.02193*, 2018.
- William X Cao, Poorva Garg, Ryan Tjoa, Steven Holtzen, Todd Millstein, and Guy Van den Broeck. Scaling integer arithmetic in probabilistic programs. In *Uncertainty in Artificial Intelligence*, pages 260–270. PMLR, 2023.
- Stefano Ceri, Georg Gottlob, Letizia Tanca, et al. What you always wanted to know about Datalog (and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166, 1989.
- William W Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18(3):288–321, 2000a.
- William W Cohen. Whirl: A word-based information representation language. *Artificial Intelligence*, 118(1-2):163–196, 2000b.
- Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, May 2003. ISSN 0004-5411. doi: 10.1145/765568.765570.
- Michiel de Jong and Fei Sha. Neural theorem provers do not learn rules without exploration. *arXiv preprint arXiv:1906.06805*, 2019.
- Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100:5–47, 2015.
- Lennert De Smet and Pedro Zuidberg Dos Martires. A fast convoluted story: Scaling probabilistic inference for integer arithmetic. In *Conference on Neural Information Processing Systems*, 2024.
- Lauren Nicole DeLong, Ramon Fernández Mir, and Jacques D Fleuriot. Neurosymbolic AI for reasoning over knowledge graphs: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

- Hongbo Deng and Jiawei Han. Probabilistic models for clustering. In *Data Clustering*, pages 61–86. Chapman and Hall/CRC, 2018.
- Jonathan Feldstein, Paulius Dilkas, Vaishak Belle, and Efthymia Tsamoura. Mapping the neuro-symbolic AI landscape by architectures: A handbook on augmenting deep learning through symbolic reasoning. *arXiv preprint arXiv:2410.22077*, 2024.
- Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.
- Melvin Fitting. *First-order logic and automated theorem proving*. Springer, New York, NY, 1996. ISBN 978-1-4612-2360-3.
- Francesca Arcelli Fontana and Ferrante Formato. A similarity-based resolution rule. *International Journal of Intelligent Systems*, 17(9):853–872, 2002.
- Renato Lui Geh, Jonas Gonçalves, Igor C Silveira, Denis D Mauá, and Fabio G Cozman. dPASP: a probabilistic logic programming environment for neurosymbolic learning and reasoning. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 21, pages 731–742, 2024.
- Giangiacomo Gerla and Maria I Sessa. Similarity in logic programming. In *Fuzzy Logic and Soft Computing*, pages 19–31. Springer, 1999.
- Lluís Godo and Ricardo O Rodríguez. Logical approaches to fuzzy similarity-based reasoning: an overview. *Preferences and similarities*, pages 75–128, 2008.
- Bernd Gutmann, Ingo Thon, Angelika Kimmig, Maurice Bruynooghe, and Luc De Raedt. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 11(4-5):663–680, 2011.
- Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2 edition, 2012.
- Joxan Jaffar, Jean-Louis Lassez, and Michael J Maher. A theory of complete logic programs with equality. *The Journal of Logic Programming*, 1(3):211–223, 1984.
- Pascual Julián-Iranzo and Clemente Rubio-Manzano. Proximity-based unification theory. *Fuzzy Sets and Systems*, 262:21–43, 2015.
- Pascual Julián-Iranzo, Clemente Rubio-Manzano, and Juan Gallardo-Casero. Bousi~ prolog: a prolog extension language for flexible query answering. *Electronic Notes in Theoretical Computer Science*, 248:131–147, 2009.
- Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! 2019.
- Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning*, pages 2863–2872. PMLR, 2018.
- Lorenzo Loconte, Nicola Di Mauro, Robert Peharz, and Antonio Vergari. How to turn your knowledge graph embeddings into generative models. *Advances in Neural Information Processing Systems*, 36:77713–77744, 2023.
- Jaron Maene and Luc De Raedt. Soft-unification in deep probabilistic logic. *Advances in Neural Information Processing Systems*, 36:60804–60820, 2023.
- Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504, 2021.
- Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 13–22, 2009.

- Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, and Luc De Raedt. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, 328:104062, 2024.
- Jesús Medina, Manuel Ojeda-Aciego, and Peter Vojtáš. Similarity-based unification: a multi-adjoint approach. *Fuzzy sets and systems*, 146(1):43–62, 2004.
- Pasquale Minervini, Matko Bošnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. Differentiable reasoning on large knowledge bases and natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5182–5190, 2020a.
- Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *International Conference on Machine Learning*, pages 6938–6949. PMLR, 2020b.
- Boris Motik, Yavor Nenov, Robert Piro, and Ian Horrocks. Handling owl:sameAs via rewriting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.
- Mathias Niepert and Guy Van den Broeck. Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.
- Ulf Nilsson and Jan Maluszynski. *Logic, Programming, and PROLOG*. John Wiley & Sons, Inc., USA, 2nd edition, 1995. ISBN 0471959960.
- Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. {RNNL}ogic: Learning logic rules for reasoning on knowledge graphs. In *International Conference on Learning Representations*, 2021.
- Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning*. River Publishers, 2023.
- George Robinson and Larry Wos. Paramodulation and theorem-proving in first-order theories with equality. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 298–313. Springer, 1983.
- Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. *Advances in neural information processing systems*, 30, 2017.
- Rudolph J Rummel. Dimensionality of nations project: attributes of nations and behavior of nation dyads, 1950-1965. 1992.
- Tian Sang, Paul Beame, and Henry A Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–82, 2005.
- Taisuke Sato. A statistical learning method for logic programs with distribution semantics. *Logic Programming*, pages 715–730, 1995.
- Maria I Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical computer science*, 275(1-2):389–426, 2002.
- Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5516–5522, 2020.
- Efthymia Tsamoura and Boris Motik. Goal-driven query answering over first-and second-order dependencies with equality. *arXiv preprint arXiv:2412.09125*, 2024.
- Efthymia Tsamoura, David Carral, Enrico Malizia, and Jacopo Urbani. Materializing knowledge bases via trigger graphs. *Proceedings of the VLDB Endowment*, 14(6):943–956, 2021.
- Emile Van Krieken, Pasquale Minervini, Edoardo M Ponti, and Antonio Vergari. On the independence assumption in neurosymbolic learning. In *Proceedings of the 41st International Conference on Machine Learning*, pages 49078–49097, 2024.

- Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, and Luc De Raedt. Tp-compilation for inference in probabilistic logic programs. *International Journal of Approximate Reasoning*, 78:15–32, 2016.
- Leon Weber, Pasquale Minervini, Jannes Münchmeyer, Ulf Leser, and Tim Rocktäschel. NLProlog: Reasoning with weak unification for question answering in natural language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6151–6161, 2019.
- Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. Better together: Unifying datalog and equality saturation. *Proceedings of the ACM on Programming Languages*, 7(PLDI):468–492, 2023.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: All our claims are based on theoretical and experimental results.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: See Section 8.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[Yes\]](#)

Justification: All assumptions are formalized and detailed proofs are given in Appendix A.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: We provide experimental details in Section 6 and Appendix B. The source code to replicate all experiments will be released upon publication.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Source code will be released upon publication. Every used dataset is available online, see Appendix B.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: See Appendix B.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report the standard deviation over 10 seeds for each experiment.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The experiments in our paper do not require GPU resources and can be run on a personal machine.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: We foresee no specific potential harms nor specific ethical considerations.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: Our work concerns fundamental research on probabilistic logic programming and neurosymbolic AI. We foresee no immediate societal impact.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper poses no particular risk of misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: See Appendix B.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: This paper does not release any new assets,

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: Our research does not involve LLMs, nor have they been used in the creation of this paper.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.

Table 3: The notation used in our paper.

Symbol	Ref.	Description
$c \in U$	Sec. 2	A constant in U .
U	Sec. 2	The Herbrand universe of $\mathcal{P}$.
X	Sec. 2	A variable ranging over the elements of U .
$\alpha \in B$	Sec. 2	A ground atom in B .
B	Sec. 2	The Herbrand base of $\mathcal{P}$.
$f \in \mathcal{F}$	Sec. 2	A fact in $\mathcal{F}$.
$\mathcal{F}$	Sec. 2	Set of facts, i.e., ground atoms.
$h \leftarrow b_1, \dots, b_n$	Sec. 2	Datalog rule.
$\mathcal{R}$	Sec. 2	Set of non-ground rules.
$\mathcal{P} := (\mathcal{R}, \mathcal{F})$	Sec. 2	Datalog program.
$\mathcal{P}_{\text{ex}}$	Ex. 2	Program in the running example.
$M(\mathcal{P})$	Sec. 2	Least Herbrand model of $\mathcal{P}$.
$\mathcal{P} \models \alpha$	Sec. 2	Entailment, ground atom α is entailed by $\mathcal{P}$.
$\mathcal{E}_U$	Sec. 2	The set of all equivalence relations over the set U .
e_U	Sec. 3	An equivalence relation over U .
e_B	Sec. 3	An equivalence relation over B .
$\approx$	Sec. 3	Predicate to denote equivalence under e_U .
$\rho := U \times U$	Sec. 3.1	The equivalence relation in which all elements are equivalent.
$\approx$	Sec. 3.1	Predicate to denote equivalence under ρ .
$S _{e_U}$	Def. 1	Saturation of the set S subject to the equivalence relation e_U .
$\mathcal{P} _{e_U}$	Def. 2	Saturation of the program $\mathcal{P}$ subject to the equivalence relation e_U .
$P_f : \mathcal{F} \rightarrow [0, 1]$	Sec. 2	Probability for each fact in $\mathcal{F}$.
$P_d : F(\rho) \rightarrow [0, 1]$	Def. 2	Probability for each equality fact in $F(\rho)$.
$\mathcal{P}_{\mathbf{p}} := (\mathcal{F}, \mathcal{R}, P_f)$	Sec. 2	Probabilistic logic program.
$w \subseteq \mathcal{F}$	Sec. 2	Possible world, subset of the facts $\mathcal{F}$.
$2^{\mathcal{F}}$	Sec. 2	Set of all possible subsets of $\mathcal{F}$.
$P_{\mathcal{F}} : 2^{\mathcal{F}} \rightarrow [0, 1]$	Eq. 1	Probability distribution over all possible worlds of the probabilistic logic program $\mathcal{P}$.
$P_{\mathcal{E}} : \mathcal{E}_U \rightarrow [0, 1]$	Eq. 9	Probability distribution over $\mathcal{E}_U$.
$F(e_U) := \{a \approx^{e_U} b \mid (a, b) \in e_U\}$	Sec. 3	Set of equivalence facts for e_U .
$C_{e_U}(\mathcal{R})$	Eq. 4	Set of congruence rules subject to e_U .
$\mathcal{P}_s := (\mathcal{F} \cup F(\rho), \mathcal{R} \cup C_{\rho}(\mathcal{R}), P_d)$	Def. 3	Soft unification program.
$\mathcal{P}_e := (\mathcal{R}, \mathcal{F}, P_{\mathcal{E}})$	Def. 4	Probabilistic equivalence program.
V_c	Sec. 4.1	Latent categorical variable for the constant $c \in U$.
$\text{Sg}(\mathcal{R}, p)$	Def. 5	Singularization of the rules in $\mathcal{R}$ with respect to a target predicate p .
$\text{Mag}(\mathcal{R}, \alpha)$	Sec. 5	Magic sets transformation of the rules in $\mathcal{R}$ subject to the fact α .
$\mathbf{V}$	Sec. 5	Assignment to all latent variables V_c , for $c \in U$.
$\text{Inst}(\mathcal{F}, \mathbf{V})$	Sec. 5	The set of facts that results after replacing each constant $c \in U$ in each fact in $\mathcal{F}$ with the value of the latent V_c w.r.t. $\mathbf{V}$.

A Proofs

We first establish some preliminary definitions and theorems regarding lattices, fixed points, and monotone logic programming. We then use this background to establish that both models and saturated models of Datalog programs form lattices. These results are later used to prove the main theorems from the paper.

Table 4: The additional notation used in our appendix.

Symbol	Description
θ	Grounding substitution, i.e., mapping of variables to constants.
$a\theta$	Ground atom that results after replacing each variable X in a with $\theta(X)$.
$\mathcal{M}(\mathcal{P})$	Set of all models for a Datalog program $\mathcal{P}$.
$\sqsubseteq$	Partial order.
$(S, \sqsubseteq)$	Partially order set (poset) for set S .
$\inf T$	The greatest lower bound of $(T, \sqsubseteq)$.
$T_{\mathcal{P}} : 2^{\mathcal{B}} \rightarrow 2^{\mathcal{B}}$	The immediate consequence operator.
$\text{Fix} : (S \rightarrow S) \rightarrow 2^S$	Fixed point operator over set S .
$\mathcal{M}_{e_U}(\mathcal{P})$	Set of all saturated models of $\mathcal{P}$ subject to e_U .
$\inf \mathcal{M}_{e_U}(\mathcal{P})$	Least saturated model of $\mathcal{P}$ subject to e_U .

A.1 Preliminaries

Definition 6 (Partial Order). A partial order over a set S is a binary relation $\sqsubseteq$ that is reflexive, i.e., $\forall a \in S : a \sqsubseteq a$, antisymmetric, i.e., $\forall a, b \in S : a \sqsubseteq b \wedge b \sqsubseteq a \Rightarrow a = b$, and transitive, i.e., $\forall a, b, c \in S : a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c$. A partially ordered set or poset is a tuple of the form $(S, \sqsubseteq)$, where S is a set and $\sqsubseteq$ is a partial order.

For our purposes, we assume all sets are finite.

Definition 7 (Lower Bound & Infimum). Let $(S, \sqsubseteq)$ be a poset and $T \subseteq S$ be a subset of S . An element $l \in S$ is a lower bound of T if $l \sqsubseteq t$, for each $t \in T$. The lower bound l of T for which $l' \sqsubseteq l$ holds for every other lower bound l' of T is the infimum or the greatest lower bound of T , denoted by $\inf T$.

When they exist, infima are unique by antisymmetry.

Definition 8 (Semilattice). A poset $(L, \sqsubseteq)$ is a semilattice, if for each $S \subseteq L$, S has an infimum.

Definition 9 (Subsemilattice). For a semilattice $(L, \sqsubseteq)$ and a set $S \subseteq L$, the poset $(S, \sqsubseteq)$ is a subsemilattice if S is closed under infima, i.e., $\inf T \in S$ for each subset $T \subseteq S$.

We next formalize Datalog inference using the immediate consequence operator. Here, we write θ for a substitution, i.e., a mapping from variables to constants in $\mathcal{U}$. The application $\alpha\theta$ of the substitution θ to the atom α replaces each variable X in α with the constant $\theta(X)$.

Definition 10. The immediate consequence operator $T_{\mathcal{P}} : 2^{\mathcal{B}} \rightarrow 2^{\mathcal{B}}$ adds the facts and all atoms that can be derived from an interpretation to that interpretation.

$$T_{\mathcal{P}}(I) = I \cup \mathcal{F} \cup \{h\theta \mid h \leftarrow b_1, \dots, b_n \in \mathcal{R}, \\ \theta \text{ is a substitution, } b_i\theta \in I \text{ for all } i\}.$$

We slightly deviate from the standard definition by also including I in $T_{\mathcal{P}}(I)$. But this will be useful later. Next, we define the fixed point operator, as we later use the fixed points of the immediate consequence operator.

Definition 11 (Fixed Point Operator). Given a set S , the fixed point operator $\text{Fix} : (S \rightarrow S) \rightarrow 2^S$ maps a function $f : S \rightarrow S$ to the set of fixed points of f .

$$\text{Fix}(f) = \{s \mid s \in S, f(s) = s\}$$

Besides regular Datalog models, we are interested in saturated models. Observe that the least model in Definition 2 is a saturated model.

Definition 12 (Saturated Model). For an equivalence relation $e_U \in \mathcal{E}_U$, a set of ground atoms $M \subseteq \mathcal{B}$ is a saturated model of $\mathcal{P}$ subject to e_U if the following two conditions hold:

1. M is a model, i.e., $M \models \mathcal{P}$.

2. M is a saturated set subject to e_B .

Finally, we also introduce Gram matrices, as they relate to our choice of factorization in Section 4.1.

Definition 13 (Gram matrix). *Given n vectors $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$, the Gram matrix G is the matrix with as elements the inner products $G_{ij} = \langle \vec{v}_i, \vec{v}_j \rangle$.*

Every Gram matrix is symmetric and is positive semi-definite [Horn and Johnson, 2012, Theorem 7.2.10]. For real symmetric matrices, positive semi-definite means all eigenvalues are positive.

A.2 Fixed point Semantics of (Saturated) Programs

We now use the above terminology to show that the models $\mathcal{M}(\mathcal{P})$ of a Datalog program $\mathcal{P}$ form a semilattice.

Lemma 1. *For any Datalog program $\mathcal{P}$, $(\mathcal{M}(\mathcal{P}), \subseteq)$ is a semilattice, where $\subseteq$ denotes set inclusion.*

Proof. According to Definition 8, $(\mathcal{M}(\mathcal{P}), \subseteq)$ is a semilattice if (1) it is a poset and (2) for each $S \subseteq \mathcal{M}(\mathcal{P})$, the poset $(S, \subseteq)$ has an infimum. Regarding (1), $(\mathcal{M}(\mathcal{P}), \subseteq)$ is a poset since $\subseteq$ is a reflexive, antisymmetric, and transitive binary relation. To complete our proof, we need to show that (2) holds as well. The proof continues as follows. The intersection $M = \bigcap_i M_i$ of any set of models $\{M_i\} \subseteq \mathcal{M}(\mathcal{P})$ is also a model. This is because for any rule $h \leftarrow b_1, \dots, b_n \in \mathcal{R}$ and any mapping θ , if all $b_j\theta \in M$, then all $b_j\theta \in M_i$ for all i . Since each M_i is a model, $h\theta \in M_i$ for all i , thus $h\theta \in M$. Therefore, M is a lower bound for $\{M_i\}$.

M is the greatest lower bound, as for any lower bound M' for $\{M_i\}$ it holds that $M' \subseteq M_i$ for each i . So $M' \subseteq \bigcap_i M_i = M$. This proves that every set of models has an infimum and hence $\mathcal{M}(\mathcal{P})$ is a semilattice. $\square$

As is typical in the Datalog literature [Ceri et al., 1989], we characterize the models of a Datalog program as the interpretations where nothing new is derived by the immediate consequence operator $T_{\mathcal{P}}$. The lemma below is a variant of the Knaster-Tarski theorem.

Lemma 2. *The set of fixed points of the immediate consequence operator $\text{Fix}(T_{\mathcal{P}})$ equals the set of models $\mathcal{M}(\mathcal{P})$.*

$$\text{Fix}(T_{\mathcal{P}}) = \mathcal{M}(\mathcal{P})$$

Proof. We prove the two directions $\text{Fix}(T_{\mathcal{P}}) \subseteq \mathcal{M}(\mathcal{P})$ and $\text{Fix}(T_{\mathcal{P}}) \supseteq \mathcal{M}(\mathcal{P})$ separately.

($\subseteq$) Let $I \in \text{Fix}(T_{\mathcal{P}})$, so $T_{\mathcal{P}}(I) = I$. By Definition 10, $\mathcal{F} \subseteq T_{\mathcal{P}}(I) = I$. Moreover, for any rule $h \leftarrow b_1, \dots, b_n \in \mathcal{R}$ and grounding θ , if $b_i\theta \in I$ for all i , we have $h\theta \in T_{\mathcal{P}}(I) = I$. It follows that $I \in \mathcal{M}(\mathcal{P})$.

($\supseteq$) Let $M \in \mathcal{M}(\mathcal{P})$. Again by Definition 10, $T_{\mathcal{P}}(M) = M \cup \mathcal{F} \cup \{h\theta \mid \dots\}$. As M is a model, $\mathcal{F} \subseteq M$. Moreover, for any rule $h \leftarrow b_1, \dots, b_n \in \mathcal{R}$ and grounding θ , if $b_i\theta \in M$ for all i , we have $h\theta \in M$ as M is a model of $\mathcal{P}$. Thus we have $T_{\mathcal{P}}(M) = M$. $\square$

We denote the set of all saturated models subject to e_U as $\mathcal{M}_{e_U}(\mathcal{P})$. By the above definition, every saturated model is also a model. Hence, it follows immediately that $\mathcal{M}_{e_U}(\mathcal{P}) \subseteq \mathcal{M}(\mathcal{P})$.

Next, we clarify the relation between the regular models of a Datalog program and the saturated models of a program.

Lemma 3. *Let $\mathcal{P}$ be a Datalog program and $e_U \in \mathcal{E}_U$ be an equivalence relation. Then $(\mathcal{M}_{e_U}(\mathcal{P}), \subseteq)$ is a subsemilattice of $(\mathcal{M}(\mathcal{P}), \subseteq)$.*

Proof. Let $\{M_i\} \subseteq \mathcal{M}_{e_U}(\mathcal{P})$ be a subset. We prove that the intersection $M = \bigcap_i M_i$ is a saturated model and hence $\mathcal{M}_{e_U}(\mathcal{P})$ is closed under infima. To do so, we verify the two conditions of Definition 12. 1) Due to Lemma 1, M is a model of $\mathcal{P}$. 2) Consider a ground atom $\alpha \in M \cap B$ and $e_B(\alpha, \alpha')$ where $\alpha' \in B$. Then $\alpha \in M_i$ for all i . Since each M_i is saturated, $\alpha' \in M_i$ for all i . Therefore, $\alpha' \in M$. $\square$

As a corollary, the saturated program of Definition 2 always exists.

Corollary 2. *For each Datalog program $\mathcal{P}$ and each equivalence relation $e_U \in \mathcal{E}_U$, there exists a unique least saturated model:*

$$\inf \mathcal{M}_{e_U}(\mathcal{P}) = \bigcap_{M \in \mathcal{M}_{e_U}(\mathcal{P})} M \quad (11)$$

Proof. Since $\mathcal{M}_{e_U}(\mathcal{P})$ is a subsemilattice (Lemma 3), the infimum of the set of all saturated models is also a saturated model. Uniqueness follows from antisymmetry of $\sqsubseteq$. $\square$

Finally, we show that the models of a saturated program correspond with the models of a program where the constants get instantiated.

Lemma 4. *For any equivalence relation $e_U \in \mathcal{E}_U$ and any program $\mathcal{P} = (\mathcal{F}, \mathcal{R})$, the semilattice $\mathcal{M}_{e_U}(\mathcal{P})$ is isomorphic to the semilattice $\mathcal{M}((\text{Inst}(\mathcal{F}, \mathbf{V}), \mathcal{R}))$.*

Proof. Let $\mathcal{P}' = (\text{Inst}(\mathcal{F}, \mathbf{V}), \mathcal{R})$ be the instantiated program and $\text{Inst} : U \rightarrow U/e_U$ denote the map sending each constant to its equivalence class under e_U , which we extend pointwise to atoms and sets of atoms:

$$\text{Inst}(p(t_1, \dots, t_n), \mathbf{V}) = p(\text{Inst}(t_1, \mathbf{V}), \dots, \text{Inst}(t_n, \mathbf{V})), \quad \text{Inst}(I, \mathbf{V}) = \{ \text{Inst}(\alpha, \mathbf{V}) \mid \alpha \in I \}.$$

To prove the isomorphism, we first show that Inst is a bijection between $\mathcal{M}_{e_U}(\mathcal{P})$ and $\mathcal{M}(\mathcal{P}')$ (by showing injectivity and surjectivity) and second show that infima are preserved by Inst .

Injection. Let $M_1, M_2 \in \mathcal{M}_{e_U}(\mathcal{P})$ such that $\text{Inst}(M_1, \mathbf{V}) = \text{Inst}(M_2, \mathbf{V})$. Take any atom $\alpha \in M_1$. By construction, $\text{Inst}(\alpha, \mathbf{V}) \in \text{Inst}(M_1, \mathbf{V}) = \text{Inst}(M_2, \mathbf{V})$. This means that there is an atom $\alpha' \in M_2$ such that $\text{Inst}(\alpha', \mathbf{V}) = \text{Inst}(\alpha, \mathbf{V})$. By the definition of instantiation, this implies that $(\alpha, \alpha') \in e_B$. But M_2 is saturated subject to e_B , so it also holds that $\alpha \in M_2$. We can analogously prove that any atom $\alpha \in M_2$ is also an element of M_1 , proving that $M_1 = M_2$.

Surjection. Take any instantiated model $M' \in \mathcal{M}(\mathcal{P}')$. Then define the model $M = \{ \alpha \mid \text{Inst}(\alpha, \mathbf{V}) \in M' \}$. Now it holds that $\text{Inst}(M, \mathbf{V}) = \{ \text{Inst}(\alpha, \mathbf{V}) \mid \text{Inst}(\alpha, \mathbf{V}) \in M' \} = M'$, meaning Inst is surjective.

Preservation of infima. Consider two saturated models $M_1, M_2 \in \mathcal{M}_{e_U}(\mathcal{P})$. Inst commutes with intersections as

$$\begin{aligned} \text{Inst}(M_1 \cap M_2, \mathbf{V}) &= \{ \text{Inst}(\alpha, \mathbf{V}) \mid \alpha \in M_1 \cap M_2 \} \\ &= \{ \text{Inst}(\alpha, \mathbf{V}) \mid \alpha \in M_1 \} \cap \{ \text{Inst}(\alpha, \mathbf{V}) \mid \alpha \in M_2 \} \\ &= \text{Inst}(M_1, \mathbf{V}) \cap \text{Inst}(M_2, \mathbf{V}). \end{aligned}$$

Hence Inst is a bijection that preserves infima, so $(\mathcal{M}_{e_U}(\mathcal{P}), \sqsubseteq)$ and $(\mathcal{M}(\mathcal{P}'), \sqsubseteq)$ are isomorphic semilattices. $\square$

A.3 Main results

Theorem 1. *For any equivalence relation $e_U \in \mathcal{E}_U$, the program $(\mathcal{F} \cup F(e_U), \mathcal{R} \cup C_{e_U}(\mathcal{R}))$ is a saturation $\mathcal{P}_{|e_U}$ of the program $\mathcal{P}$ subject to e_U .*

Proof. Let $\mathcal{P}_{|e_U} = (\mathcal{F} \cup F(e_U), \mathcal{R} \cup C_{e_U}(\mathcal{R}))$. According to Definition 2, to prove that $\mathcal{P}_{|e_U}$ is a saturation of $\mathcal{P}$ we need to show that $M(\mathcal{P}_{|e_U}) \cap B$ is the smallest model of $\mathcal{P}$ that is saturated subject to e_B . To prove this, we first show that $\{M \cap B \mid M \in \mathcal{M}(\mathcal{P}_{|e_U})\} = \mathcal{M}_{e_U}(\mathcal{P})$, by checking the two directions ($\subseteq$ and $\supseteq$).

($\subseteq$) Let $M \in \mathcal{M}(\mathcal{P}_{|e_U})$. We show that $M \cap B \in \mathcal{M}_{e_U}(\mathcal{P})$ by verification of the two conditions of Definition 12.

1. Given that $\mathcal{F} \subseteq \mathcal{F} \cup F(e_U)$ and $\mathcal{R} \subseteq \mathcal{R} \cup C_{e_U}(\mathcal{R})$, every atom entailed in $\mathcal{P}$ is also entailed in $\mathcal{P}_{|e_U}$ due to monotonicity. Therefore, $M \cap B$ is also a model of $\mathcal{P}$.

2. Let $\alpha = p(t_1, \dots, t_n) \in M \cap B$ and suppose $e_B(\alpha, \alpha')$ where $\alpha' = p(t'_1, \dots, t'_n) \in B$. The congruence rule (4) for predicate p grounded by $X_i \mapsto t'_i$ and $X_i \mapsto t_i$ is

$$p(t'_1, \dots, t'_n) \leftarrow p(t_1, \dots, t_n) \wedge \bigwedge_{i=1}^n t'_i \stackrel{eu}{\approx} t_i.$$

By Equation 2, $e_B(\alpha, \alpha')$ implies $e_U(t_i, t'_i)$ for all $i \in \{1, \dots, n\}$. Since $e_U(t_i, t'_i)$, we have $t_i \stackrel{eu}{\approx} t'_i \in F(e_U) \subseteq M$. So as M is a model, $p(t'_1, \dots, t'_n) \in M$. Therefore, M is saturated.

($\supseteq$) Let $M \in \mathcal{M}_{e_U}(\mathcal{P})$. We show $M \cup F(e_U) \in \mathcal{M}(\mathcal{P}_{|e_U})$, by verifying that M satisfies all rules in $\mathcal{R} \cup C_{e_U}(\mathcal{R})$ and contains all facts in $\mathcal{F} \cup F(e_U)$. First, since M is a model of $\mathcal{P}$, we have M contains all facts $\mathcal{F}$ and satisfies all rules $\mathcal{R}$. Lastly, consider a congruence rule in $C_{e_U}(\mathcal{R})$

$$p(X_1, \dots, X_n) \leftarrow p(X'_1, \dots, X'_n) \wedge \bigwedge_{i=1}^n X_i \stackrel{eu}{\approx} X'_i$$

and a grounding θ with $X_i \mapsto t_i$ and $X'_i \mapsto t'_i$. Suppose all body atoms are in M : $p(t'_1, \dots, t'_n) \in M \cap B$ and $t_i \stackrel{eu}{\approx} t'_i \in M$ for all i . Since $t_i \stackrel{eu}{\approx} t'_i \in F(e_U)$, we have $e_U(t_i, t'_i)$ for all i . By Equation 2, this means $e_B(p(t_1, \dots, t_n), p(t'_1, \dots, t'_n))$. Since $p(t'_1, \dots, t'_n) \in M \cap B$ and M is saturated, we have $p(t_1, \dots, t_n) \in M$. Therefore, M is closed under all congruence rules.

We have shown $\{M \cap B \mid M \in \mathcal{M}(\mathcal{P}_{|e_U})\} = \mathcal{M}_{e_U}(\mathcal{P})$. In other words, the models of $\mathcal{P}_{|e_U}$ without the equality facts are precisely the saturated models of $\mathcal{P}$. The theorem by Corollary 2, as the least saturated model $\inf \mathcal{M}_{e_U}(\mathcal{P})$ exists and equals the least model of $\mathcal{P}'$. $\square$

Theorem 2. For each possible world $w \subseteq F(\rho)$ of a soft unification program $\mathcal{P}_s$ and each similarity function d satisfying

$$d(\vec{v}_1, \vec{v}_2) = 1 \text{ if and only if } \vec{v}_1 = \vec{v}_2, \forall \vec{v}_1, \vec{v}_2 \in \mathbb{R}^k, \quad (6)$$

$P(w) = 1$ implies that the $\approx$ -facts in w satisfy the semantics of transitivity.

Proof. Consider three constants $a, b, c \in U$, such that $(a \approx b) \in w$ and $(b \approx c) \in w$. Because $P(w) = 1$, it follows that $P(a \approx b) = 1$ and $P(b \approx c) = 1$, meaning that $\vec{v}_a = \vec{v}_b$ and $\vec{v}_b = \vec{v}_c$. Since $P(w) = 1$, it follows that $P(a \approx c) = 1$ and hence, $(a \approx c) \in w$. $\square$

Theorem 3. The probability $P(\alpha)$ a ground atom $\alpha \notin \mathcal{F}$ is true in a soft unification program $\mathcal{P}_s$ is not a multilinear polynomial function in the embeddings.

Proof. As defined in (1), the probability an atom in a probabilistic logic program $\mathcal{P}_p$ is

$$P(\alpha) = \sum_{\substack{w \in 2^{\mathcal{F}} \\ (w, \mathcal{R}) \models \alpha}} \prod_{f \in w} P_f(f) \prod_{f \in \mathcal{F} \setminus w} (1 - P_f(f)).$$

In the case of soft unification program $\mathcal{P}_s$, this becomes

$$P(\alpha) = \sum_{\substack{w \in 2^{\mathcal{F}(\rho)} \\ (\mathcal{F} \cup w, \mathcal{R} \cup C_\rho(\mathcal{R})) \models \alpha}} \prod_{a \approx b \in w} d(\vec{v}_a, \vec{v}_b) \prod_{a \approx b \in \mathcal{F}(\rho) \setminus w} (1 - d(\vec{v}_a, \vec{v}_b)).$$

As soon as there are more than two constants in the Herbrand universe, the vector $\vec{v}_c$ for each constant c occurs multiple times in the above product. This implies $P(\alpha)$ is not multilinear.

As an example, consider a program with facts $\{p(a), q(b)\}$ and the rules $\{r(X) :- p(X), q(X)\}$. If we query $r(c)$ in this program we get $P(r(c)) = d(\vec{v}_a, \vec{v}_c) \cdot d(\vec{v}_b, \vec{v}_c)$, because $a \approx b$ and $b \approx c$ allow us to conclude $p(c)$ and $q(c)$, respectively. If the function d is for instance the inner product, then it follows that $P(r(c)) = \langle \vec{v}_a, \vec{v}_c \rangle \cdot \langle \vec{v}_b, \vec{v}_c \rangle$, which is not linear in $\vec{v}_c$. $\square$

Theorem 4. Consider a matrix M that contains all the marginal probabilities of constants being equivalent under the probabilistic equivalence semantics, i.e., $M_{i,j} = P(c_i \approx c_j)$ with $c_i, c_j \in U$. If $P_{\mathcal{E}}$ is factorized as in (9), M is positive semi-definite.

Proof. As shown in Section 4.1, under the factorization of Equation 9, every element $M_{i,j}$ is an inner product between the probability vectors of c_i and c_j . In other words, M is a Gram matrix, which is positive semi-definite [Horn and Johnson, 2012, Theorem 7.2.10]. $\square$

Theorem 5. *For each distribution $P_{\mathcal{E}} : \mathcal{E}_U \rightarrow [0, 1]$ and each target ground p -atom α , we have*

$$\mathbb{E}_{e_U \sim P_{\mathcal{E}}} [\mathcal{P}_{|e_U} \models \alpha] = \mathbb{E}_{e_U \sim P_{\mathcal{E}}} [(\mathcal{F} \cup F(e_U), \text{Mag}(\text{Sg}(\mathcal{R}, p), \alpha)) \models \alpha].$$

Proof. To prove this theorem, it suffices to show that for each $e_U \in P_{\mathcal{E}}$, $(\mathcal{F} \cup F(e_U), \text{Mag}(\text{Sg}(\mathcal{R}, p), \alpha))$ entails α if and only if $\mathcal{P}_{|e_U}$ entails α .

We start by recalling Theorem 1, which says that

$$\mathcal{P}_{|e_U} \models \alpha \Leftrightarrow (\mathcal{F} \cup F(e_U), \mathcal{R} \cup C(\mathcal{R})) \models \alpha.$$

On the right side, we have a regular Datalog program. So as singularization preserves entailment [Marnette, 2009], it holds that

$$\mathcal{P}_{|e_U} \models \alpha \Leftrightarrow (\mathcal{F} \cup F(e_U), \text{Sg}(\mathcal{R}, p)) \models \alpha.$$

Furthermore, due to the soundness of the magic sets transformation [Beeri and Ramakrishnan, 1987, Theorem 4.1], it also holds that

$$\mathcal{P}_{|e_U} \models \alpha \Leftrightarrow (\mathcal{F} \cup F(e_U), \text{Mag}(\text{Sg}(\mathcal{R}, p), \alpha))$$

As the above equivalence holds for any equivalence relation e_U , it also holds in expectation, and the theorem follows. $\square$

Theorem 6. *For each distribution $P_{\mathcal{E}} : \mathcal{E}_U \rightarrow [0, 1]$ and each target ground p -atom α , we have*

$$\mathbb{E}_{e_U \sim P_{\mathcal{E}}} [\mathcal{P}_{|e_U} \models \alpha] = \lim_{k \rightarrow \infty} \frac{1}{k} \sum_{i=1}^k \mathbb{1}[(\text{Inst}(\mathcal{F}, \mathbf{V}_i), \text{Mag}(\mathcal{R}, \alpha)) \models \text{Inst}(\alpha, \mathbf{V}_i)], \text{ where } \mathbf{V}_i \sim P_{\mathcal{E}}. \quad (10)$$

Proof. From the isomorphism of Lemma 4, it follows that

$$\mathcal{P}_{|e_U} \models \alpha \Leftrightarrow (\text{Inst}(\mathcal{F}, \mathbf{V}), \mathcal{R}) \models \text{Inst}(\alpha, \mathbf{V}).$$

Furthermore, due to the correctness of the magic sets transformation [Beeri and Ramakrishnan, 1987, Theorem 4.1], it also holds that

$$\mathcal{P}_{|e_U} \models \alpha \Leftrightarrow (\text{Inst}(\mathcal{F}, \mathbf{V}), \text{Mag}(\mathcal{R}, \alpha)) \models \text{Inst}(\alpha, \mathbf{V}).$$

Hence, in expectation, we also get

$$\mathbb{E}_{e_U} [\mathcal{P}_{|e_U} \models \alpha] = \mathbb{E}_{e_U} [(\text{Inst}(\mathcal{F}, \mathbf{V}), \text{Mag}(\mathcal{R}, \alpha)) \models \text{Inst}(\alpha, \mathbf{V})].$$

Finally, we can apply the law of large numbers to arrive at the theorem.

$$\mathbb{E}_{e_U} [\mathcal{P}_{|e_U} \models \alpha] = \lim_{k \rightarrow +\infty} \frac{1}{k} \sum_{i=1}^k \mathbb{1}[(\text{Inst}(\mathcal{F}, \mathbf{V}_i), \text{Mag}(\mathcal{R}, \alpha)) \models \text{Inst}(\alpha, \mathbf{V}_i)].$$

$\square$

B Experimental Details & Hyperparameters

Hyperparameters. For the countries knowledge graph and the differentiable finite state machines, we use similar hyperparameters as DeepSoftLog, see Table 5 and Table 7. In all experiments, we used 100 samples for the gradient estimation and batch size 1 as this provided a sufficient training signal. For the nations knowledge graph, we performed Bayesian tuning on the validation set, optimizing for the MRR (see Table 6).

Program Templates The same program templates as the NTP and DeepSoftLog are used. For Countries S1, we have

$$r_1(X, Y) \leftarrow r_2(Y, X)$$

$$r_1(X, Y) \leftarrow r_2(X, Z) \wedge r_2(Z, Y)$$

For Countries S2, we have

$$r_1(X, Y) \leftarrow r_2(Y, X)$$

$$r_1(X, Y) \leftarrow r_2(X, Z) \wedge r_2(Z, Y)$$

$$r_1(X, Y) \leftarrow r_2(X, Z) \wedge r_3(Z, Y)$$

For Countries S3, we have

$$r_1(X, Y) \leftarrow r_2(Y, X)$$

$$r_1(X, Y) \leftarrow r_2(X, Z) \wedge r_2(Z, Y)$$

$$r_1(X, Y) \leftarrow r_2(X, Z) \wedge r_3(Z, Y)$$

$$r_1(X, Y) \leftarrow r_2(X, A) \wedge r_3(B, Y) \wedge r_4(A, B)$$

For nations, the following templates are repeated 20 times each.

$$r_1(X, Y) \leftarrow r_2(X, Y)$$

$$r_1(X, Y) \leftarrow r_1(X, Y)$$

$$r_1(X, Y) \leftarrow r_2(X, Z) \wedge r_3(Z, Y).$$

For the differentiable finite state machines, we simply implement a finite state in datalog. Datalog does not support functors, meaning that the finite state implementation has a fixed max length on the inputs it accepts. As we only test on inputs of max length 8 this is not a problem, however.

$$\text{accepts}(S1, S2, S3, S4, S5, S6, S7, S8, S9) \leftarrow \text{run}(\text{end_state}, S1, S2, S3, S4, S5, S6, S7, S8, S9)$$

$$\text{run}(\text{STATE}, \text{SYMBOL}, S1, S2, S3, S4, S5, S6, S7, S8) \leftarrow$$

$$\text{run}(\text{OLDSTATE}, S1, S2, S3, S4, S5, S6, S7, S8, -1)$$

$$\text{transition}(\text{OLDSTATE}, \text{STATE}, \text{SYMBOL})$$

Regularization. The nations knowledge graph quickly exhibited overfitting. Proper regularization is often crucial to obtain competitive results on link prediction [Lacroix et al., 2018]. As we have a probabilistic model, we can simply regularize the entropy, meaning we minimize the entropy of the latent random variables. In the differentiable finite state machines we regularize the neural network using weight decay.

Data License. The countries (ODbL licence) and nations (CC0 license) knowledge graphs, grammars (CC0 license), and MNIST dataset (MIT license) are all publicly available.

Compute. We performed the experiments on servers with an Intel i7-12700 CPU and 64GB RAM, although lower resources may suffice. No GPU or TPU compute was used. Roughly speaking, one differentiable finite state machine experiment takes 3 minutes, one countries experiment takes 3 to 5 hours (depending on S1/S2/S3), and one nations experiment takes about 1 hour.

Learned Rules Some example of learned rules, for countries (S2):

$$\text{neighborOf}(X, Y) \leftarrow \text{neighborOf}(Y, X)$$

$$\text{locatedIn}(X, Y) \leftarrow \text{locatedIn}(X, Z), \text{locatedIn}(Z, Y)$$

And for the nations knowledge graph:

$$\text{reexports}(X, Y) \leftarrow \text{booktranslations}(X, Y)$$

$$\text{conferences}(X, Y) \leftarrow \text{conferences}(Y, X)$$

$$\text{embassy}(X, Y) \leftarrow \text{warning}(X, Z), \text{commonbloc2}(Z, Y)$$

Additional Experiments. In Table 8, we repeat the knowledge graph experiments on the Kinship and UMLS datasets. We apply the same evaluation protocol and use the dataset splits from Qu et al. [2021]. Hyperparameters are again selected by tuning on the validation MRR for the UMLS dataset. For the Kinship dataset, we simply reuse these hyperparameters.

Table 5: Hyperparameters used in the countries experiment (c.f. Table 1). Identical hyperparameters are used for the three different variants (S1, S2, and S3).

Hyperparameter name	Value
Optimizer	AdamW
Learning rate	0.1
Number of samples	100
Number of epochs	1
Batch size	1

Table 6: Hyperparameters used in the nations experiment (c.f. Table 1).

Hyperparameter name	Value
Optimizer	AdamW
Learning rate	0.0013
Number of samples	100
Number of epochs	2
Batch size	1
Gradient Clipping	0.000093
Entropy regularization	0.0000025
Max fixed-point iterations	5

C Linking Equivalence and Possible World Semantics

The probabilistic equality and possible world semantics that have been described in the main paper are orthogonal to each other and can be combined by having a distribution both over the facts and over equivalence relations.

$$P(\alpha) = \mathbb{E}_{e_u, w}[(w, \mathcal{R})|_{e_u} \models \alpha]$$

For example, consider a program with the facts $\{r(a), r(b)\}$ and with no rules. Now, we both impose a probability distribution on the facts $\{r(a) \mapsto 0.2, r(b) \mapsto 0.8\}$ and the equivalence $P(a \approx b) = 0.5$. Then, if we query $r(a)$, we get

$$\begin{aligned} P(r(a)) &= P(a \approx b) \cdot (P(r(a)) + (1 - P(r(a))) \cdot P(r(b))) + (1 - P(a \approx b)) \cdot P(r(a)) \\ &= 0.5 \cdot (0.2 + 0.8 \cdot 0.8) + 0.5 \cdot 0.2 \\ &= 0.52 \end{aligned}$$

On the other hand, we can also interpret the equivalence as a specific low-rank parameterization of the possible world semantics.

Table 7: Hyperparameters used in the differentiable finite state machines experiment (c.f. Table 2). Identical hyperparameters are used for the three different languages.

Hyperparameter name	Value
Optimizer	AdamW
Learning rate (embeddings)	0.1
Learning rate (neural network)	0.0001
Number of samples	100
Number of epochs	25
Batch size	1
Weight Decay	0.01

Table 8: Link prediction results on the UMLS and Kinship knowledge graphs. Hyperparameters are selected by tuning on the validation MRR, and we report the mean and standard deviation over 10 random seeds. Splits and CTP baseline results are taken from Qu et al. [2021].

Datasets	Metrics	CTP	Ours
Kinship	MRR	0.335	0.408
	Hits@1	0.177	0.234
	Hits@3	0.376	0.483
	Hits@10	0.703	0.795
UMLS	MRR	0.404	0.557
	Hits@1	0.288	0.392
	Hits@3	0.430	0.666
	Hits@10	0.674	0.880

$$\begin{aligned}
P(\alpha) &= \mathbb{E}_{e_U \sim P_{\mathcal{E}}} [\mathcal{P}_{|e_U} \models \alpha] && \text{By Definition 4.} \\
&= \mathbb{E}_{e_U \sim P_{\mathcal{E}}} [(\mathcal{F} \cup F(e_U), \mathcal{R} \cup C_{e_U}(\mathcal{R})) \models \alpha] && \text{By Theorem 1.} \\
&= \mathbb{E}_w [(\mathcal{F} \cup w, \mathcal{R} \cup C(\mathcal{R})) \models \alpha]
\end{aligned}$$

In the last line, the possible world distribution is defined such that $P(w) = P(e_U)$ when there exists e_U with $w = F(e_U)$ and 0 otherwise. Observe that this distribution is not factorized into independent facts, as is usual in probabilistic logic programming.

D Magic Sets Transformation

As the equality atoms do not appear in any rule heads, there is no need to specialize the magic towards equality as was done by [Benedikt et al., 2018]. Instead, we perform a standard magic sets transformation. We demonstrate this on the running example below, without differentiating between the free and bound adornments.

Example 7. We present the program transformation $\text{Mag}(\text{Sg}(\mathcal{R}_{ex}, q), q(a, c))$ of the rules in the example program $\mathcal{P}_{ex}$ of Example 2. We introduce the special relation *magic* to guard the execution of *r*, and *q* as the target relation. To query an atom of *r*, for instance $r(a, c)$, we simply add the fact *magic*(*a*, *c*) to the program and query *q*(*a*, *c*). Note that the magic and singularization transforms are not unique, due to the different possible orderings of atoms.

$$\begin{aligned}
r(X, Y) &\leftarrow \text{magic}(X, Y) \wedge r(X, Z_1) \wedge Z_1 \approx Z_2 \wedge r(Z_2, Y) \\
\text{magic}(X, Z_1) &\leftarrow r(X, Y). \\
\text{magic}(Z_2, Y) &\leftarrow r(X, Y) \wedge r(X, Z_1) \wedge Z_1 \approx Z_2 \\
q(X, Y) &\leftarrow \text{magic}(X, Y) \wedge X \approx X_1 \wedge Y \approx Y_1 \wedge r(X_1, Y_1)
\end{aligned}$$