A Proofs

We first state the monotonicity of conjunction, which will be useful later.

Theorem 6. *In both fuzzy and probabilistic semantics, it holds that when c is independent of both a and b, we have $P(a) \leq P(b) \Rightarrow P(a \wedge c) \leq P(b \wedge c)$.*

Proof. In fuzzy semantics, the t-norm is monotonous by definition. Under probabilistic semantics, it follows as $P(a \wedge c) \leq P(b \wedge c)$ reduces to $P(a)P(c) \leq P(b)P(c)$ because of the independencies. □

Theorem 2. *Definition 2 holds when the soft-unification function s is $\wedge$-transitive.*

Proof. Suppose b is the rest of the proof, such that $P(\pi) = P(b \wedge (t_i \simeq t_j))$ and $P(\pi') = P(b \wedge (t_i \simeq t_k) \wedge (t_k \simeq t_j))$. By the monotonicity of the conjunction and independence between b and the soft-unifications, it now follows that $P(t_i \simeq t_j) \geq P((t_i \simeq t_k) \wedge (t_k \simeq t_j)) \Rightarrow P(\pi) \geq P(\pi')$. Using the soft-unification function the first part becomes $s(t_i, t_j) \geq s(t_i, t_k) \wedge s(t_k, t_j)$, which is exactly the $\wedge$-transitivity rule. □

Theorem 3. *Under the fuzzy semiring, definition 3 cannot hold if s is $\wedge$-transitive.*

Proof. By the monotonicity of t-norms, $s(x, y) > s(x, z)$ and $s(y, z) > s(x, z)$ implies $s(x, y) \wedge s(y, z) \geq s(x, z) \wedge s(x, z)$. Due to the idempotence of the minimum, this becomes $s(x, y) \wedge s(y, z) \geq s(x, z)$. On the other hand, the $\wedge$-transitivity of these inequalities implies $s(x, y) \wedge s(y, z) < s(x, z)$. This inequality is strict as $s(x, y) = s(y, z) \leq s(x, z)$ implies either $s(x, y) = s(x, z)$ or $s(x, z) = s(x, z)$. □

Theorem 4. *If the soft-unification function s is a $\times$-similarity with probabilistic semantics, we satisfy all stated properties (from Def. 1-4).*

Proof.

- Def. 1 follows from the well-definedness of the ProbLog semantics, by considering $\simeq$ as a regular probabilistic fact.
- Def. 2 follows from theorem 2 as s is transitive.
- For Def. 3, we can always create a y between some symbols x and z, by taking the mean in the embedding space (see theorem 5).
- Def. 4 follows from the symmetry and differentiability of addition and multiplication. Note that we assume that s is differentiable.

□

Theorem 5. *A soft-unification function s is a $\times$-similarity, iff $s(x, y) = e^{-d(\mathcal{E}(x), \mathcal{E}(y))}$, where d is a distance function on an embedding space and $\mathcal{E}$ maps symbols to this space.*

Proof. We first prove that given that if s is a $\times$-similarity, it can be written as $e^{-d(x,y)}$. We do this by showing that $d(x, y) = -\ln(s(x, y))$ is a distance.

1. As $s(x, x) = 1$, we have that $d(x, x) = -\ln(s(x, x)) = -\ln(1) = 0$.
2. The symmetry of d is implied by the symmetry of s:
$$d(x, y) = -\ln(s(x, y)) = -\ln(s(y, x)) = d(y, x)$$
3. The triangle inequality can be proven from the $\times$-transitivity, and the monotonicity of $\ln$:
$$d(x, z) = -\ln(s(x, z)) \leq -\ln(s(x, y) \cdot (y, z)) = d(x, y) + d(y, z)$$

As we do not have distinguishability (there can be a $x \neq y$ such that $d(x, y) = 0$), the embedding space is a pseudometric. The other direction can be proven in a similar fashion, by evaluating that $e^{-d(x,y)}$ satisfies reflexivity, symmetry, and $\times$-transitivity. □

B Example programs

We implement a variation on the alarm basian network in DeepSoftLog where the earthquake and landslide symbols are embedded. On the right, we show the translated ProbLog program.

$0.2 :: \texttt{event}(\sim\texttt{earthquake}).$
$0.5 :: \texttt{hears\_alarm}(\texttt{mary}).$
$\texttt{alarm} :- \texttt{event}(\sim\texttt{landslide}).$
$\texttt{calls}(\texttt{X}) :- \texttt{alarm}, \texttt{hears\_alarm}(\texttt{X}).$

$0.2 :: \texttt{event}(\sim\texttt{V}_1) :- \sim\texttt{V}_1 \simeq \sim\texttt{earthquake}.$
$0.5 :: \texttt{hears\_alarm}(\texttt{mary}).$
$\texttt{alarm} :- \texttt{event}(\sim\texttt{landslide}).$
$\texttt{calls}(\texttt{X}) :- \texttt{alarm}, \texttt{hears\_alarm}(\texttt{X}).$
$0.5 :: \sim\texttt{landslide} \simeq \sim\texttt{earthquake}.$
$0.5 :: \sim\texttt{earthquake} \simeq \sim\texttt{landslide}.$
$\sim\texttt{landslide} \simeq \sim\texttt{landslide}.$
$\sim\texttt{earthquake} \simeq \sim\texttt{earthquake}.$

Next, we also demonstrate an example with a linear rule.

$\texttt{eq}(\texttt{X}, \texttt{X}).$
$\texttt{q} :- \texttt{eq}(\texttt{f}(\sim\texttt{a}), \texttt{f}(\sim\texttt{b})).$

$\texttt{eq}(\texttt{V}_1, \texttt{V}_2) :- \texttt{V}_1 \simeq \texttt{V}_2.$
$\texttt{q} :- \texttt{eq}(\texttt{f}(\sim\texttt{a})), \texttt{f}(\sim\texttt{b})).$
$\texttt{f}(\texttt{V}_1) \simeq \texttt{f}(\texttt{V}_1) :- \texttt{V}_1 \simeq \texttt{V}_2.$
$\texttt{a} \simeq \texttt{a}.$
$\texttt{b} \simeq \texttt{b}.$
$\sim\texttt{a} \simeq \sim\texttt{a}.$
$0.1 :: \sim\texttt{a} \simeq \sim\texttt{b}.$
$0.1 :: \sim\texttt{a} \simeq \sim\texttt{f}(\texttt{a}).$
$0.1 :: \sim\texttt{a} \simeq \sim\texttt{f}(\texttt{b}).$
$\sim\texttt{b} \simeq \sim\texttt{b}.$
$0.1 :: \sim\texttt{b} \simeq \sim\texttt{f}(\texttt{b}).$
$0.1 :: \sim\texttt{b} \simeq \sim\texttt{f}(\texttt{a}).$
$\sim\texttt{f}(\texttt{a}) \simeq \sim\texttt{f}(\texttt{a}).$
$0.1 :: \sim\texttt{f}(\texttt{a}) \simeq \sim\texttt{a}.$
$0.1 :: \sim\texttt{f}(\texttt{a}) \simeq \sim\texttt{b}.$
$0.1 :: \sim\texttt{f}(\texttt{a}) \simeq \sim\texttt{f}(\texttt{b}).$
$\sim\texttt{f}(\texttt{b}) \simeq \sim\texttt{f}(\texttt{b}).$
$0.1 :: \sim\texttt{f}(\texttt{b}) \simeq \sim\texttt{a}.$
$0.1 :: \sim\texttt{f}(\texttt{b}) \simeq \sim\texttt{b}.$
$0.1 :: \sim\texttt{f}(\texttt{b}) \simeq \sim\texttt{f}(\texttt{a}).$

C Semantics

To prove the equivalence of the transformation with soft-unification, we need to prove two separate facts. First that if two terms soft-unify, they will unify in the transformed program. Second that they will produce the same soft-unification facts.

Theorem 6. *Consider two DeepSoftLog expressions a and b, and the ProbLog expression a^* which is the translation of a. Now a will soft-unify with b if and only if a^* unifies with b (using regular unification). Furthermore, they will result in the same substitution (ignoring the extra introduced variables), and the same soft-unifications.*

Proof. We prove the theorem by structural induction on a.

1. If a is a variable or constant, we have $a = a^*$, so it holds trivially.

2. If a is an embedded term, we have that $a^* = {\sim}V$, where V is a fresh variable. Hence both a will unify with b iff b is embedded. If b is embedded, both will add the soft-unfication fact $b \simeq a$, as the translated rule will have added this in the body.

3. If a is a functor, we have that $a = f(t_1, ..., t_k)$ and $a^* = f(t_1^*, ..., t_k^*)$. If b is not of the form $f(s_1, ..., s_k)$, both the soft-unification and unification will fail. In the other case, they will both give the same result as the unifications between s_1 and t_1 or t_1^* give the same result by structural induction. As rules are linear, variables of the t_i and t_j do not overlap, meaning that the substitutions can be combined without conflicts.

4. If a is an atom (i.e. a rule head), we can use an identical proof as for functors.

Suppose t_1 and t_2 is the corresponding term in the translated program. If t_1 is a variable or a non-embedded term, we have $t_1 = t_2$, which means both cases will result in the same substitution when unified with a third term t_3. In the case that t_1 is an embedded term, t_2 will be of the form ${\sim}V$ where V is a variable. Hence, when t_3 is not embedded, the soft-unification with t_1 will fail, and the regular unification with t_2 will also fail. When ${\sim}t_3$ is embedded, the soft-unification with s_2 succeeds and regular unification also succeeds with the substitution $V \to t_3$. The substitutions returned by soft-unification are not exactly the same as in regular unification, because of the introduced variables. However, as these are unique and do not appear in the DeepSoftLog program, they can be safely ignored. □

The above analysis can also be extended to the case of non-linear rules (see the section on linearization).

D Experiment details

D.1 MNIST-addition

The code for MNIST-addition is given in listing 2. Hyperparameters are summarized in table 5. Of note in this example is that we encode as much background knowledge as possible. The green cut encodes the independency of the sum of lower digits from the higher digits, so we can find the 1-best proof faster during evaluation (an alternative solution would be to use a geometric mean heuristic [24]). The embedded functors `mod_ten_add` and `carry` could be learned by neural networks. However, for maximal performance, we hand-coded them. This is possible as we know what the probability distribution of the modulo addition and carry should be.

Listing 2: Code for the MNIST addition experiments

```
digit(X) :- member(X, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).
embed_digit(EMB, DIGIT) :- digit(DIGIT), eq(~DIGIT, EMB).
eq(X, X).

add(X,Y,Z) :- add_(X, Y, Z, ~0).

add_([], [], [], ~0).
add_([], [], [1], ~1).
add_([HX|TX], [HY|TY], [HZ|TZ], CARRY) :-
    embed_digit(~mod_ten_add(HX, HY, CARRY), HZ),
    !, % green cut for faster evaluation
    add_(TX, TY, TZ, ~carry(HX, HY, CARRY)).
```

We ran the experiments on CPU (Intel® Core™ i7-2600 CPU @ 3.40GHz) with 16GB of RAM.

D.2 Countries

Hyperparameters are summarized in table 6. We ran all experiments on a single CPU (Apple M2).

optimizer	AdamW
learning rate	0.0003
learning rate schedule	cosine
training epochs	100
weight decay	0.00001
batch size	4
embedding dimensions	10
embedding initialization	one-hot, fixed
neural networks	LeNet5
max search depth	/

Table 5: Hyperparameters for the MNIST-addition experiments.

optimizer	AdamW
learning rate	0.01
learning rate schedule	cosine
training epochs	6
weight decay	0
batch size	4
embedding dimensions	100
embedding initialization	uniform over hypersphere
neural networks	/
max search depth	2 (S1, S2) or 3 (S3)
max branching factor	4

Table 6: Hyperparameters for the countries experiments.

D.3 Differentiable Turing Machine

Hyperparameters are summarized in table 3. We ran all experiments on a single CPU (Apple M2).

optimizer	AdamW
learning rate embeddings	0.1
learning rate perception	0.0001
learning rate schedule	cosine
training epochs	25
weight decay	0.00001
batch size	8
embedding dimensions	3
embedding initialization	uniform over hypersphere
neural networks	LeNet5
max search depth	/

Table 7: Hyperparameters for the differentiable finite state machine experiment.

E Embeddings as fuzzy logic

A different perspective on embeddings from what we considered in this paper, is to see them as a discrete distribution (i.e. the embedding is a probability vector). When we make sure embeddings are normalized (i.e. positive and sum to one) and take the dot product as a soft-unification function, we essentially get the probability that two embeddings are equal. By conjoining different soft-unifications, we have a fuzzy interpretation, as the soft-unifications are assumed to be independent.

As a demonstration, we apply this to the visual sudoku problem [3]. This benchmark requires classifying if a grid of images constitutes a valid sudoku puzzle. We follow the same protocol as in [32]. Hyperparameters for the visual sudoku experiment are in table 9. The learning rate, weight decay, and gradient clipping were chosen by Bayesian optimization on the 11th validation split. We averaged the results over the other 10 splits. We ran all experiments on CPU (Intel(R) Xeon(R) CPU

Visual Sudoku	4×4	9×9
CNN	51.5 ± 3.34	51.2 ± 2.20
NeuPSL [28]	89.7 ± 2.20	51.5 ± 1.37
A-NeSI [32]	89.8 ± 2.08	62.3 ± 2.20
DeepSoftLog	$\mathbf{94.2} \pm 1.84$	$\mathbf{65.0} \pm 1.94$

Table 8: Accuracy on visual sudoku classification. Previous results are adapted from [32].

E3-1225 v3 @ 3.20GHz). The 4×4 and 9×9 runs took about 95 and 630 seconds per experiment respectively.

Table 8 summarizes the results of DeepSoftLog and compares them with current state-of-the-art. Surprisingly, the crude fuzzy approximation outperforms all existing systems by a considerable margin.

	4×4	9×9
optimizer	AdamW	AdamW
learning rate	0.00162	0.000671
training epochs	100	300
weight decay	0.0000144	0.000116
batch size	1	1
gradient clipping	2.445	2.753
embedding dimensions	10	10

Table 9: Hyperparameters for the visual sudoku experiments.