Neural Decision–Propagation for Answer Set Programming

Thomas Eiter[1] , **Katsumi Inoue**[2] , **Sota Moriyama**[3,2]
[1]Vienna University of Technology (TU Wien), Austria
[2]National Institute of Informatics, Japan
[3]The Graduate University for Advanced Studies, SOKENDAI, Japan
thomas.eiter@tuwien.ac.at, {inoue, sotam}@nii.ac.jp

Abstract

Integration of Answer Set Programming (ASP) with neural networks has emerged as a promising tool in Neuro-symbolic AI. While existing approaches extend the capabilities of ASP to real world domains, their reasoning pipelines depend on classical solvers, which is a bottleneck for scalability. To tackle this problem, we propose a new method to compute stable models, called decision–propagation (DProp), which alternates falsity decisions and truth propagations. Successful DProp computations are shown to capture the stable model semantics. We then develop Neural DProp (NDProp), a differentiable extension of DProp with neural computation for decisions and fuzzy evaluation for propagations. We evaluate the capabilities of NDProp for learning decision heuristics as well as neuro-symbolic integration, and compare it with existing neuro-symbolic approaches. The results show that NDProp can learn to efficiently compute stable models, and it improves accuracy and scalability on neuro-symbolic benchmarks.

1 Introduction

Neuro-symbolic AI aims to develop systems that bring together robust learning in neural networks with reasoning via symbolic representations [d'Avila Garcez and Lamb, 2023; Wang *et al.*, 2025]. There has been a growing interest in using logic programming as a general symbolic reasoning framework for such integrated systems [Evans and Grefenstette, 2018; Yang *et al.*, 2020; Manhaeve *et al.*, 2021]. Answer Set Programming (ASP), a declarative programming paradigm widely used for knowledge representation and reasoning tasks [Lifschitz, 2008; Brewka *et al.*, 2011], is used in such systems. Neuro-symbolic integration in ASP has been attempted in various domains such as Visual Question Answering [Eiter *et al.*, 2022], compliance checking [Barbara *et al.*, 2023] and many more [Borroto *et al.*, 2025].

One prominent approach to accomplishing neuro-symbolic integration with ASP is to train neural networks with symbolic knowledge expressed with ASP. NeurASP [Yang *et al.*, 2020] integrates neural atoms into ASP programs, allowing neural network outputs to be directly incorporated into symbolic reasoning and learning. SLASH [Skryagin *et al.*, 2023] extends this line of work with neural-probabilistic predicates and removes low-probability predicate outcomes before ASP solving, thereby reducing the number of stable models that must be considered. However, these approaches largely follow a hybrid design in which neural components and symbolic ASP solvers remain separated by an interface: continuous neural predictions must be translated into symbolic atoms/literals, constraints, or probabilistic facts before reasoning can be performed. This introduces a combinatorial grounding problem, as the system must reason over multiple candidate associations between perceptual entities and symbolic objects before being processed by an ASP solver.

To connect neural outputs and symbolic representations smoothly, it would be desirable to compute answer sets in continuous domains. Then, to make this connection easily, we first propose a novel method for computing stable models called *decision–propagation* (DProp). A DProp run iteratively alternates between the following two phases: (i) decision, which selects undecided atoms to commit to false nondeterministically, and (ii) propagation, which updates the current partial assignment by deriving consequences of the program under the current assumptions. We then prove that successful DProp computations (runs that end without contradictions) capture the stable model semantics.

Subsequently, we propose *Neural DProp* (NDProp), a differentiable extension of DProp. NDProp uses neural networks for falsity decisions and fuzzy logic operators for propagations. As NDProp is identical to DProp in the crisp case, it is able to prove stability when the outputs are binary under a certain condition, making it easy to train in an unsupervised manner. These make the integration of NDProp into neuro-symbolic systems seamless, while still preserving a clear connection to classical stable model semantics. Furthermore, the learned decision steps and parallel capabilities via GPUs make this approach significantly more scalable.

To demonstrate the effectiveness of NDProp, we conduct three kinds of experiments: (1) random ASP solving to determine whether NDProp can learn the solving dynamics for given ASP problems, (2) neuro-symbolic integration to determine whether using NDProp has merits over existing approaches that use classical solvers, and (3) fuzzy reasoning to determine whether NDProp can reason and find stable models given fuzzy values as input. The results show that NDProp performs better than the respective baselines; it is superior with respect to accuracy, and has much higher inference speed than existing ASP-based neuro-symbolic approaches.

In summary, our contributions are:

- We introduce a novel method for computing stable models called DProp, and prove that a successful DProp computation captures stable model semantics.
- We develop NDProp, a differentiable extension of DProp that uses neural decisions and fuzzy propagations, and show that it coincides with DProp at the crisp case.
- We analyze the capabilities of NDProp in learning the solving dynamics of random ASP problems, and attempt to integrate NDProp into neuro-symbolic frameworks.
- We demonstrate that NDProp performs better than respective baselines, specifically with accuracy and inference speed.

Consequently, we propose a fully-differentiable answer set solver based on novel fixpoint semantics, yielding improved scalability compared to previous neural approaches to ASP.[1]

2 Preliminaries

2.1 Answer Set Programming

Answer Set Programming (ASP) is a declarative paradigm based on non-monotonic logic and is widely used for knowledge representation and reasoning [Lifschitz, 2008; Brewka *et al.*, 2011]. A *rule* r in a *(normal logic) program* P has the form

$$a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_l, \tag{1}$$

where a, b_i, and c_j are atoms and not denotes default negation. For a rule r of the form (1), the atom a is the *head* of the rule denoted as $H(r)$, and $b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_l$ is the *body* of the rule denoted as $B(r)$. The sets $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{c_1, \ldots, c_l\}$ represent the positive and default-negated body atoms of r, respectively. An *interpretation* I, i.e., a subset of the set $\mathcal{A}$ of all atoms, is a *model* of P if it satisfies each rule r of P, i.e., $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$ imply $H(r) \in I$. Furthermore, I is a *stable model* (*answer set*) of P if it is the least model of the *Gelfond–Lifschitz reduct* P^I, which is obtained from P by deleting every rule whose negative body intersects I and removing all negative literals from the remaining rules.

Given two interpretations I, J and a normal logic program P, an immediate consequence operator with respect to J [Van Gelder, 1993] is defined as

$$C_{P,J}(I) = \{H(r) \in \mathcal{A} \mid B^+(r) \subseteq I, B^-(r) \cap J = \emptyset\}.$$

We define $C^0_{P,J}(I) = I$ and $C^{k+1}_{P,J}(I) = C_{P,J}(C^k_{P,J}(I))$ for integer $k\,(\geq 0)$. When P^+ is a *definite program*, i.e., a program whose rule contains no default negation ($l = 0$), for fixed J there is a least fixpoint of $C_{P^+,J}$ starting from $\emptyset$, which is the least model of P^+, denoted as $C^\infty_{P^+,J}(\emptyset)$ [van Emden and Kowalski, 1976]. Then, the next lemma holds.

Lemma 1. *S is a stable model of P iff $S = C^\infty_{P,S}(\emptyset)$.*

Proof. Let $\mathcal{T}_P$ be the immediate consequence operator of a definite program P, and write $\mathcal{T}^\infty_P(\emptyset)$ for the result of iterating $\mathcal{T}_P$ from $\emptyset$ to its least fixpoint. By definition, S is a stable model of P iff $S = \mathcal{T}^\infty_{P^S}(\emptyset)$, where P^S is the Gelfond–Lifschitz reduct. For normal programs, iterating $C_{P,S}$ from $\emptyset$ coincides with iterating $\mathcal{T}_{P^S}$, so $C^\infty_{P,S}(\emptyset) = \mathcal{T}^\infty_{P^S}(\emptyset)$. Hence S is stable iff $S = C^\infty_{P,S}(\emptyset)$. □

2.2 Fuzzy Interpretations

In many real-world settings it is necessary to reason with vague or incomplete information. Fuzzy Answer Set Programming (FASP) extends ASP by interpreting atoms with a truth degree in the interval $[0, 1]$ and evaluating rule bodies, while adopting stable-model-based semantics for fuzzy interpretations [Nieuwenborgh *et al.*, 2006; Alviano and Peñaloza, 2013].

Let $\mathcal{A}$ be a set of propositional atoms. A *fuzzy interpretation* is a mapping $I : \mathcal{A} \to [0, 1]$ assigning each atom a degree of truth. To evaluate rule bodies, FASP fixes a *t-norm* $\otimes : [0, 1]^2 \to [0, 1]$ (fuzzy conjunction), i.e., a commutative, associative and monotone operator with unit 1. Typical t-norms include Gödel, Product, and Łukasiewicz. Similarly, *t-conorm* $\oplus : [0, 1]^2 \to [0, 1]$ represents fuzzy disjunction.

For a rule r of the form (1), the degree to which the body is satisfied under I is

$$I(B(r)) := \Big(\bigotimes_{i=1}^{m} I(b_i)\Big) \otimes \Big(\bigotimes_{j=1}^{l} (1 - I(c_j))\Big),$$

where default negation is interpreted using strong negation $I(not\, c) = 1 - I(c)$.

3 Decision–Propagation for ASP

In this section, we describe DProp computations, and show that successful DProp computations capture stable model semantics. DProp uses a forward chaining approach similar to [Lefèvre *et al.*, 2017; Dal Palù *et al.*, 2009], with the biggest difference being that the decisions are made at an atom and not at a rule level, making the differentiable extension much simpler.

Formally, DProp operates on an atom set $\mathcal{A}$ of P, with each state splitting $\mathcal{A}$ into disjoint true, false, and undecided sets T, F, and $U = \mathcal{A} \setminus (T \cup F)$, respectively. With initial sets $T^0 = C^\infty_{P,\mathcal{A}}(\emptyset)$, $F^0 = \emptyset$, and $U^0 = \mathcal{A} \setminus (T^0 \cup F^0)$, a *run* in DProp performs the following iterations ($k \geq 1$) until termination:

1. **Decision.** Select an undecided atom $a \in U^{k-1}$ (via some policy), then update $F^k = F^{k-1} \cup \{a\}$.
2. **Propagation.** With F^k fixed, compute $T^k = C^\infty_{P,\mathcal{A}\setminus F^k}(T^{k-1})$ and set $U^k = U^{k-1} \setminus (T^k \cup F^k)$.
3. **Termination.** If $U^k = \emptyset$, stop and return T^k; otherwise continue to the next decision.

[1]A full version of this paper with appendices is available as [Eiter *et al.*, 2026]. The code is available at https://github.com/sotam2369/NDProp.

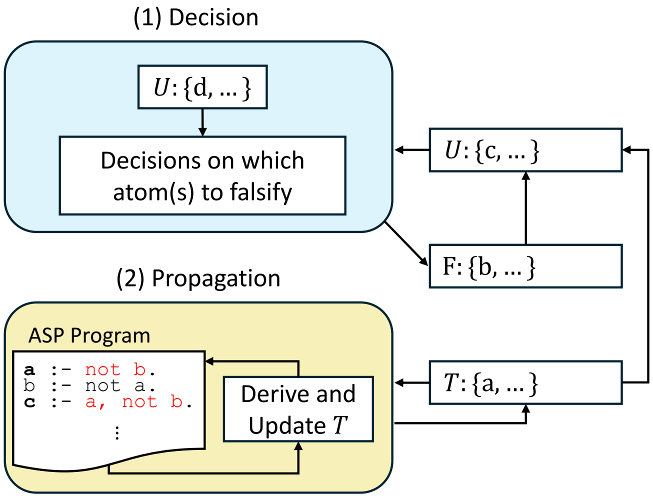

Figure 1: Decision–Propagation (DProp) iteration pipeline.

When a run terminates, we write $k = \infty$, obtaining T^∞, F^∞, and U^∞. Here, both T and F increase monotonically because Propagation and Decision never remove atoms from them.

Lemma 2. *A DProp run always terminates for a finite atom set $\mathcal{A}$.*

Proof. Decisions always remove an atom a from U^k at every iteration. Thus, the number of DProp iterations is finite, if the atom set $\mathcal{A}$ is finite. □

Definition 1 (Successful DProp computation)**.** *A successful DProp computation is a DProp run that terminates with $U^\infty = \emptyset$ and $T^\infty \cap F^\infty = \emptyset$.*

The following properties of the operator $C_{P,I}(J)$ are not hard to see.

Lemma 3. *If $F \subseteq F'$, then $C^\infty_{P,\mathcal{A}\setminus F}(\emptyset) \subseteq C^\infty_{P,\mathcal{A}\setminus F'}(\emptyset)$.*

Proof. Let $J = \mathcal{A} \setminus F$ and $J' = \mathcal{A} \setminus F'$. Since $F \subseteq F'$, we have $J' \subseteq J$. If a rule satisfies $B^-(r) \cap J = \emptyset$, then also $B^-(r) \cap J' = \emptyset$. As the reducts P^J and $P^{J'}$ are obtained by removing rules whose negative body intersects J and J' respectively, and removing negative literals from the remaining rules, $P^J \subseteq P^{J'}$. By standard properties of definite programs, their least models are monotone with respect to program inclusion, yielding $\mathcal{T}^\infty_{P^J}(\emptyset) = C^\infty_{P,\mathcal{A}\setminus F}(\emptyset) \subseteq \mathcal{T}^\infty_{P^{J'}}(\emptyset) = C^\infty_{P,\mathcal{A}\setminus F'}(\emptyset)$. □

Lemma 4. *If $T^{k-1} \subseteq C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$, then $C^\infty_{P,\mathcal{A}\setminus F^k}(T^{k-1}) = C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$.*

Proof. By assumption, $T^{k-1} \subseteq C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$. Since $C_{P,\mathcal{A}\setminus F^k}$ is monotone and $C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$ is its fixpoint, for every $n \geq 0$ we have $C^n_{P,\mathcal{A}\setminus F^k}(\emptyset) \subseteq C^n_{P,\mathcal{A}\setminus F^k}(T^{k-1}) \subseteq C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$. Since $\mathcal{A}$ is finite, there exists m such that $C^m_{P,\mathcal{A}\setminus F^k}(\emptyset) = C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$. Hence, $C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset) \subseteq C^m_{P,\mathcal{A}\setminus F^k}(T^{k-1}) \subseteq C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$, so $C^m_{P,\mathcal{A}\setminus F^k}(T^{k-1}) = C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$. Therefore, $C^\infty_{P,\mathcal{A}\setminus F^k}(T^{k-1}) = C^\infty_{P,\mathcal{A}\setminus F^k}(\emptyset)$. □

Theorem 1. *Let P be a finite program and $S \subseteq \mathcal{A}_P$, where $\mathcal{A}_P$ is the set of atoms occurring in P. Then S is a stable model of P if and only if S is the final true set of a successful DProp computation.*

Proof. ($\Rightarrow$) Let S be a stable model of P, and F^{k-1}, T^{k-1} and U^{k-1} be the sets of atoms that are true, false and undefined at the beginning of the k-th iteration, where $k = \infty$ is the final iteration. Consider a run of DProp that at each step decides the falsity of an atom $a \in (\mathcal{A} \setminus S) \cap U^{k-1}$. The propagation step then yields $T^k = C^\infty_{P,\mathcal{A}_P\setminus F^k}(T^{k-1})$, which by Lemma 4 equals $C^\infty_{P,\mathcal{A}_P\setminus F^k}(\emptyset)$. Because $F^k \subseteq \mathcal{A}_P \setminus S$, we have $S \subseteq \mathcal{A}_P \setminus F^k$ and hence, by Lemma 3, $C^\infty_{P,\mathcal{A}_P\setminus F^k}(\emptyset) \subseteq C^\infty_{P,S}(\emptyset)$. By Lemma 1, $C^\infty_{P,S}(\emptyset) = S$, so $T^k \subseteq S$ and $T^k \cap F^k = \emptyset$ for all k. At termination, $U^\infty = \emptyset$, so $T^\infty = C^\infty_{P,\mathcal{A}_P\setminus F^\infty}(\emptyset) = C^\infty_{P,T^\infty}(\emptyset)$ hence $T^\infty = S$ from Lemma 1. From Lemma 2, the computation will terminate as the given program P is finite.

($\Leftarrow$) Let S be the final true set of a successful DProp computation, and $T^{\infty-1}$ and F^∞ be the true and false sets before the final propagation step. As the propagation step does not modify F^∞, from Definition 1, $S = \mathcal{A} \setminus F^\infty$ so $S = C^\infty_{P,\mathcal{A}\setminus F^\infty}(T^{\infty-1}) = C^\infty_{P,S}(T^{\infty-1})$. From Lemma 4, $C^\infty_{P,S}(T^{\infty-1}) = C^\infty_{P,S}(\emptyset)$, and Lemma 1 implies that S is a stable model of P. □

Remark 1 (Batch decisions)**.** *The proof only relies on the fact that F grows monotonically and is held fixed during propagation. Hence any variant that, in a single decision step, adds a set $E \subseteq U$ to F (i.e., $F \leftarrow F \cup E$) satisfies the same invariants and therefore admits the same characterization: every DProp computation of such a variant yields a stable model.*

4 Neural Decision–Propagation

We now instantiate DProp with differentiable operators, yielding a neural solver NDProp that is an extension of DProp. The overview of the NDProp pipeline is shown in the middle section of Figure 2. Corresponding to the true and false sets T and F, we consider true and false *degrees* $(\tau, \phi) \in [0,1]^n \times [0,1]^n$. Furthermore, decisions are implemented via neural decision modules that gradually increase falsity.

4.1 Neural Extensions of DProp

Each atom a_i carries a true and false degree τ_i and ϕ_i. Literal evaluations are done with the val operator, which outputs for positive and negative literals

$$\mathrm{val}(a_i) = \tau_i \text{ and } \mathrm{val}(not\, a_i) = \phi_i,$$

respectively. Regarding (τ_i, ϕ_i), values near $(0,0)$ indicate undecided atoms, $(1,0)$ denotes derived truth, $(0,1)$ denotes explicit falsity, and $(1,1)$ encodes contradiction. For NDProp, we combine these into a single membership score

$$p_i = \tfrac{1}{2}(\tau_i + 1 - \phi_i) \in [0,1],$$

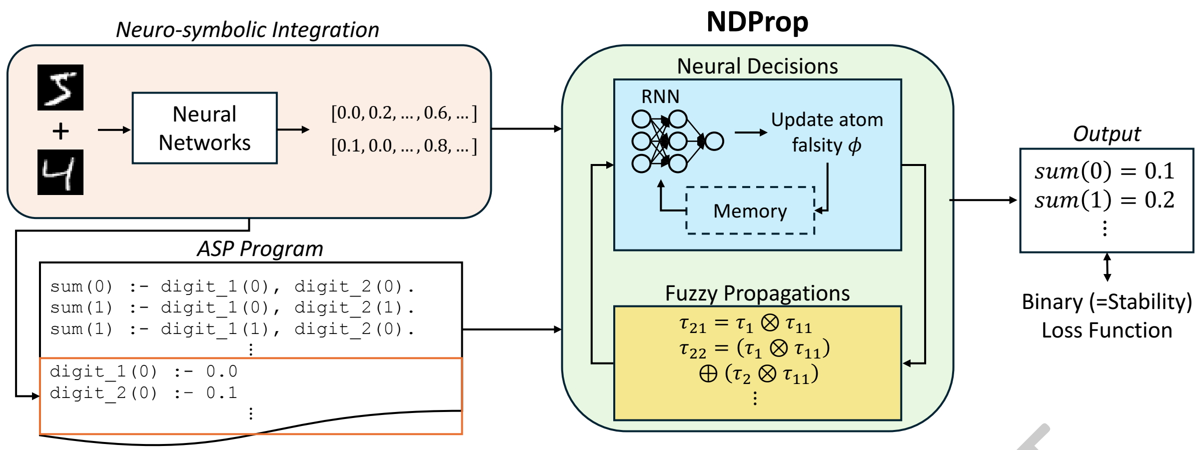

Figure 2: NDProp pipeline. The neuro-symbolic integration is added for specific instantiations, and the arrow leading from the neuro-symbolic block to the ASP program is added only when the backbone neural networks are being trained.

which serves as the atom's soft truth value indicating whether the atom should be included in the answer set or not. Importantly, this becomes binary iff (τ_i, ϕ_i) is $(1, 0)$ or $(0, 1)$, meaning that if the variable is either undecided $(0, 0)$ or $(1, 1)$ contradictory, the score will never become binary. We denote by (τ^t, ϕ^t) the degrees at iteration t.

Decision Phase

Decisions update only ϕ while holding τ fixed, as in DProp. Given a t-norm $\otimes$ and t-conorm $\oplus$, the undecided degree μ_i for atom a_i is defined as

$$\mu_i^t := 1 - (\tau_i^t \oplus \phi_i^t) = (1 - \tau_i^t) \otimes (1 - \phi_i^t) \in [0, 1].$$

We then produce decisions via a neural decision module, which in this case is a Recurrent Neural Network (RNN) that takes the current input and hidden state s^{t-1} and yields the decision logits d^t and updated hidden state s^t as follows:

$$(d^t, s^t) = \mathrm{RNN}_{\mathrm{dec}}\big([\tau^t \,\|\, \phi^t \,\|\, \mu^t], s^{t-1}\big),$$

where $\|$ denotes the concatenation operation. RNNs retain historical information with the hidden states s^t, making it suitable for our approach that requires producing decisions over multiple steps. We then apply a sigmoid function $\sigma : \mathbb{R} \to (0, 1)$ on the decision logits to yield the decision scores:

$$\delta^t = \sigma\big(d^t\big).$$

The falsity degrees move toward these selections via

$$\phi^{t+1} = \phi^t + (\mu^t \odot \delta^t), \tag{2}$$

with $\odot$ denoting elementwise multiplication. Note that ϕ^{t+1} remains in $[0, 1]$, since $\mu^t \leq 1 - \phi^t$ and $\delta^t \in (0, 1)$. Because μ^t becomes zero for already decided atoms, this additive rule directly mirrors the discrete falsity decision ($F^t = F^{t-1} \cup E$ cf. Remark 1) when (τ^t, ϕ^t) and δ^t are binary.

Propagation Phase

Propagation updates only τ while holding ϕ fixed, as in DProp. Given a t-norm $\otimes$ and t-conorm $\oplus$, for each rule r_j, its rule body contributes support

$$\rho_j(\tau, \phi) := \bigotimes_{\ell \in B(r_j)} \mathrm{val}(\ell).$$

Each head atom a_i aggregates these supports via the soft immediate consequence operator

$$(\mathcal{C}_{P,\phi}(\tau))_i := \bigoplus_{j : H(r_j) = a_i} \rho_j(\tau, \phi).$$

Given ϕ^{t+1}, we update τ from the current state τ^t by iterating $\mathcal{C}_{P,\phi^{t+1}}$ until convergence:

$$\tau^{t+1} = \mathcal{C}^{\infty}_{P,\phi^{t+1}}(\tau^t).$$

In practice, we stop the iteration early when the maximum change between successive applications of $\mathcal{C}_{P,\phi^{t+1}}$ falls below a fixed convergence threshold, or the iteration reaches a given pre-defined value.

Crisp Case Recovers DProp

We give the proof that DProp can be emulated with a specific constraint on NDProp, showing that NDProp is a differentiable extension of DProp.

Definition 2 (Set induced by ψ)**.** *Let $\psi \in [0, 1]^n$ be a real valued vector with the i-th value ψ_i corresponding to atom a_i. The set induced by ψ is defined as*

$$[\psi] := \{\, a_i \mid \psi_i = 1, 1 \leq i \leq n \,\}.$$

In the crisp case, the operator $\mathcal{C}_{P,\phi}$ makes the same atoms true as $C_{P,\mathcal{A}\setminus F}$, that is:

Lemma 5. *Let $(\tau, \phi) \in \{0, 1\}^n \times \{0, 1\}^n$, $T = [\tau]$ and $F = [\phi]$. Then, $[\mathcal{C}_{P,\phi}(\tau)] = C_{P,\mathcal{A}\setminus F}(T)$.*

Proof. With binary (τ, ϕ), a positive literal evaluates to $\tau_i \in \{0, 1\}$ and a default-negated literal evaluates to $\phi_i \in \{0, 1\}$ for atom a_i. Thus each rule r_j's support $\rho_j(\tau, \phi)$ is 1 exactly when $B^+(r_j) \subseteq [\tau]$ and $B^-(r_j) \subseteq [\phi]$. Aggregating supports via t-conorm $\oplus$ yields exactly the heads derived by $C_{P,\mathcal{A}\setminus F}(T)$, so the induced true set coincides. $\square$

Then we obtain the following property:

Proposition 1 (Crisp Case). *Let $(\tau, \phi) \in \{0,1\}^n \times \{0,1\}^n$, $\delta \in \{0,1\}^n$, and propagation compute the fixpoint of $\mathcal{C}_{P,\phi}$. Then, one update of NDProp coincides with one iteration of DProp.*

Proof. With binary (τ, ϕ), the set induced by undecided degree $\mu = (1-\tau) \otimes (1-\phi)$ corresponds to $U = \mathcal{A} \setminus (T \cup F)$ when $T = [\tau]$ and $F = [\phi]$. Thus, the update $\phi' = \phi + (\mu \odot \delta)$ coincides with $F' = F \cup (U \cap [\delta])$. From Remark 1, deciding falsity of a set of atoms is fine, thus this exactly coincides with the decision step of DProp when the falsity of a set of atoms $U \cap [\delta]$ is decided.

Since $(\tau, \phi') \in \{0,1\}^n \times \{0,1\}^n$, from Lemma 5 $[\mathcal{C}_{P,\phi'}(\tau)] = C_{P,\mathcal{A}\setminus F'}(T)$. When $\mathcal{C}_{P,\phi'}(\tau)$ is applied until a fixpoint, this exactly coincides with the propagation step of DProp. Furthermore, when μ, δ is binary, τ and ϕ are guaranteed to be binary as well. $\square$

4.2 Loss Alignment

We prove that enforcing NDProp to output binary values is equivalent to enforcing stability. Here, $\psi' \leq \psi$ is equivalent to $\forall i \ (\psi'_i \leq \psi_i)$ and **0** and **1** are n-dimensional vectors of all 0s and all 1s, respectively.

Lemma 6. *Let $\phi \leq \phi'$. Then, $\mathcal{C}_{P,\phi}(\tau) \leq \mathcal{C}_{P,\phi'}(\tau)$, and consequently $\mathcal{C}^{\infty}_{P,\phi}(\boldsymbol{0}) \leq \mathcal{C}^{\infty}_{P,\phi'}(\boldsymbol{0})$.*

Proof. For fixed τ, negative literals evaluate as $\mathrm{val}(not\ a_i) = \phi_i$. Since the t-norm $\otimes$ and t-conorm $\oplus$ are monotone in each argument, $\phi \leq \phi'$ implies each rule support $\rho(\tau, \phi) \leq \rho(\tau, \phi')$, hence $\mathcal{C}_{P,\phi}(\tau) \leq \mathcal{C}_{P,\phi'}(\tau)$. Because $\mathcal{C}_{P,\phi} \leq \mathcal{C}_{P,\phi'}$ pointwise, iterating from **0** yields $\mathcal{C}^{k}_{P,\phi}(\mathbf{0}) \leq \mathcal{C}^{k}_{P,\phi'}(\mathbf{0})$ for all k, and taking the limit gives $C^{\infty}_{P,\phi}(\mathbf{0}) \leq C^{\infty}_{P,\phi'}(\mathbf{0})$. $\square$

Lemma 7. *Let $\tau^{t-1} \leq \mathcal{C}^{\infty}_{P,\phi^t}(\boldsymbol{0})$. Then, $\mathcal{C}^{\infty}_{P,\phi^t}(\tau^{t-1}) = \mathcal{C}^{\infty}_{P,\phi^t}(\boldsymbol{0})$.*

Proof. The operator $\mathcal{C}_{P,\phi^t}$ is monotone in τ by monotonicity of $\otimes$ and $\oplus$. Since $\tau^{t-1} \leq \mathcal{C}^{\infty}_{P,\phi^t}(\mathbf{0})$ and $\mathcal{C}^{\infty}_{P,\phi^t}(\mathbf{0})$ is a fixpoint of $\mathcal{C}_{P,\phi^t}$, for every $n \geq 0$ we have $\mathcal{C}^{n}_{P,\phi^t}(\mathbf{0}) \leq \mathcal{C}^{n}_{P,\phi^t}(\tau^{t-1}) \leq \mathcal{C}^{\infty}_{P,\phi^t}(\mathbf{0})$. Taking limits yields $\mathcal{C}^{\infty}_{P,\phi^t}(\tau^{t-1}) = \mathcal{C}^{\infty}_{P,\phi^t}(\mathbf{0})$. $\square$

Theorem 2 (Binary terminal state yields stability). *Let P be a finite program and $S \subseteq \mathcal{A}_P$, where $\mathcal{A}_P$ is the set of atoms occurring in P. Then S is a stable model of P if and only if propagation is applied until convergence with the resulting membership score $p_i \in \{0,1\}$ and $S = [p]$.*

Proof. ($\Leftarrow$) Let $p \in \{0,1\}^n$ be the membership score obtained from $(\tau^\infty, \phi^\infty)$, and let $T = [\tau]$ and $F = [\phi]$. Since $p_i = \frac{1}{2}(\tau_i + 1 - \phi_i)$ and $\tau_i^\infty, \phi_i^\infty \in [0,1]$, $p_i \in \{0,1\}$ implies $(\tau_i^\infty, \phi_i^\infty) \in \{(1,0),(0,1)\}$, and thus $T \cap F = \emptyset$, $T \cup F = \mathcal{A}$ and $T = [p]$. ϕ increases monotonically as there are no negatives in Equation (2). As propagation is applied until convergence $\tau^\infty = \mathcal{C}^{\infty}_{P,\phi^\infty}(\tau^{\infty-1})$, and from Lemmas 6 and 7, $\tau^\infty = \mathcal{C}^{\infty}_{P,\phi^\infty}(\mathbf{0})$. Since $[\mathbf{0}] = \emptyset$, from Lemma 5, $[\tau^\infty] = [\mathcal{C}^{\infty}_{P,\phi^\infty}(\mathbf{0})] = C^{\infty}_{P,\mathcal{A}\setminus F}(\emptyset) = C^{\infty}_{P,T}(\emptyset)$. From Lemma 1, $[p] = [\tau^\infty]$ is thus a stable model of P.

($\Rightarrow$) Let τ^∞ and ϕ^∞ be the true and false degrees after the final iteration. From Proposition 1, if we set the decision scores to be binary and make the propagation phase compute until a fixpoint of $\mathcal{C}_{P,\phi}$, the iteration coincides with a DProp iteration. From Theorem 1, there exists a successful DProp computation that ends with the stable model S, and since τ, ϕ is binary, $T \cap F = [\tau^\infty] \cap [\phi^\infty] = \emptyset$ and $U = \emptyset$, $(\tau^\infty, \phi^\infty) \in \{(\mathbf{1},\mathbf{0}),(\mathbf{0},\mathbf{1})\}$ and thus p is binary and $[p] = S$. $\square$

From Theorem 2, we can conclude that when the output p of NDProp is trained with a loss function that enforces binary outputs, it inherently forces the NDProp to learn to output stable models, given that the final propagation is done until convergence. This holds even with a single iteration, since the decision step can falsify a set of atoms at once, c.f. Remark 1.

5 Experiments

In this section, we perform experiments to answer the following three questions:

(**Q1**) Can NDProp learn to solve ASP problems?

(**Q2**) Can NDProp compete with existing ASP-based neuro-symbolic frameworks?

(**Q3**) What are novel settings in explicit need of NDProp?

5.1 Random Logic Programs

To assess NDProp's behavior on unstructured problems, we first evaluate it on randomly generated logic programs. Random instances are utilized to evaluate neural approaches to symbolic solving, e.g., in Boolean Satisfiability [Selsam *et al.*, 2019], where they are used to evaluate whether learned models can learn to solve problems that are generally difficult to find heuristics for.

We follow the linear model $L(N2, c_1, c_2)$ for random negative two-literal (N2L) programs [Wang *et al.*, 2015]. For n atoms, each pure rule $a \leftarrow not\, b$ (with $a \neq b$) appears independently with probability c_1/n and each contradiction rule $a \leftarrow not\, a$ appears with probability c_2/n. Random N2L programs are attractive for an experimental setting as they are easy to produce, while being considerably difficult, as it includes NP-complete instances. The same setting has been used in other work that attempted to apply GNNs to Answer Set Computation [Ielo and Ricca, 2021].

As supported models of N2L programs are known to be stable models as well, we look at problems that are not in this regard. Specifically, we look at the k-$LP(N, L)$ generator that has been studied for phase transitions in ASP [Zhao and Lin, 2003]. We use the 3-LP generator, which has been shown to be hard around $L/N = 5$, where L and N correspond to the number of rules and atoms respectively.

Setup

We focus on having three datasets per generator (N2L, and 3-LP) with different number n of atoms. Specifically, we use easy (5–10), medium (11–50), and hard (51-100). We specifically use the hard set to determine whether the model is able to generalize to bigger problems. Each dataset consists of 1000 training and 100 validating instances and 100 testing instances. As the baseline, we implement RDProp,

	Addition			Addition2			Add2x2			Apply2x2			Member3			Member5		
	NDProp	NASP	SLASH	NDProp	NASP	SLASH	NDProp	NASP	SLASH	NDProp	NASP	SLASH	NDProp	NASP	SLASH	NDProp	NASP	SLASH
Train	**160**	1612	430	**235**	TO	2374	**692**	TO	TO	346	385	**166**	**110**	2829	TO	**969**	TO	TO
Test	**0.37**	24.25	9.94	**1.28**	33.90	3.73	**1.15**	41.59	3.75	**1.52**	5.31	4.32	**0.79**	5.13	3.08	**4.40**	496.4	6.38
R.Acc	**99.02**	98.54	98.91	98.92	98.46	**99.00**	**99.04**	98.59	98.94	**100.0**	**100.0**	**100.0**	**97.25**	96.81	97.16	**96.97**	11.56	8.92
I.Acc	**98.20**	97.11	97.83	96.10	94.57	**96.47**	**95.73**	93.70	95.17	**100.0**	**100.0**	**100.0**	**98.97**	98.57	98.73	**97.53**	59.67	60.30

Table 1: Neuro-symbolic MNIST results for NDProp, NeurASP (NASP) and SLASH, all trained for 10 epochs. Metrics are Train: Training time, Test: Testing time, R.Acc: Recognition Accuracy (%), and I.Acc: Inference accuracy (%).

Method	Split	N2L			3-LP		
		E	M	H	E	M	H
NDProp	E	**100**	<u>80</u>	<u>17</u>	**100**	<u>61</u>	<u>12</u>
	M	**100**	**96**	**42**	**100**	**86**	**37**
Random	–	77	1	0	56	0	0
RDProp-1	–	40	9	0	7	0	0
RDProp-10	–	96	37	4	60	3	0
RDProp-100	–	**100**	68	15	99	14	0

Table 2: Percentage of instances for which a stable model was found, when trained on the easy and medium sets of each corresponding dataset. RDProp-K corresponds to DProp with random decisions, with a total of K restarts. Best accuracy is in bold, and second best accuracy is underlined.

which is DProp with random decisions. We test RDProp with 1, 10, and 100 restarts, denoted as RDProp-1, RDProp-10, and RDProp-100 respectively. We compare NDProp with RDProp to see whether NDProp is able to learn the decision making process to outperform random decisions.

For NDProp, we train for 1000 epochs, with the setup of an iteration of 10 (50 at test time), a hidden dimension of 32, a logical dimension of 32, a learning rate of 1×10^{-3}, and the Gödel t-norm. The logical dimension means the model maintains multiple logical states at once, and we select the state whose output is closest to binary at the end. For the loss function, we use the smallest Binary Cross Entropy (BCE) loss over all stable models as the objective function.

Results

Table 2 provides the percentage of problems in each test set solved. The results show that NDProp is effectively learning the decision dynamics for solving random problems. Specifically, NDProp is shown to completely outperform RDProp in the 1-shot setting, with performance on medium and easy training datasets exceeding that of 100-shot RDProp. This empirically supports our claim that NDProp is an extension of DProp, and that it can learn the decision heuristics for finding stable models for logic programs.

5.2 Neuro-Symbolic Tasks

As NDProp can approximate answer sets differentiably, it is natural to combine with existing neural networks based architectures to perform neuro-symbolic inference. In all settings, the input consists of one or more MNIST images, and supervision is provided only through arithmetic relations between the underlying concepts, rather than through per-image labels [Manhaeve *et al.*, 2021; Gaunt *et al.*, 2017; Tsamoura *et al.*, 2021]. The goal is therefore to jointly learn perception using symbolic reasoning from weak, relational supervision. This will allow us to evaluate the merits of using an end-to-end differentiable system, as opposed to a system that relies on separate training of the perception module and the reasoning module.

ASP Encodings. In contrast to prior neuro-symbolic ASP systems that use neural predicates to interface with perception modules [Yang *et al.*, 2020], we adopt a different strategy. We embed the output of neural models as fuzzy facts into a fixed ASP program that encodes the arithmetic constraints (Figure 2). While this may seem to force NDProp to perform as a FASP solver, the objective is to produce binary outputs, explicitly forcing the backbone perception model to predict binary values as well.

Setup

In this experiment, we perform evaluations on six variants of the MNIST-based arithmetic reasoning tasks that are commonly used as neuro-symbolic benchmarks [Yang *et al.*, 2020; Manhaeve *et al.*, 2021]: 1 and 2 digit addition, Add2x2, Apply2x2 (with handwritten operator images) and 3 and 5 digit membership. For NDProp, we use the setup with iteration of 1, hidden dimension of 8, logical dimension of 2, learning rate of 1×10^{-3}, and product t-norm.

We compare the performance of NDProp with ASP-based neuro-symbolic systems: NeurASP [Yang *et al.*, 2020] and SLASH [Skryagin *et al.*, 2023]. Experiments are conducted with a maximum of 10 epochs for all approaches, with a timeout set to 3000 seconds; the training is cut off when the budget is reached, and accuracy is checked at the end of the epoch. The training setup used is identical to those given in the original papers.

Results

Table 1 reports recognition accuracy, inference accuracy (accuracy of final output), and runtime (seconds) for both training and testing splits, averaged over 3 runs. Across six neuro-symbolic tasks, NDProp achieves higher digit and inference accuracy than NeurASP and SLASH for five, with the exception of the Addition2 task, where SLASH has a slightly higher recognition and inference accuracy. Empirically, this shows the benefits of using end-to-end trainable systems: jointly optimizing perception and reasoning leads to more cohesive and consistent outputs, leading to performance similar to or better than systems that rely on symbolic solvers. This is similar to findings in prior works that use differentiable computation of supported models [Takemura and Inoue, 2024], yielding faster training time and higher accuracy in certain setups.

Furthermore, the training time for NDProp is significantly faster than NeurASP and SLASH. Specifically, for the Member5 task, both NeurASP and SLASH fail to train even one epoch within the time budget, leading to very low accuracy. On the other hand, NDProp has a longer training time for Apply2x2; this is thought to be because there are extremely many grounded atoms in this task, which have to be processed at once for the forward pass of NDProp, leading to a much higher computational cost.

The difference is more prominent in the testing speed, where NDProp is an average of 41.8 times faster than NeurASP, and 6.8 times faster than SLASH. This is mainly due to the GPU capabilities of NDProp; GPUs allow for higher parallelization, allowing our model to process multiple instances with one forward pass, even during testing. In comparison, NeurASP can only process one instance at a time, while SLASH can process multiple instances only at training time.

5.3 Reasoning with Incomplete Data

In this experiment, we look at settings where data is incomplete. Given the predicted outputs of models with incomplete training, NDProp can add a layer of reasoning on top, yielding a solution that is both correct in terms of the given requirements and is able to account for the information given by the incomplete data.

Setup

Visual sudoku is a task where each cell has an image corresponding to a number, and the goal is to find the correct solution for the given grid, with MNIST being the most common image used. In this experiment, we hypothesize a setting where the neural network is fixed; we pretrain the model on MNIST images, and reason on its predictions to find the solution to the task. By having models that are trained on less data, we can simulate a situation in which the given predictions are not correct 100% of the time.

We specifically focus on the 4x4 setting with each cell containing an image with a number between 0-4 (0 for empty cell). We vary the amount of labeled training data for the MNIST predictor (100%, 50%, 10%, 5% and 1%) with 500 steps of training, to test whether the framework can perform under imperfect settings. We compare against two baselines that use the same predictions:

- **ASP:** Give the current predictions other than 0 as a fact, and solve the ASP program.
- **Neural:** Train a neural classifier that takes as input the predicted values of each cell (0 for empty cells), and predict the digit in each cell. This is trained until convergence.

In this setting, our framework is used in a separate way to the neuro-symbolic tasks. As the input facts are not trained to become binary, we use the facts only as a heuristic to guide NDProp to a solution. This is provided through the RNN's initial hidden states s^0 for each atom, which is computed by a single-layer MLP. We train for 100 epochs (which is sufficient for the model to converge) using NDProp with iterations of 3, hidden dimension of 8, logical dimension of 2, learning rate of 1×10^{-3}, and Gödel t-norm.

	100%	50%	10%	5%	1%
NDProp	**95.33**	**93.80**	**92.43**	**88.07**	**74.30**
ASP	92.77	92.40	90.03	85.60	70.47
Neural	84.07	84.17	83.23	80.87	73.30
Digit Acc.	99.25	99.24	99.12	98.58	96.89

Table 3: Visual sudoku accuracy (%) with different digit classifier training sizes. Best score is bolded, and the second best is underlined. The row of Digit Acc. shows the prediction accuracy on MNIST (digits 0-4) of the backbone neural model.

Results

Table 3 shows the percentage of problems where all digits for each cell were given correctly over 3 runs. In every setup, we can see that NDProp performs the best, with constant gains against the second best method. An interesting trend that can be observed is that when the amount of data goes up, the ASP baseline performs much better. This is because when the predictions are correct, a complete reasoning mechanism will always get the correct solution.

On the other hand, while increasing the performance of the digit classifier does not have too dramatic of an impact for the neural method, we can see that the deterioration in performance is much smaller. This shows that neural models in general are very strong with imperfect inputs, showing merits over classical methods that completely plummet at 1%. NDProp is able to achieve the best of both worlds: when the predictions are accurate, NDProp can properly reason on them to get the correct solution, and when the predictions are less accurate, it uses the information to find a solution that is most compatible with the predictions.

6 Related Work

6.1 ASP Solvers

Classical ASP systems such as smodels [Niemelä and Simons, 1997] and DLV [Leone *et al.*, 2006] perform 2-valued branching search by first examining the true case and then the false case (or vice-versa). Later systems such as clingo [Gebser *et al.*, 2016] and WASP [Alviano *et al.*, 2013] adopt CDCL-inspired architectures that combine deterministic propagation with branching decisions, allowing for more efficient search and learning from conflicts.

DProp differs from these systems in that it focuses on deciding only the falsity of atoms and propagating the true atoms. Furthermore, this design treats stability as a result of computation: when the solution is contradiction-free, stability is guaranteed (Theorem 1). These features enable a differentiable extension by NDProp, whose crisp case provably retains the same theoretical properties as DProp. Such differentiable extensions are not easily achievable with existing ASP solvers.

Fuzzy answer set programming is an extension of ASP to handle degrees of truth using fuzzy logic [Nieuwenborgh *et al.*, 2006; Alviano and Peñaloza, 2013]. These approaches adapt the stable model semantics to allow the use of fuzzy truth values, often employing t-norms to define rule satisfaction and model construction.

6.2 Fixpoint Semantics

A wide range of fixpoint semantics for ASP have been proposed in the literature. Earlier works formalized nondeterministic fixpoint computation in various ways. For example, Saccà and Zaniolo [1990] incorporate a backtracking mechanism into bottom-up search, and Inoue and Sakama [1996] characterize a parallel procedure to compute stable models [Inoue *et al.*, 1992; Shirai and Hasegawa, 2004] by a fixpoint operator with case splitting and constraints. Efficient search strategies to limit the number of case splitting during fixpoint computation could be learned by machine learning techniques, but there has been no such attempt yet.

Fages [1991] shows a non-contradictory strategy for justification-based derivation like truth maintenance system yields a stable model at a fixpoint. Similarly, the notion of persistent computation by Liu *et al.* [2010] selects applicable rules to compute stable models, and the idea leads to ASP solvers GASP [Dal Palù *et al.*, 2009] and ASPeRiX [Lefèvre *et al.*, 2017]. Such a selection strategy is similar to DProp, which captures stable models as computations of iterations using propagation and choice. The main difference is that in GASP and ASPeRix, the choice is performed on all negative body literals of a chosen rule, while DProp performs decisions on any undecided atom; the latter eases extension into the continuous domain.

Several works characterize stable models via alternating fixpoints and approximation operators [Fitting, 2002]. The alternating fixpoint theory [Przymusinski, 1991] defines stable models as fixpoints of an operator that alternates between deriving positive and negative information. Denecker *et al.* [2000] develop an approximation theory that captures various non-monotonic semantics including stable models, through the use of approximating operators and their fixpoints. These frameworks yield stable models in different ways than DProp, which directly produces stable models through propagation–decision steps.

6.3 Neural Answer Set Solving

Neurosymbolic integration of ASP is a field being actively explored. NeurASP [Yang *et al.*, 2020] finds stable models of programs that contain neural atoms using symbolic solvers, and uses the results to train the neural perception model. SLASH [Skryagin *et al.*, 2023] extends this system to support joint distributions, richer queries, and scalable inference.

In contrast to works that use classical ASP solvers, several works have explored the use of differentiable approaches for solving ASP problems. One approach to accomplish this is by using deep learning with discrete solvers such as clingo. Differentiable ASP/SAT [Nickles, 2018] compute stable models with ASP solvers utilizing derivatives of a cost function, and Dodaro *et al.* [2022] explore the use of deep learning to generate domain-specific solvers for WASP.

Another differentiable approach is to directly compute supported or stable models in vector spaces. Aspis *et al.* [2020] use a matrix representation of logic programs to compute supported models, and solve a root-finding problem using Newton's method. Takemura and Inoue [2022] propose another matricized method by defining a cost function that yields a supported model when the cost is minimized to be zero. Sato *et al.* [2026] extend such matricized methods by encoding loop formulas in vector spaces, so that stable models can be obtained by minimizing a single objective. Ielo and Ricca [2021] use GNNs to solve negative two literal programs, which are known to have the property that all supported models are stable models. GNNs have been used in Boolean constraint solvers other than ASP, such as Boolean Satisfiability [Selsam *et al.*, 2019] and Maximum Satisfiability [Moriyama and Inoue, 2025], with the latter specifically showing that t-norms are beneficial for developing differentiable solvers. Our approach is classified as a differentiable approach as well, with the main differences being how the solutions are being computed: prior works use the cost function to optimize the variable assignments, while NDProp uses a propagation-decision process to search for stable models, with the decision heuristics being learned. Additionally, our work requires only binarization to get stable models rather than supported models, while prior works require the use of loop formulas or do not consider it at all.

7 Conclusion

We have presented decision–propagation (DProp), a novel method for computing stable models. DProp alternates between falsity decisions and truth propagations, and a successful DProp computation is shown to capture the stable model semantics. We then proposed NDProp, a differentiable extension of DProp that uses neural decisions and fuzzy propagations, and show that it coincides with DProp at the binary limit. Experiments on random logic programs and neuro-symbolic benchmarks show that NDProp is able to learn to solve ASP problems, and can be combined with neural models to train end-to-end. Compared to existing approaches that use symbolic reasoning, our approach incorporates learning in the reasoning step as well, resulting in a higher accuracy. The results display the power of a differentiable and learnable solver in pipelines where reasoning is required.

Throughout the experiments, we have noticed that some t-norms work better than others in each setting. An extensive study of t-norms in particular settings is left out for future work, as each t-norm has different properties that are beneficial in different settings [Eiter *et al.*, 2025]. Another direction to explore is whether the semantics of NDProp aligns with FASP. At the moment, NDProp is aimed at normal logic programs, but with modifications, it could be used to solve fuzzy variants. Finally, extending the framework to support often used constructs such as constraints, choices and aggregates is left for future work. This would allow for use in more complex neuro-symbolic tasks, and would be an important step towards applying the framework to real-world scenarios.

Acknowledgements

This work has been supported by JSPS KAKENHI Grant Number JP25K03190, JST CREST Grant Number JPMJCR22D3 and Grant-in-Aid for JSPS Fellows Grant Number JP26KJ1236, Japan. This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/COE12. The authors thank colleagues in the KBS Group at TU Wien for helpful discussions.

References

[Alviano and Peñaloza, 2013] Mario Alviano and Rafael Peñaloza. Fuzzy answer sets approximations. *Theory Pract. Log. Program.*, 13(4-5):753–767, 2013.

[Alviano *et al.*, 2013] Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A Native ASP Solver Based on Constraint Learning. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference*, volume 8148 of *LNCS*, pages 54–66. Springer, 2013.

[Aspis *et al.*, 2020] Yaniv Aspis, Krysia Broda, Alessandra Russo, and Jorge Lobo. Stable and supported semantics in continuous vector spaces. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 59–68, 2020.

[Barbara *et al.*, 2023] Vito Barbara, Massimo Guarascio, Nicola Leone, Giuseppe Manco, Alessandro Quarta, Francesco Ricca, and Ettore Ritacco. Neuro-Symbolic AI for Compliance Checking of Electrical Control Panels. *Theory Pract. Log. Program.*, 23(4):748–764, 2023.

[Borroto *et al.*, 2025] Manuel Borroto, Antonio Ielo, Giuseppe Mazzotta, and Francesco Ricca. Answer Set Programming and Neurosymbolic AI: Applications and Future Perspectives (Invited Talk). In *Hybrid Models for Coupling Deductive and Inductive Reasoning*, pages 1–21. Springer, 2025.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Dal Palù *et al.*, 2009] Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. GASP: Answer Set Programming with Lazy Grounding. *Fundam. Informaticae*, 96(3):297–322, 2009.

[d'Avila Garcez and Lamb, 2023] Artur d'Avila Garcez and Luís C. Lamb. Neurosymbolic AI: the 3rd wave. *Artif. Intell. Rev.*, 56(11):12387–12406, 2023.

[Denecker *et al.*, 2000] Marc Denecker, Victor Marek, and Mirosław Truszczyński. *Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning*, pages 127–144. Springer US, Boston, MA, 2000.

[Dodaro *et al.*, 2022] Carmine Dodaro, Davide Ilardi, Luca Oneto, and Francesco Ricca. Deep Learning for the Generation of Heuristics in Answer Set Programming: A Case Study of Graph Coloring. In Georg Gottlob, Daniela Inclezan, and Marco Maratea, editors, *Logic Programming and Nonmonotonic Reasoning - 16th International Conference*, volume 13416 of *LNCS*, pages 145–158. Springer, 2022.

[Eiter *et al.*, 2022] Thomas Eiter, Nelson Higuera, Johannes Oetsch, and Michael Pritz. A Neuro-Symbolic ASP Pipeline for Visual Question Answering. *Theory Pract. Log. Program.*, 22(5):739–754, 2022.

[Eiter *et al.*, 2025] Thomas Eiter, Katsumi Inoue, Nelson Higuera, and Sota Moriyama. T-norm selection for object detection in autonomous driving with logical constraints. In *Advances in Neural Information Processing Systems*, volume 38, pages 86906–86933, 2025.

[Eiter *et al.*, 2026] Thomas Eiter, Katsumi Inoue, and Sota Moriyama. Neural Decision-Propagation for Answer Set Programming. *CoRR*, abs/2605.01797, 2026.

[Evans and Grefenstette, 2018] Richard Evans and Edward Grefenstette. Learning Explanatory Rules from Noisy Data. *J. Artif. Intell. Res.*, 61:1–64, 2018.

[Fages, 1991] François Fages. A New Fixpoint Semantics for General Logic Programs Compared with the Well-Founded and the Stable Model Semantics. *New Gener. Comput.*, 9(3/4):425–444, 1991.

[Fitting, 2002] Melvin Fitting. Fixpoint semantics for logic programming a survey. *Theor. Comput. Sci.*, 278(1-2):25–51, 2002.

[Gaunt *et al.*, 2017] Alexander L. Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. Differentiable Programs with Neural Libraries. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1213–1222. PMLR, 2017.

[Gebser *et al.*, 2016] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory Solving Made Easy with Clingo 5. In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming*, volume 52 of *OASIcs*, pages 2:1–2:15, 2016.

[Ielo and Ricca, 2021] Antonio Ielo and Francesco Ricca. Answer Set Computation of Negative Two-Literal Programs Based on Graph Neural Networks: Preliminary Results. In Stefania Monica and Federico Bergenti, editors, *Proceedings of the 36th Italian Conference on Computational Logic*, volume 3002, pages 188–195. CEUR-WS.org, 2021.

[Inoue and Sakama, 1996] Katsumi Inoue and Chiaki Sakama. A Fixpoint Characterization of Abductive Logic Programs. *J. Log. Program.*, 27(2):107–136, 1996.

[Inoue *et al.*, 1992] Katsumi Inoue, Miyuki Koshimura, and Ryuzo Hasegawa. Embedding Negation as Failure into a Model Generation Theorem Prover. In Deepak Kapur, editor, *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction*, volume 607 of *LNCS*, pages 400–415. Springer, 1992.

[Lefèvre *et al.*, 2017] Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia. ASPeRiX, a first-order forward chaining approach for answer set computing. *Theory Pract. Log. Program.*, 17(3):266–310, 2017.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[Lifschitz, 2008] Vladimir Lifschitz. What Is Answer Set Programming? In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1594–1597. AAAI Press, 2008.

[Liu *et al.*, 2010] Lengning Liu, Enrico Pontelli, Tran Cao Son, and Miroslaw Truszczynski. Logic programs with abstract constraint atoms: The role of computations. *Artif. Intell.*, 174(3-4):295–315, 2010.

[Manhaeve *et al.*, 2021] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artif. Intell.*, 298:103504, 2021.

[Moriyama and Inoue, 2025] Sota Moriyama and Katsumi Inoue. Graph-based attention for differentiable maxsat solving. In *Advances in Neural Information Processing Systems*, volume 38, pages 137740–137768, 2025.

[Nickles, 2018] Matthias Nickles. Differentiable SAT/ASP. In Elena Bellodi and Tom Schrijvers, editors, *Proceedings of the 5th International Workshop on Probabilistic Logic Programming*, pages 62–74. CEUR-WS.org, 2018.

[Niemelä and Simons, 1997] Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Logic Programming and Nonmonotonic Reasoning, 4th International Conference*, LNCS, pages 421–430. Springer, 1997.

[Nieuwenborgh *et al.*, 2006] Davy Van Nieuwenborgh, Martine De Cock, and Dirk Vermeir. Fuzzy Answer Set Programming. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Logics in Artificial Intelligence, 10th European Conference*, volume 4160 of *LNCS*, pages 359–372. Springer, 2006.

[Przymusinski, 1991] Teodor C. Przymusinski. Three-Valued Nonmonotonic Formalisms and Semantics of Logic Programs. *Artif. Intell.*, 49(1-3):309–343, 1991.

[Saccà and Zaniolo, 1990] Domenico Saccà and Carlo Zaniolo. Stable Models and Non-Determinism in Logic Programs with Negation. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 205–217. ACM Press, 1990.

[Sato *et al.*, 2026] Taisuke Sato, Akihiro Takemura, and Katsumi Inoue. Towards End-to-End ASP Computation. *Neurosymbolic Artificial Intelligence*, 2:25, 2026.

[Selsam *et al.*, 2019] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT Solver from Single-Bit Supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Shirai and Hasegawa, 2004] Yasuyuki Shirai and Ryuzo Hasegawa. Answer Set Computation Based on a Minimal Model Generation Theorem Prover. In *PRICAI 2004: Trends in Artificial Intelligence, 8th Pacific Rim International Conference on Artificial Intelligence*, volume 3157 of *LNCS*, pages 43–52. Springer, 2004.

[Skryagin *et al.*, 2023] Arseny Skryagin, Daniel Ochs, Devendra Singh Dhami, and Kristian Kersting. Scalable Neural-Probabilistic Answer Set Programming. *J. Artif. Intell. Res.*, 78:579–617, 2023.

[Takemura and Inoue, 2022] Akihiro Takemura and Katsumi Inoue. Gradient-Based Supported Model Computation in Vector Spaces. In Georg Gottlob, Daniela Inclezan, and Marco Maratea, editors, *Logic Programming and Nonmonotonic Reasoning - 16th International Conference*, LNCS, pages 336–349. Springer, 2022.

[Takemura and Inoue, 2024] Akihiro Takemura and Katsumi Inoue. Differentiable Logic Programming for Distant Supervision. In *27th European Conference on Artificial Intelligence*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 1301–1308. IOS Press, 2024.

[Tsamoura *et al.*, 2021] Efthymia Tsamoura, Timothy M. Hospedales, and Loizos Michael. Neural-Symbolic Integration: A Compositional Perspective. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 5051–5060. AAAI Press, 2021.

[van Emden and Kowalski, 1976] Maarten H. van Emden and Robert A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *J. ACM*, 23(4):733–742, 1976.

[Van Gelder, 1993] Allen Van Gelder. The Alternating Fixpoint of Logic Programs with Negation. *J. Comput. Syst. Sci.*, 47(1):185–221, 1993.

[Wang *et al.*, 2015] Kewen Wang, Lian Wen, and Kedian Mu. Random logic programs: Linear model. *Theory Pract. Log. Program.*, 15(6):818–853, 2015.

[Wang *et al.*, 2025] Wenguan Wang, Yi Yang, and Fei Wu. Towards Data-And Knowledge-Driven AI: A Survey on Neuro-Symbolic Computing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 47(2):878–899, 2025.

[Yang *et al.*, 2020] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing Neural Networks into Answer Set Programming. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 1755–1762, 2020.

[Zhao and Lin, 2003] Yuting Zhao and Fangzhen Lin. Answer Set Programming Phase Transition: A Study on Randomly Generated Programs. In Catuscia Palamidessi, editor, *Logic Programming, 19th International Conference*, volume 2916 of *LNCS*, pages 239–253. Springer, 2003.