Computing First-Order Logic Programs by Fibring Artificial Neural Networks

Sebastian Bader — ICCL, TU-Dresden, Germany
sebastian.bader@inf.tu-dresden.de
(supported by the GK334 of the German Research Foundation)
Artur d'Avila Garcez — City University London, UK

Pascal Hitzler — AIFB, Uni-Karlsruhe, Germany

TECHNISCHE UNIVERSITÄT DRESDEN

INTERNATIONAL CENTER FOR COMPUTATIONAL LOGIC

Outline and Motivation

- Logic programs
  - widely accepted paradigm for knowledge representation and reasoning
  - readable by humans
- Fibring artificial neural networks
  - trainable from raw data
  - robust with respect to noisy data
- A neural implementation of a logic program
  - tries to combine the advantages of both paradigms
  - this is only proof of possibility
- Conclusions and further work

Logic Programs & their Semantics

- A simple logic programs $\mathcal{P}$:

$even(0).$	*% 0 is an even number*
$even(s(X)) \leftarrow not\ even(X).$	*% the successor s(X) of a non-even X is even*
$odd(s(X)) \leftarrow even(X).$	*% the successor s(X) of an even X is odd*

- The corresponding Herbrand-base $\mathcal{B}_\mathcal{P}$ and two interpretations:

$$\mathcal{B}_\mathcal{P} = \{even(0), even(s(0)), \ldots, odd(0), odd(s(0)), \ldots\}$$

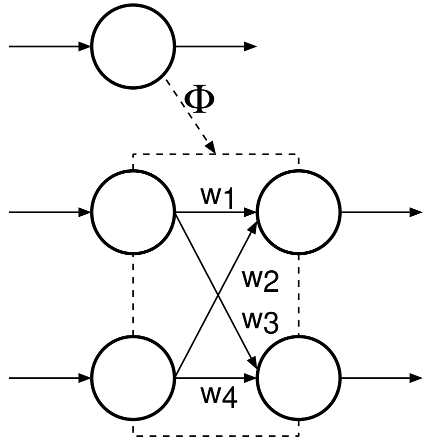
$$I_1 = \{even(0)\} \qquad I_2 = \{even(0), odd(0)\}$$

- The associated immediate consequence operator $T_\mathcal{P}$:

$$I = \{even(0), even(s(s(0)))\}$$

$$T_\mathcal{P}(I) = \{even(0), odd(s(0)), odd(s(s(s(0)))), even(s(s(0))), \ldots\}$$

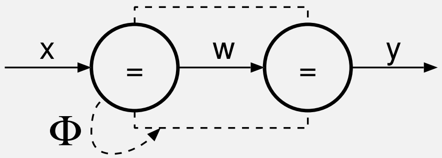
Fibring Artificial Neural Networks [GG04]



- A (fibring) neural network is a set of simple connected units
- Activation of single neurons can influence other weights through a fibring function

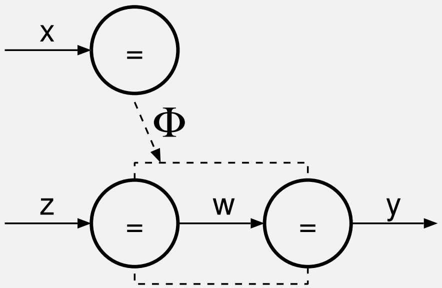
- Example 1 ($y = x^2$):

$$\Phi : (w, x) \mapsto x$$



- Example 2 ($y = (x > 0)?z : 0$):

$$\Phi : (w, x) \mapsto \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

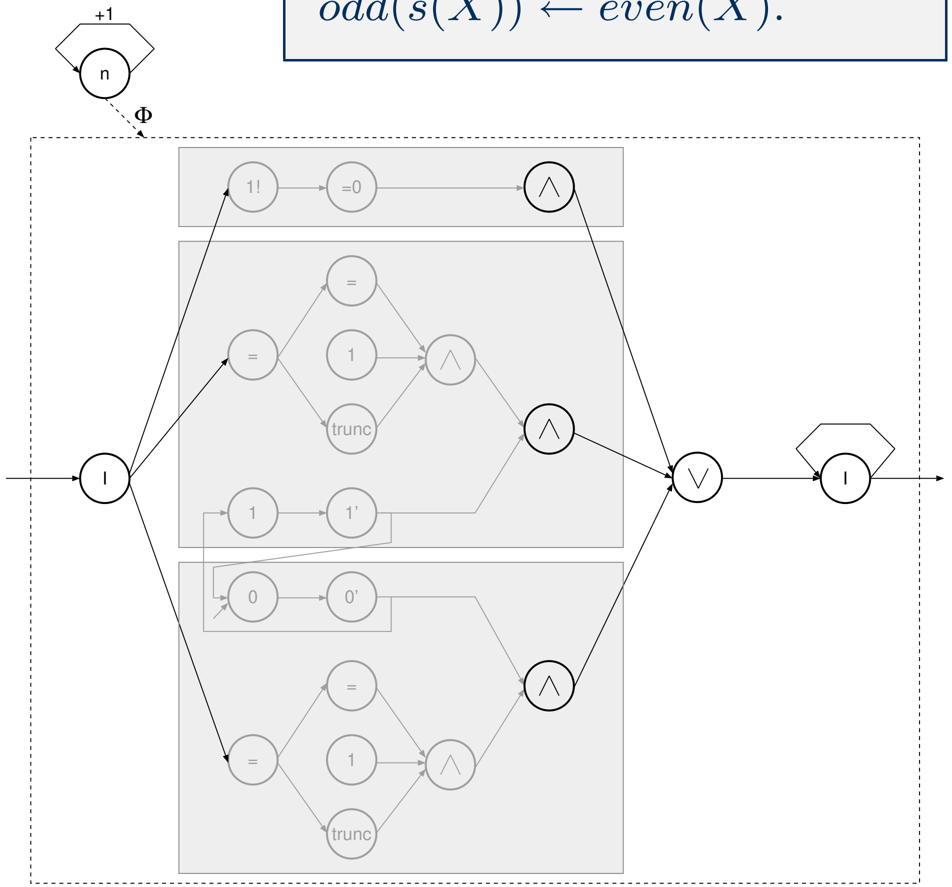


Implementing Logic Programs - Idea

There is ...

- a global counter n fibring the network
- a neuron providing the interpretation I
- a subnetwork for each clause, that fires if the clause outputs the atom n
- a neuron collecting $T_{\mathcal{P}}(I)$

$$even(0).$$
$$even(s(X)) \leftarrow not\ even(X).$$
$$odd(s(X)) \leftarrow even(X).$$



Embedding Interpretations into Real Numbers

- Interpretations are sets of atoms

$$I_1 = \{even(0)\} \qquad\qquad I_2 = \{even(0), odd(0)\}$$

- A (bijective) levelmapping $l : I_{\mathcal{P}} \to \mathbb{N}$ assigns a unique natural number to each atom, hence enumerates the Herbrandbase

$$\mathcal{B}_{\mathcal{P}} = [even(0)_1, odd(0)_2, even(s(0))_3, odd(s(0))_4, \ldots]$$

- We can define an embedding R as follows:

$$R : I_{\mathcal{P}} \to \mathbb{R} : I \mapsto \sum_{A \in I} 4^{-l(A)}$$

$$R(I_1) = 4^{-1} = 0.25 \qquad\qquad R(I_2) = 4^{-1} + 4^{-2} = 0.3125$$

$$(= 0.10000\ldots_4) \qquad\qquad (= 0.11000_4)$$

Prefixing Interpretations

- We can define the prefix function pref on $I_{\mathcal{P}}$ as follows:

$$\text{pref} : I_{\mathcal{P}} \times \mathbb{N} \to I_{\mathcal{P}} : (I, n) \mapsto \{A \in I \mid l(A) \leq x\}$$

$$I = \{even(0)_1, even(s(0))_3, odd(s(0))_4, odd(s(s(0)))_6\}$$

$$\text{pref}(I, 3) = \{even(0)_1, even(s(0))_3\}$$

- pref can be embedded into $\mathbb{R}$:

$$\text{pref} : \mathbb{R} \times \mathbb{N} \to \mathbb{R} : (x, n) \mapsto \frac{\text{trunc}(x \cdot B^n)}{B^n}$$

$$\begin{tikzcd}
I \in I_{\mathcal{P}} \arrow[r, "\text{pref}"] \arrow[d, "R"'] & I' \in I_{\mathcal{P}} \arrow[d, "R"] \\
x \in \mathbb{R} \arrow[r, "\text{pref}"] & x' \in \mathbb{R}
\end{tikzcd}$$

$$x = R(I) \approx 0.269$$

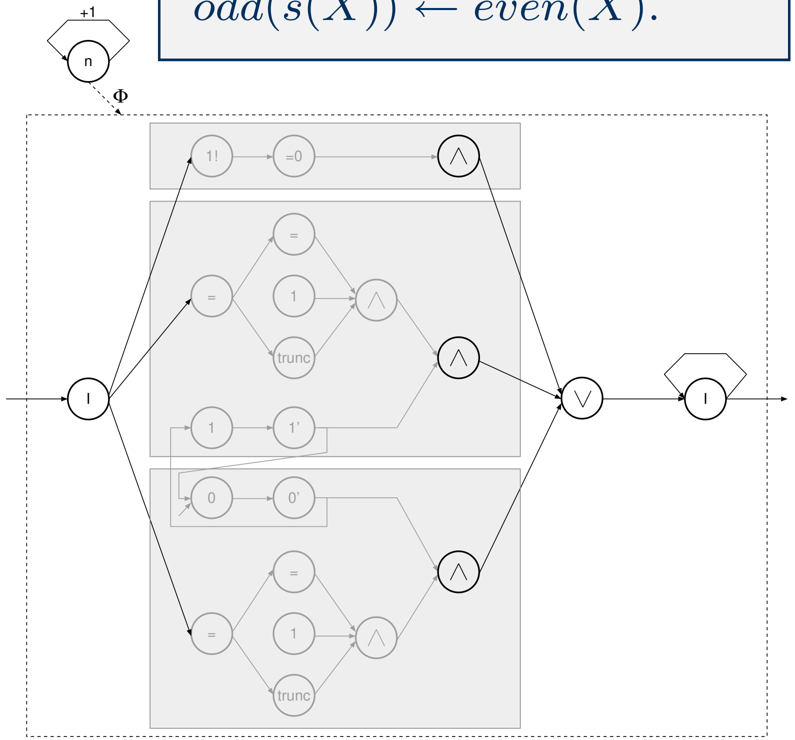
$$\text{pref}(x, 3) = R(\text{pref}(I, 3)) \approx 0.265$$

Implementing Logic Programs - Idea (again)

$$even(0).$$

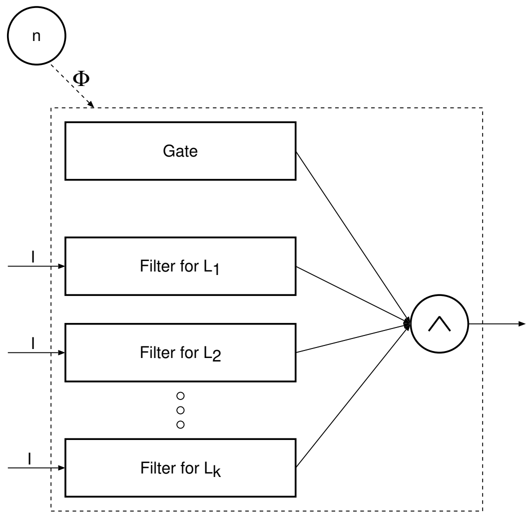
$$even(s(X)) \leftarrow not\ even(X).$$

$$odd(s(X)) \leftarrow even(X).$$



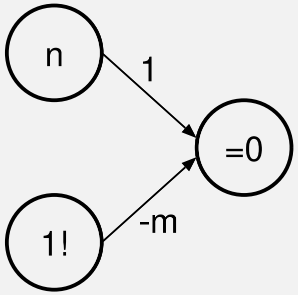
For each clause there is ...

- a 'gate'
- a 'filter' for each body literal
- an 'and'-neuron

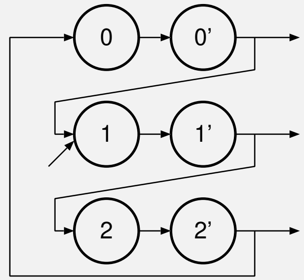


Gates and Filters for Clauses ($H \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_n$)

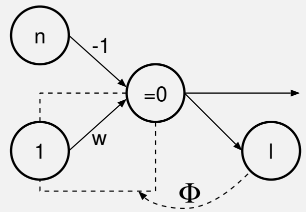
- The "gate" decides whether the clause can output atom n
  - Constant



  - Remainders



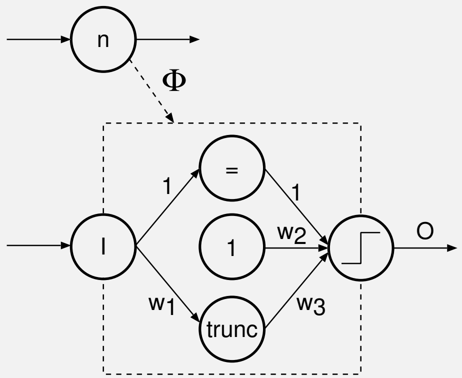
  - Powers



- A filter for atom n checks whether this atom is entailed in I, by computing a modified pref-function

$$\text{pref} : I_{\mathcal{P}} \times \mathbb{N} \rightarrow I_{\mathcal{P}}$$

$$(x, n) \mapsto \frac{\text{trunc}(x \cdot B^n)}{B^n}$$



$$\Phi(w_1, n) = 4^n$$

$$\Phi(w_2, n) = -4^{-n}$$

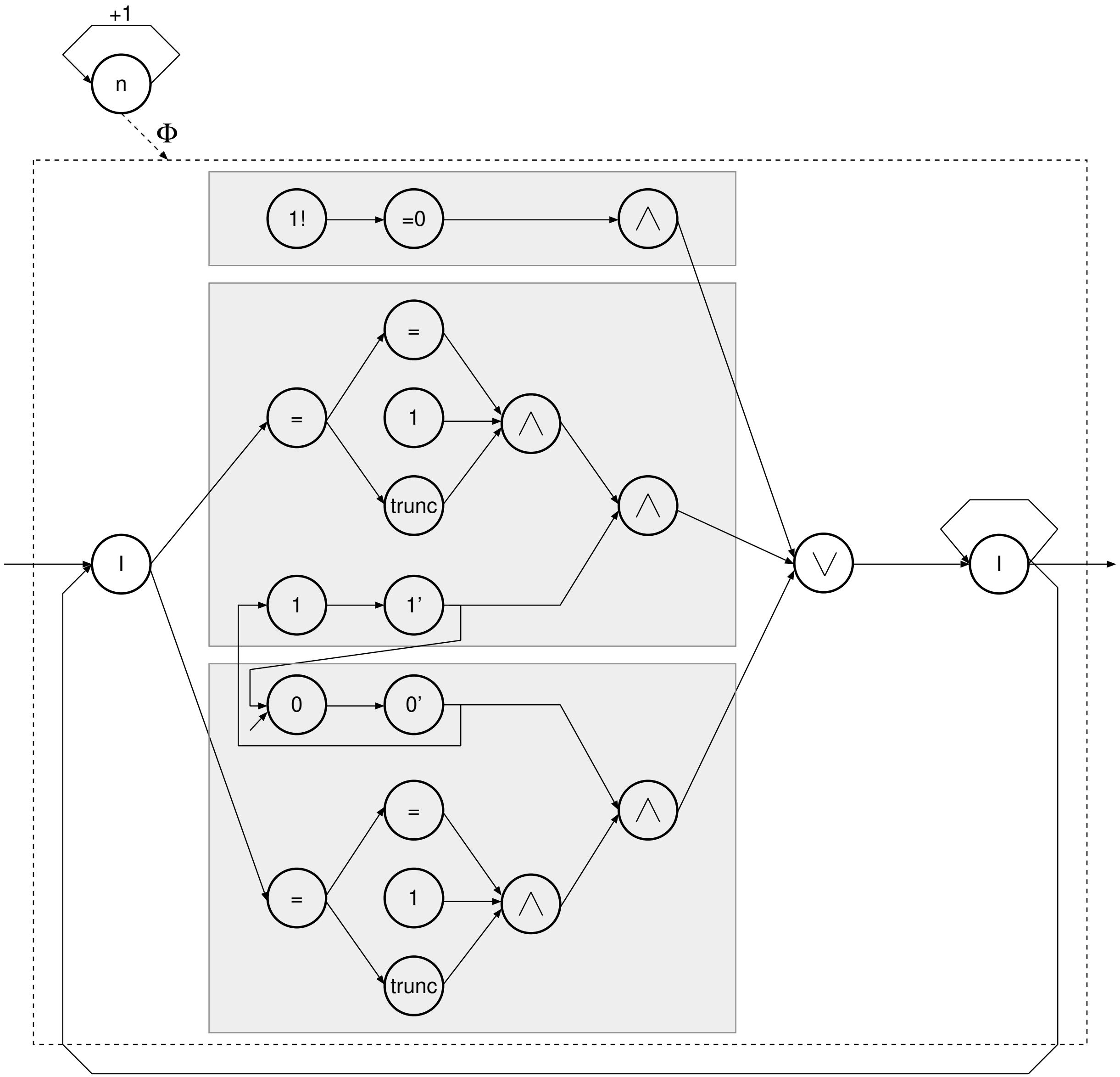
$$\Phi(w_3, n) = 4^{-n}$$

A Worked Example

$even(0).$

$even(s(X)) \leftarrow not\ even(X).$

$odd(s(X)) \leftarrow even(X).$



Conclusion

- We can implement the $T_{\mathcal{P}}$-operator associated to a logic program $\mathcal{P}$ in a neural network

Further Work

- Development of training techniques for fibring neural networks
- Automatic construction of fibring neural networks
- ...

Thank you for your attention