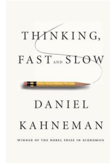
How to make logics neurosymbolic

Luc De Raedt
Giuseppe Marra & Vincent Derkinderen & Sebastijan Dumancic & Robin Manhaeve & Thomas Winters & Angelika Kimmig

DTAI ÖREBRO UNIVERSITET KU LEUVEN LEUVEN.AI INSTITUTE TAILOR WASP EOS THE EXCELLENCE OF SCIENCE

Learning and Reasoning both needed



- System 1 - thinking fast - can do things like 2+2 = ? and recognise objects in image
- System 2 - thinking slow - can reason about solving complex problems - planning a complex task
- alternative terms — data-driven vs knowledge-driven, symbolic vs subsymbolic, solvers and learners, neuro-symbolic…

- **A lot of work on integrating learning and reasoning, neural symbolic computation to integrate logic / symbols reasoning with neural networks**

  see also arguments by Marcus, Darwiche, Levesque, Tenenbaum, Geffner, Bengio, Le Cun, Kautz, …

Real-life problems involve two important aspects.

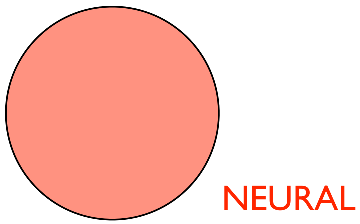
https://www.theorie-blokken.be/nl/gratis-proefexamen

Who can go first ?

A. The red car
B. The blue van
C. The white car

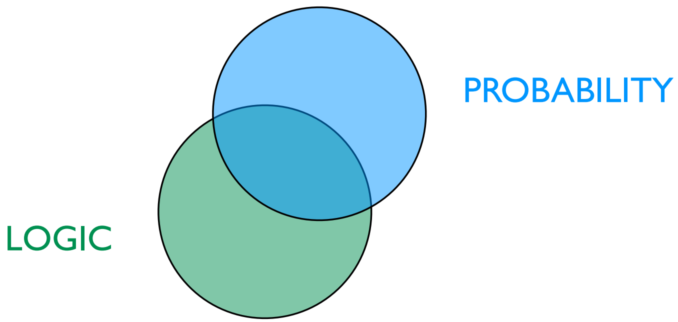
Thinking fast

MAIN PARADIGM in AI
Focus on Learning



Thinking slow = reasoning

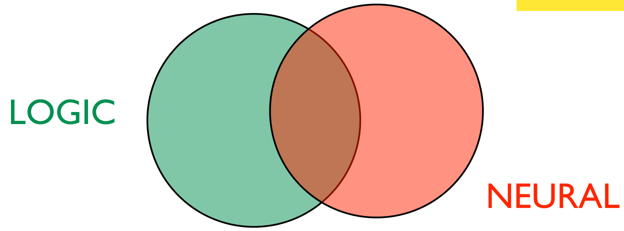
TWO MAIN PARADIGMS in AI



Their integration has been well studied in
Probabilistic (Logic) Programming and Statistical Relational AI (StarAI)

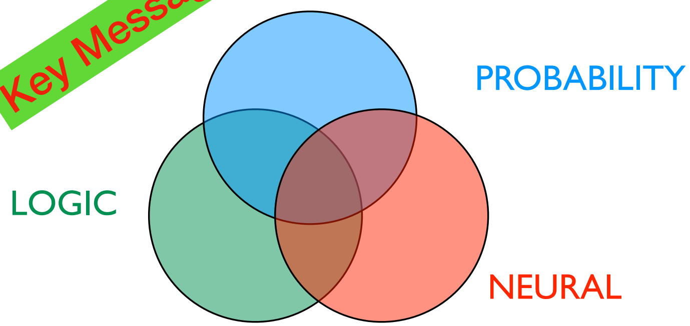
Neurosymbolic = Neuro + Logic

WARNING
TALK MAY NOT COVER ALL of NESY



Neurosymbolic = Neuro + Logic + Probability

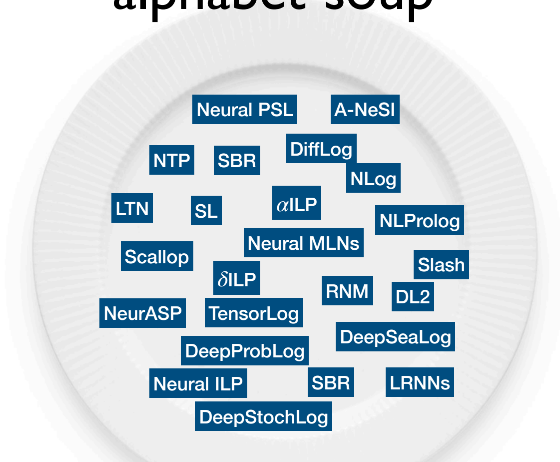
Key Message 1



see Manhaeve et al. NeSy Book

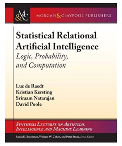
interpret PROBABILITY broadly (including fuzzy)

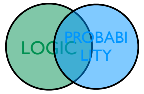
The NeuroSymbolic alphabet-soup

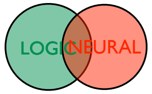


check our survey on arxiv — Marra, Dumancic, Manhaeve & De Raedt, 23

Key Message 2







StarAI and NeSy share similar problems and thus similar solutions apply

See also [De Raedt et al., IJCAI 20; Marra et al, arxiv]

Key Message 3

Provide recipe for

Kautz

Neural : : Symbolic

"an interface layer (<> pipeline) between neural & symbolic components"

Part 1: NeSy AI - a little Survey

Part 2: The Recipe

Part 3: DeepStochLog and DeepProbLog

Part 1: NeSy AI - a little survey

check our survey on arxiv — Marra, Dumancic, Manhaeve & De Raedt, 23

Logic Programs

as in the programming language Prolog

Propositional logic program

burglary.
hears_alarm_mary.

earthquake.
hears_alarm_john.

alarm :– earthquake.

alarm :– burglary.

calls_mary :– alarm, hears_alarm_mary.

calls_john :– alarm, hears_alarm_john.

Two proofs (by refutation)

:- calls_mary.

:- alarm, hears_alarm_mary.

:- earthquake, hears_alarm_mary.

:- burglary, hears_alarm_

:- hears_alarm_mary.

:- hears_alarm_mary.

▯

▯

LOGIC

A proof-theoretic view

Logic as constraints

as in SAT solvers

Propositional logic

IFF / AND

calls(mary) ↔ hears_alarm(mary) ∧ alarm

calls(john) ↔ hears_alarm(john) ∧ alarm

OR

alarm ↔ earthquake ∨ burglary

Model / Possible World

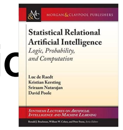
{ burglary,
hears_alarm(john),
alarm,
calls(john)}

the facts that are true
in this model / possible world

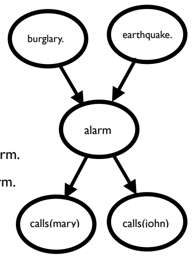
LOGIC

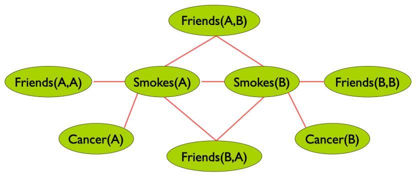
A model-theoretic view

Two types of probabilistic graphical models and StarAI systems



0.1 :: burglary.
0.05 :: earthquake.
alarm :– earthquake.
alarm :– burglary.
0.7::calls(mary) :– alarm.
0.6::calls(john) :– alarm.





$1.5 \quad \forall x\ Smokes(x) \Rightarrow Cancer(x)$

$1.1 \quad \forall x, y\ Friends(x,y) \Rightarrow \big(Smokes(x) \Leftrightarrow Smokes(y)\big)$

Probabilistic Logic Programs
ProbLog

directed
Bayesian Net

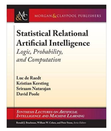
Markov Logic

undirected
Markov Net
model theoretic

key representatives

LOGIC PROBABILITY

Two types of Neural Symbolic Systems



Just like in StarAI

Logic as a kind of ***neural program***

directed StarAI approach and
logic programs

Logic as the ***regularizer***
(reminiscent of Markov Logic Networks)

undirected StarAI approach and
(soft) constraints

Also, many NeSy systems are doing
knowledge based model construction KBMC
where logic is used as a template

Just like in StarAI

LOGIC NEURAL

Logic as a neural program

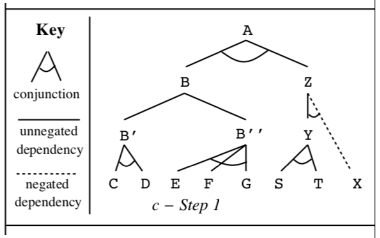
directed StarAI approach and logic programs

- KBANN (Towell and Shavlik AIJ 94)
- Turn a (propositional) Prolog program into a neural network and learn

```
A :- B, Z.
B :- C, D.
B :- E, F, G.
Z :- Y, not X.
Y :- S, T.
```

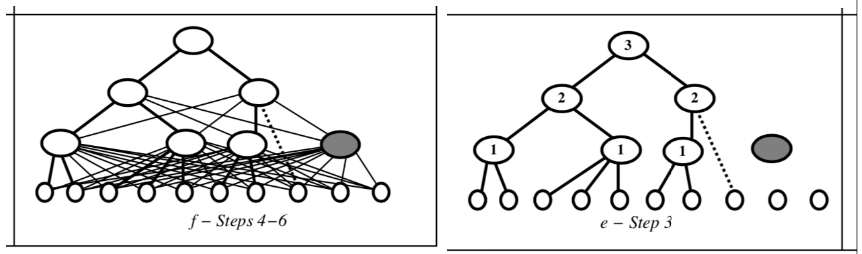
REWRITE

```
A   :- B, Z.
B   :- B'.
B   :- B''.
B'  :- C, D.
B'' :- E, F, G.
Z   :- Y, not X.
Y   :- S, T.
```



Logic as a neural program

directed StarAI approach and logic programs



ADD LINKS — ALSO SPURIOUS ONES

HIDDEN UNIT

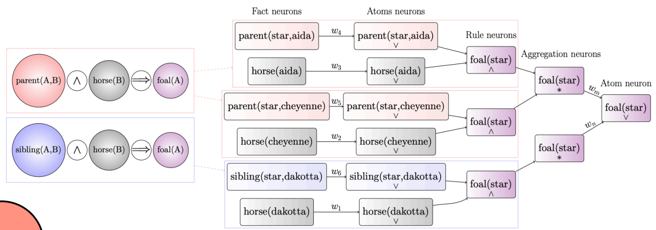
and then learn

(Details of activation & loss functions not mentioned)

Lifted Relational Neural Networks

directed StarAI approach and logic programs

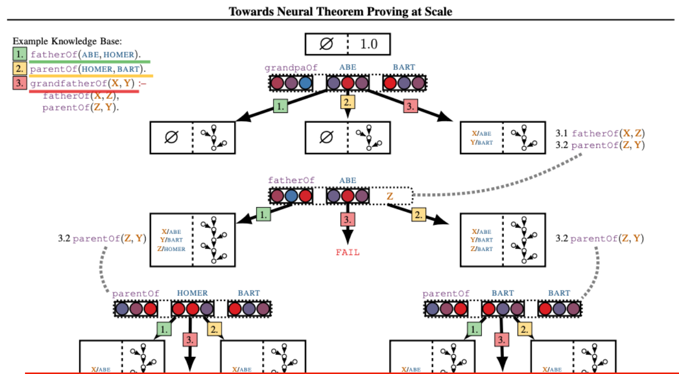
- Directed (fuzzy) NeSy
- similar in spirit to the Bayesian Logic Programs and Probabilistic Relational Models
- Of course, other kind of (fuzzy) operations for AND, OR and Aggregation (cf. later)



[Sourek, Kuzelka, et al JAIR]

directed StarAI approach and logic programs

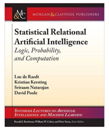
Neural Theorem Prover



the logic is encoded in the network
how to reason logically ?

[Rocktäschel Riedel, NeurIPS 17; Minervini et al.]

Two types of Neural Symbolic Systems



Just like in StarAI

Logic as a kind of ***neural program***

directed StarAI approach and logic programs

Logic as the ***regularizer***
(reminiscent of Markov Logic Networks)

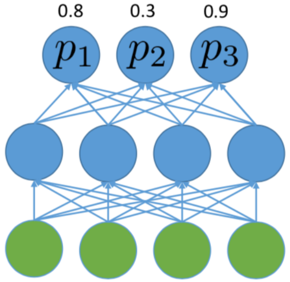
undirected StarAI approach and (soft) constraints

Also, many NeSy systems are doing
knowledge based model construction KBMC
where logic is used as a template

Logic as constraints

undirected StarAI approach and (soft) constraints

multi-class classification



This constraint should be satisfied

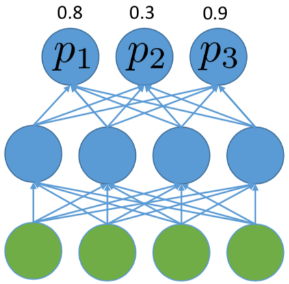
$$(\neg x_1 \wedge \neg x_2 \wedge x_3)\vee$$
$$(\neg x_1 \wedge x_2 \wedge \neg x_3)\vee$$
$$(x_1 \wedge \neg x_2 \wedge \neg x_3)$$

figures and example from Xu et al., ICML 2018

Logic as constraints

undirected StarAI approach and (soft) constraints

multi-class classification



Probability that constraint is satisfied

$$(1-x_1)(1-x_2)x_3+$$
$$(1-x_1)x_2(1-x_3)+$$
$$x_1(1-x_2)(1-x_3)$$

basis for SEMANTIC LOSS

(weighted model counting)

Logic as a regularizer

undirected StarAI approach and (soft) constraints

Semantic Loss:

- Use logic as constraints (very much like "propositional MLNs)
- Semantic loss

$$SLoss(T) \propto -\log \sum_{X \models T} \prod_{x \in X} p_i \prod_{\neg x \in X} (1-p_i)$$

- Used as regulariser

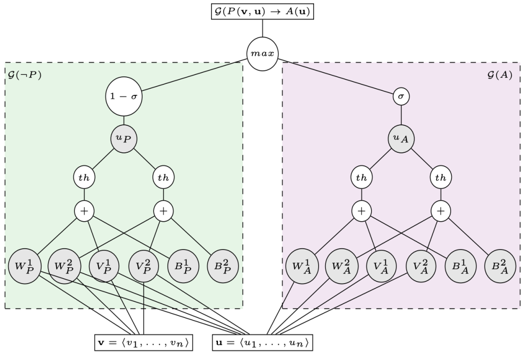
$$Loss = TraditionalLoss + w.SLoss$$

- Use weighted model counting , close to StarAI

Logic Tensor Networks

undirected StarAI approach and (soft) constraints

$P(x,y) \rightarrow A(y)$, with $\mathcal{G}(x) = \mathbf{v}$ and $\mathcal{G}(y) = \mathbf{u}$



LOGIC NEURAL

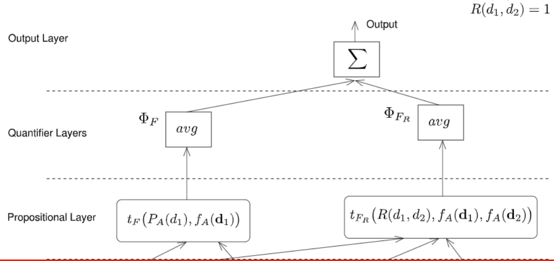
Serafini & Garcez

Semantic Based Regularization

undirected StarAI approach and (soft) constraints

$F := \forall d\ P_A(d) \Rightarrow A(d)$
$F_R := \forall d\ \forall d'\ R(d,d') \Rightarrow ((A(d) \wedge A(d')) \vee (\neg A(d) \wedge \neg A(d')))$
$C = \{d_1, d_2\}$

Evidence Predicate Groundings
$P_A(d_1) = 1$
$R(d_1, d_2) = 1$

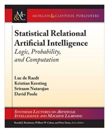


the logic is encoded in the network
how to reason logically ?

LOGIC NEURAL

Diligenti et al. AIJ

Two types of Neural Symbolic Systems



Just like in StarAI

Logic as a kind of ***neural program***

directed StarAI approach and logic programs

Logic as the ***regularizer*** ***(reminiscent of Markov Logic Networks)***

undirected StarAI approach and (soft) constraints

Consequence :
the logic is encoded in the network
the ability to logically reason is lost
logic is not a special case

LOGIC NEURAL

Part 2: The Recipe

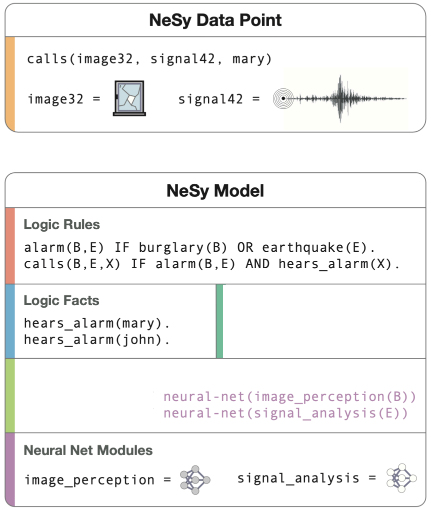
A recipe for NeSy

STEP 1

1. Take your favorite symbolic (logic / rule based) representation

(applied on DeepProbLog)

layout Pieter Robberechts



A recipe for NeSy

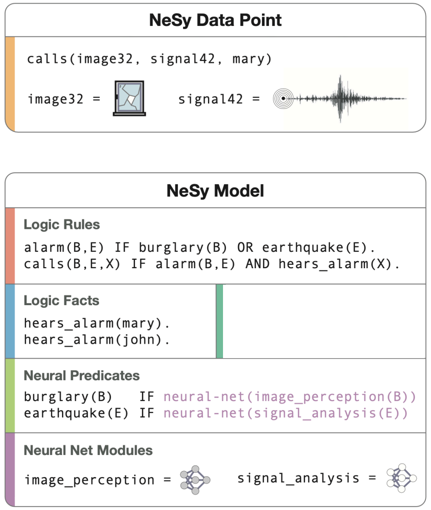
STEP 1

1. Take your favorite symbolic (logic / rule based) representation

2. Interpret neural networks as neural predicates

(applied on DeepProbLog)

layout Pieter Robberechts



A recipe for NeSy

STEP 1

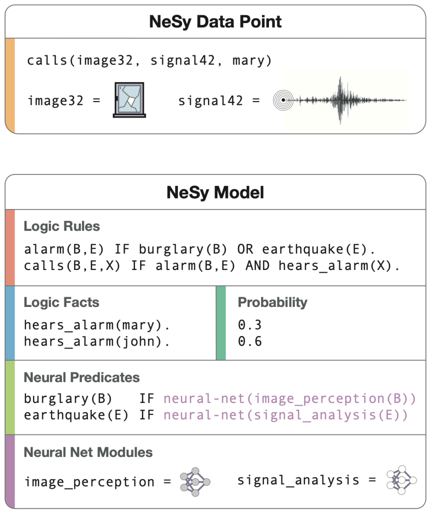
1. Take your favorite symbolic (logic / rule based) representation

2. Interpret neural networks as neural predicates

3. Turn the 0/1 or True/False into Probabilistic or Fuzzy Interpretation

(applied on DeepProbLog)

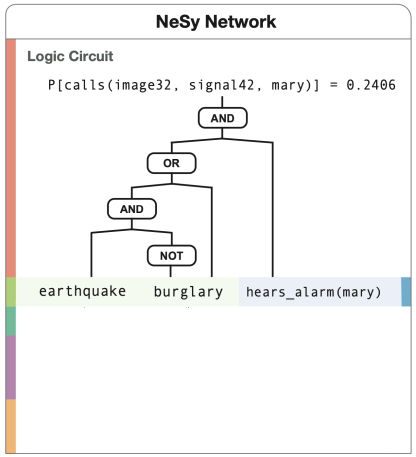
layout Pieter Robberechts



A recipe for NeSy

STEP 2

4. Construct logical proof / explanation for example



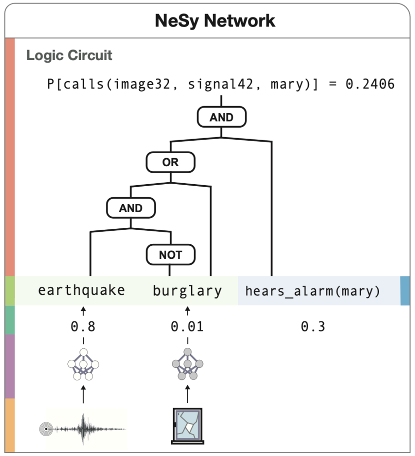
layout Pieter Robberechts

A recipe for NeSy

STEP 2

4. Construct logical proof / explanation for example

5. Add the neural networks to the corresponding predicates (*reparametrise*)



layout Pieter Robberechts

A recipe for NeSy

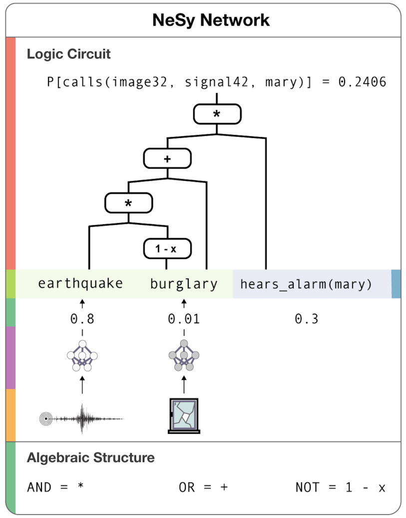
STEP 3

4. Construct logical proof / explanation for example

5. Add the neural networks to the corresponding predicates (*reparametrise*)

6. Replace OR and AND by $\oplus$ and $\otimes$

7. Differentiate



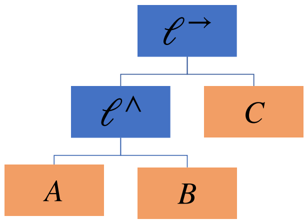
layout Pieter Robberechts

A recipe for NeSy

Where do the numbers come from ?

From logic formulae to circuits

$\ell((A \wedge B) \rightarrow C)$ $\quad\quad$ $\ell(Q)$



The query Q determines the structure

A recipe for NeSy

Where do the numbers come from ?

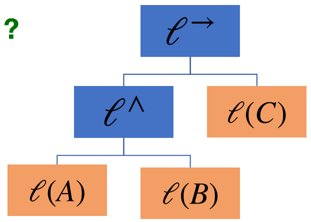
From logic formulae to circuits

$\ell_F((A \wedge B) \rightarrow C)$ $\quad\quad$ $\ell(Q)$

What is the algebraic structure ? = Parametric circuit

What operators ?

What labeling functions ?



The query Q determines the structure (potentially after knowledge compilation)

A recipe for NeSy

Where do the numbers come from ?

Boolean

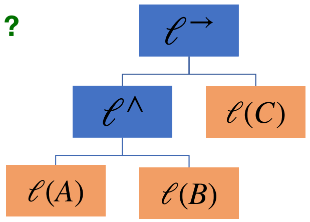
$\ell_F((A \wedge B) \rightarrow C)$

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

p	q	p→q
T	T	T
T	F	F
F	T	T
F	F	T

What operators ?

What labeling functions ?

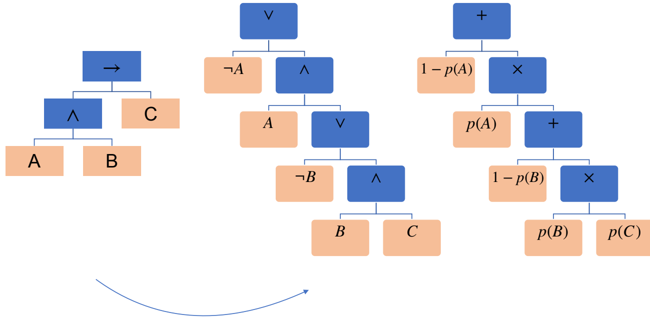


The query Q determines the structure (potentially after knowledge compilation)

A recipe for NeSy

Where do the numbers come from ?

Probability



Knowledge Compilation **(computationally expensive)**

Probabilistic structure is explicit in compiled formula.

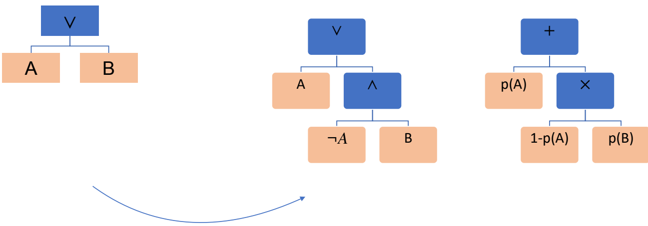
A recipe for NeSy

Where do the numbers come from ?

Probability

Why Compile

P(A ∨ B) = P(A) + P(B) - P(A ∧ B)



Knowledge Compilation **(computationally expensive)**

Probabilistic structure is explicit in compiled formula.

A recipe for NeSy

Where do the numbers come from ?

Fuzzy

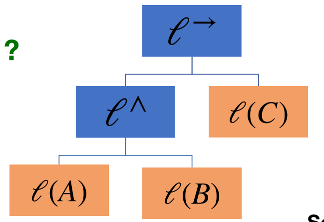
- t-norm extends conjunction to [0,1] interval
- Three fundamental t-norms:
  - Lukasiewicz t-norm: $t_L(x,y) = \max(0, x+y-1)$
  - Goedel t-norm: $t_G(x,y) = \min(x,y)$
  - Product t-norm: $t_P(x,y) = x \cdot y$

Other operators derived from the t-norm

	Product	Łukasiewicz	Gödel
$x \wedge y$	$x \cdot y$	$\max(0, x+y-1)$	$\min(x,y)$
$x \vee y$	$x+y-x \cdot y$	$\min(1, x+y)$	$\max(x,y)$
$\neg x$	$1-x$	$1-x$	$1-x$
$x \Rightarrow y \ (x > y)$	y/x	$\min(1, 1-x+y)$	y

What operators ?

What labeling functions ?



continuous and differentiable

but a measure of **vagueness** not of uncertainty

Many problems
See [Van Krieken et al AIJ]

Logic as soft constraints
Markov Logic

Propositional logic			Model / Possible World
10:	calls(mary) <- hears_alarm(mary) ∧ alarm		{burglary,
20:	calls(john) <- hears_alarm(john) ∧ alarm		hears_alarm(john),
			alarm,
30:	alarm <- earthquake v burglary		calls(john)}

probability of world ~ e^10 x e^20 x e^30

using weighted model counting (WMC)

weights/probabilities are on the formulae (soft constraints)

the higher the weight , the harder or more logical the constraint

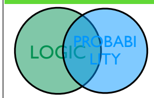
Probability

$w(f1) = e\hat{}10$ $\quad$ $w(\text{not } f1) = e\hat{}0 = 1$

$w(f2) = e\hat{}20$ $\quad$ $w(\text{not } f2) = e\hat{}0 = 1$

$w(f3) = e\hat{}30$ $\quad$ $w(\text{not } f3) = e\hat{}0 = 1$

(need to normalise to get probability distribution)



erc

Logic as soft constraints

Probabilistic Soft Logic [Bach & Getoor]

Propositional logic		Model / Possible World
10 :	calls(mary) <- hears_alarm(mary) ∧ alarm	{0.7 burglary,
		0.8 hears_alarm(john),
20 :	calls(john) <- hears_alarm(john) ∧ alarm	0.5 alarm,
30 :	alarm <- earthquake v burglary	0.3 calls(john)}

atoms are no longer true or false in worlds but true or false to a certain degree

logic : a constraint is satisfied (1) or not (0) by a world

fuzzy logic : the distance to satisfaction

the higher the distance, the less likely the world

calls(john) <- hears_alarm(john) ∧ alarm

≥ 0.5 $\quad$ 0.7 $\quad$ 0.8

$$A \wedge B = min(1, 1.5 - 1) = 0.5$$

Rule evaluates to $min(1, 1 - 0.5 + 0.3) = 0.8$ when calls(john) =0.3

Lukasiewicz T-norm

For 0 and 1 we get boolean logic

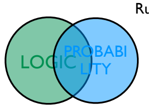
$A \vee B = min(1, A + B)$

$A \wedge B = min(1, A + B - 1)$

$A \leftarrow B = min(1, 1 + A - B)$ **(residuum)**

evaluates to 1 when rule is satisfied

when $B \leq A$



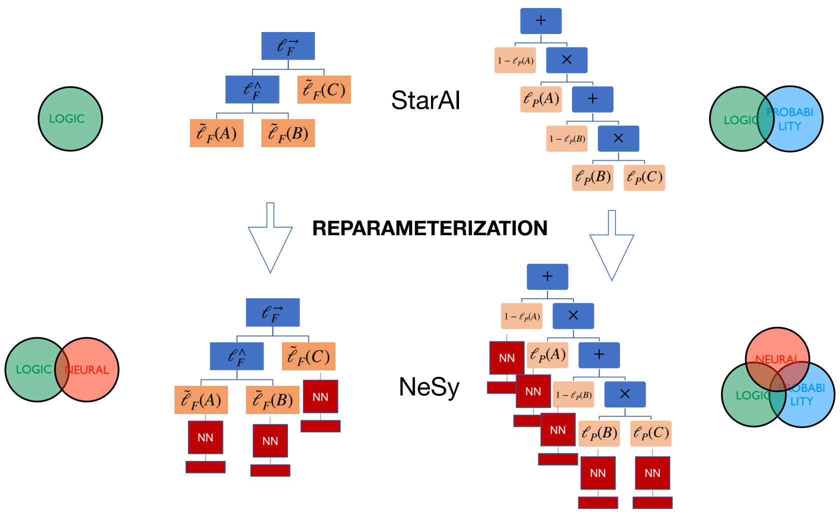
Fuzzy

w= e^ [—20 x (1-0.8)]

erc

See Van Krieken et al AIJ 22

From StarAI to NeSy



Part 3: DeepStochLog and DeepProbLog

Two types of probabilistic models / programs

- Based on a *random graph model*
  - Bayesian Nets and ProbLog **-> DeepProbLog [AIJ 21]**
- Based on a *random walk model*
  - Probabilistic grammars and Stochastic Logic Programs [Muggleton] **-> DeepStochLog [AAAI 22]**

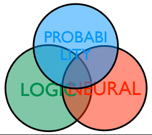
Our method/recipe:
Take an existing probabilistic logic and inject neural predicates that act ako interface

DeepProbLog

DeepProbLog = Probability + Logic + Neural Network

DeepProbLog = ProbLog + Neural Network

Related work in NeSy	**DeepProbLog**
Logic is made less expressive	Full expressivity is retained
Logic is pushed into the neural network	Maintain both logic and neural networ
Fuzzy logic	Probabilistic logic programming
Language semantics unclear	Clear semantics



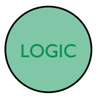
Holds also for DeepStochLog

DeepStochLog = SLPs + Neural Network

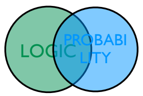
PART 3 A

From Prolog to ProbLog

FROM



TO



Logic Programs

as in the programming language Prolog

Propositional logic program

burglary.
hears_alarm(mary).

earthquake.
hears_alarm(john).

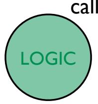
alarm :– earthquake.
alarm :– burglary.

calls(mary) :– alarm, hears_alarm(mary).
calls(john) :– alarm, hears_alarm(john).

Two proofs (by refutation)

:- calls(mary).
:- alarm, hears_alarm(mary).
:- earthquake, hears_alarm(mary).
:- burglary, hears_alarm(
:- hears_alarm(mary).
:- hears_alarm(mary)
[]
[]

A proof-theoretic view



Probabilistic Logic Programs

as in the probabilistic programming language ProbLog

Propositional logic program

0.1 :: burglary.
0.3 ::hears_alarm(mary).

Probabilistic facts

0.05 ::earthquake.
0.6 ::hears_alarm(john).

alarm :– earthquake.
alarm :– burglary.

calls(mary) :– alarm, hears_alarm(mary).
calls(john) :– alarm, hears_alarm(john).

Key Idea (Sato & Poole)
the distribution semantics:

unify the basic concepts in logic and probability:

random variable ~ propositional variable

an interface between logic and probability

LOGIC PROBABILITY

Probabilistic Logic Programs

as in the probabilistic programming language ProbLog

Propositional logic program

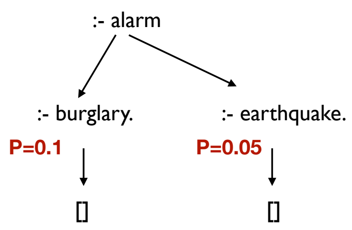
0.1 :: burglary.
0.3 ::hears_alarm(mary).

0.05 ::earthquake.
0.6 ::hears_alarm(john).

alarm :– earthquake.
alarm :– burglary.

calls(mary) :– alarm, hears_alarm(mary).
calls(john) :– alarm, hears_alarm(john).

Two proofs (by refutation)



Probability of one proof : $\prod_{f:fact \in Proof} P_f$

LOGIC PROBABILITY

Probabilistic Logic Programs

as in the probabilistic programming language ProbLog

Propositional logic program

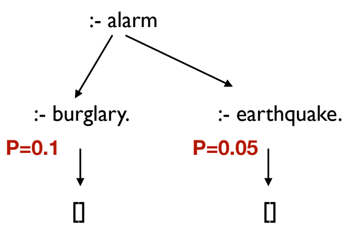
0.1 :: burglary.
0.3 ::hears_alarm(mary).

0.05 ::earthquake.
0.6 ::hears_alarm(john).

alarm :– earthquake.
alarm :– burglary.

calls(mary) :– alarm, hears_alarm(mary).
calls(john) :– alarm, hears_alarm(john).

Disjoint sum problem



Probability of one proof : $\prod_{f:fact \in Proof} P_f$

P(alarm) = P(burg OR earth)
= P(burg) + P(earth) - P(burg AND earth)
=/= P(burg) + P(earth)

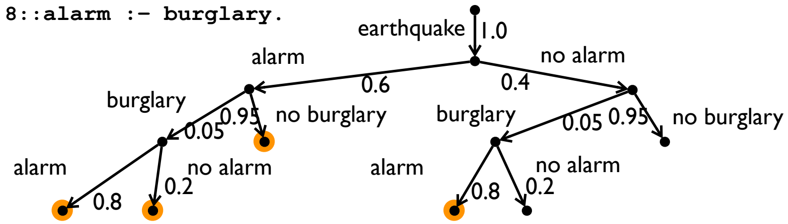
LOGIC PROBABILITY

Probabilistic Logic Program Semantics

```
earthquake.
0.05::burglary.

0.6::alarm :- earthquake.
0.8::alarm :- burglary.
```

[Vennekens et al, ICLP 04]

probabilistic causal laws



P(alarm)=0.6×0.05×0.8+0.6×0.05×0.2+0.6×0.95+0.4×0.05×0.8

Probabilistic Logic Program Semantics

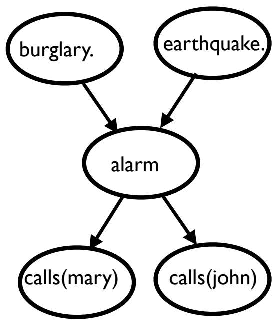
Propositional logic program

0.1 :: burglary.
0.05 :: earthquake.

alarm :– earthquake.
alarm :– burglary.

0.7::calls(mary) :– alarm.
0.6::calls(john) :– alarm.

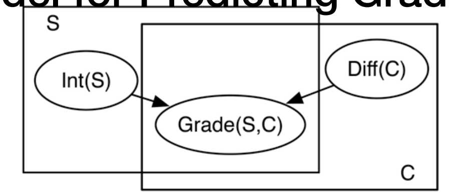
Bayesian Network



Bayesian net encoded as Probabilistic Logic Program
PLPs correspond to directed graphical models

ProbLog has both (directed) probabilistic graphic models, the programming language Prolog (and probabilistic databases) as special case

Flexible and Compact Relational Model for Predicting Grades



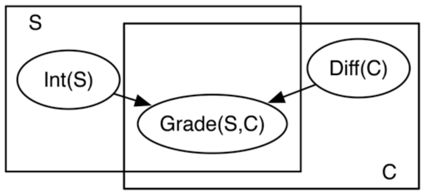
"Program" Abstraction:

- S, C logical variable representing students, courses
- the set of individuals of a type is called a population
- Int(S), Grade(S, C), D(C) are parametrized random variables

Grounding:

- for every student s, there is a random variable Int(s)
- for every course c, there is a random variable Di(c)
- for every s, c pair there is a random variable Grade(s,c)
- all instances share the same structure and parameters

ProbLog by example: Grading



Shows relational structure

grounded model: replace variables by constants

Works for any number of students / classes (for 1000 students and 100 classes, you get 101100 random variables); still only few parameters

With SRL / PP

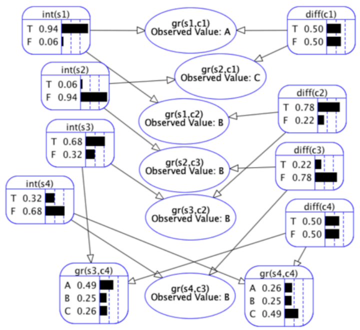
build and learn compact models,

from one set of individuals - > other sets;

reason also about exchangeability,

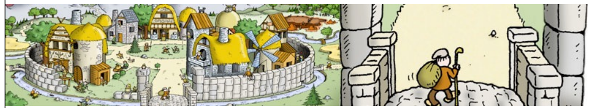
build even more complex models,

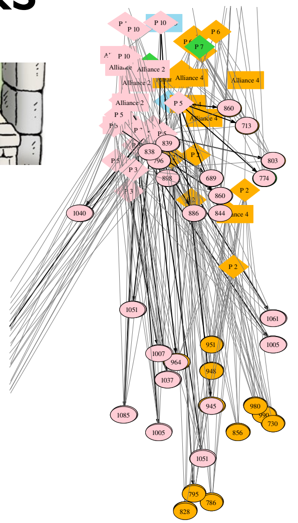
incorporate background knowledge



ProbLog applications

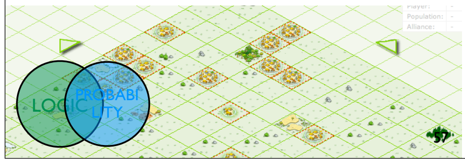
Dynamic networks





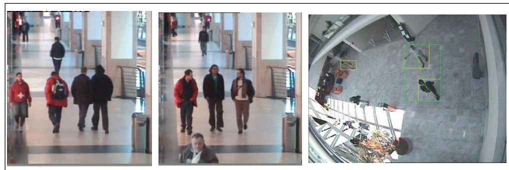
Travian: A massively multiplayer real-time strategy game

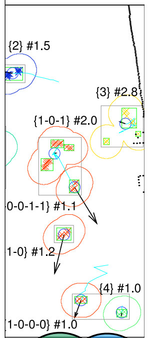
Can we build a model of this world ? Can we use it for playing better ?



[Thon et al, MLJ 11]

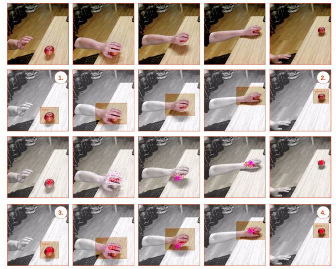
Activity analysis and tracking





- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?

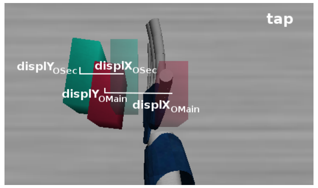
[Skarlatidis et al, TPLP 14; Nitti et al, IROS 13, ICRA 14, MLJ 16]



[Persson et al, IEEE Trans on Cogn. & Dev. Sys. 19; IJCAI 20]

Learning relational affordances

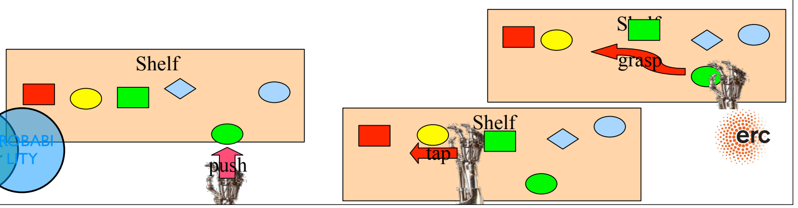
But focus on learning probabilistic aspects, neural networks still pipelined



similar to probabilistic Strips (with continuous distributions)

Learning relational affordances between two objects (learnt by experience)

Moldovan et al. ICRA 12, 13, 14; Auton. Robots 18



Biology

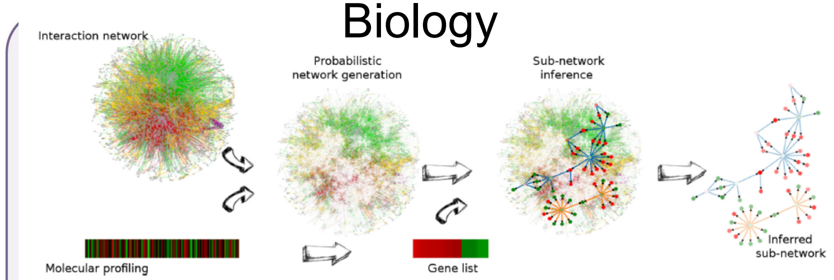


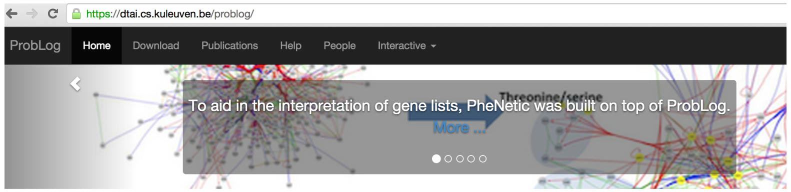
Figure 1. Overview of PheNetic, a web service for network-based interpretation of 'omics' data. The web service uses as input a genome wide interaction network for the organism of interest, a user generated molecular profiling data set and a gene list derived from these data. Interaction networks for a wide variety of organisms are readily available from the web server. Using the uploaded user-generated molecular data the interaction network is converted into a probabilistic network: edges receive a probability proportional to the levels measured for the terminal nodes in the molecular profiling data set. This probabilistic interaction network is used to infer the sub-network that best links the genes from the gene list. The inferred sub-network provides a trade-off between linking as many genes as possible from the gene list and selecting the least number of edges.

- Causes: Mutations
  - All related to similar phenotype
- Effects: Differentially expressed genes
- 27 000 cause effect pairs

- Interaction network:
  - 3063 nodes
    - Genes
    - Proteins
  - 16794 edges
    - Molecular interactions
    - Uncertain

- Goal: connect causes to effects through common subnetwork
  - = Find mechanism
- Techniques:
  - DTProbLog
  - Approximate inference

[De Maeyer et al., Molecular Biosystems 13, NAR 15] [Gross et al. Communications Biology, 19]



Introduction.

Probabilistic logic programs are logic programs in which some of the facts are annotated with probabilities.

ProbLog is a tool that allows you to intuitively build programs that do not only encode **complex interactions** between a large sets of **heterogenous components** bu
uncertainties that are present in real-life situations.

The engine tackles several tasks such as computing the marginals given evidence and learning from (partial) interpretations. ProbLog is a suite of efficient algorithms
tasks. It is based on a conversion of the program and the queries and evidence to a weighted Boolean formula. This allows us to reduce the inference tasks to well-s
weighted model counting, which can be solved using state-of-the-art methods known from the graphical model and knowledge compilation literature.

The Language. Probabilistic Logic Programming.

ProbLog makes it easy to express complex, probabilistic models.

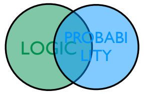
```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :- friend(X,Y), influences(Y,X), smokes(Y).
```

LOGIC PROBABILITY

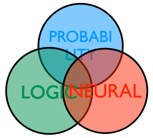
PART 3 B

From ProbLog to DeepProbLog

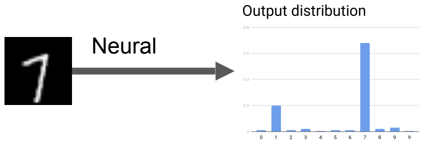
FROM



TO



Neural predicate



- Neural networks have uncertainty in their predictions
- A normalized output can be interpreted as a probability distribution
- Neural predicate models the output as probabilistic facts
- No changes needed in the probabilistic host language

Key Idea DeepProbLog

unify the basic concepts in logic and neural networks:

neural predicate ~ neural net

an interface between logic and neural nets

PROBABILITY LOGIC NEURAL

The neural predicate

The output of the neural network is probabilistic facts in DeepProbLog

Example:

```
nn(mnist_net, [X], Y, [0 ... 9] ) :: digit(X,Y).
```

Instantiated into a (neural) Annotated Disjunction:

```
0.04::digit( ,0) ; 0.35::digit( ,1) ; ... ;
0.53::digit( ,7) ; ... ; 0.014::digit( ,9).
```

PROBABILITY LOGIC NEURAL

DeepProbLog exemplified: MNIST addition

Task: Classify pairs of MNIST digits with their sum

Benefit of DeepProbLog:

- Encode addition in logic
- Separate addition from digit classification

```
nn(mnist_net, [X], Y, [0 ... 9] ) :: digit(X,Y).

addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

Examples:

```
addition( , ,8), addition( , ,4), addition( , ,11), …
```

DeepProbLog exemplified: MNIST addition

Task: Classify pairs of MNIST digits with their sum

Benefit of DeepProbLog:

- Encode addition in logic
- Separate addition from digit classification

```
nn(mnist_net, [X], Y, [0 ... 9] ) :: digit(X,Y).

addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

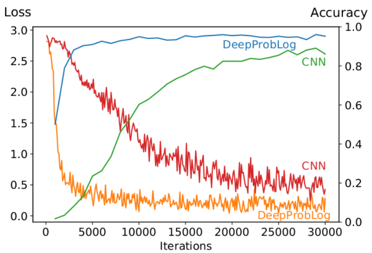
```
addition( , ,8) :- digit( ,N1), digit( ,N2), 8 is N1 + N2.
```

Examples:

```
addition( , ,8), addition( , ,4), addition( , ,11), …
```

MNIST Addition

- Pairs of MNIST images, labeled with sum
- Baseline: CNN
  - Classifies concatenation of both images into classes 0 ...18
- DeepProbLog:
  - CNN that classifies images into 0 … 9
  - Two lines of DeepProblog code
- Result:
- Fewer iterations necessary
- Higher accuracy achieved



Example

Learn to classify the sum of pairs of MNIST digits

Individual digits are not labeled!

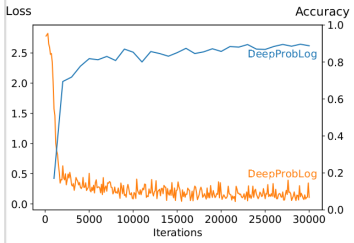
E.g. (, , 8)

Could be done by a CNN: classify the concatenation of both images into 19 classes

However:

Multi-digit MNIST addition with MNIST

```
number ( [ ] , Result , Result ) .
number ( [H | T ] , Acc , Result) :–
  digit(H, Nr ), Acc2 is Nr +10*Acc ,
  number ( T , Acc2 , Result ) .
number (X,Y) :– number (X, 0 ,Y ) .

multiaddition(X, Y, Z ) :–
    number (X, X2 ) ,
    number (Y, Y2 ) ,
    Z is X2+Y2 .
```



(b) Multi-digit (**T2**)

Noisy Addition

```
nn(classifier, [X], Y, [0 .. 9]) :: digit(X,Y).
t(0.2) :: noisy.

1/19 :: uniform(X,Y,0) ; ... ; 1/19 :: uniform(X,Y,18).

addition(X,Y,Z) :- noisy, uniform(X,Y,Z).
addition(X,Y,Z) :- \+noisy, digit(X,N1), digit(Y,N2), Z is N1+N2.
```

(a) The DeepProbLog program.

	Fraction of noise					
	0.0	0.2	0.4	0.6	0.8	1.0
Baseline	93.46	87.85	82.49	52.67	8.79	5.87
DeepProbLog	97.20	95.78	94.50	92.90	46.42	0.88
DeepProbLog w/ explicit noise	96.64	95.96	95.58	94.12	73.22	2.92
Learned fraction of noise	0.000	0.212	0.415	0.618	0.803	0.985

Table 3: The accuracy on the test set for **T4**.

Inference & Learning

ProbLog Inference

Answering a query in a ProbLog program happens in four steps

1. Grounding the program w.r.t. the query
2. Rewrite the ground logic program into a propositional logic formula
3. Compile the formula into an arithmetic circuit
4. Evaluate the arithmetic circuit

```
0.1 :: burglary.
0.5 :: hears_alarm(mary).
0.2 :: earthquake.
0.4 :: hears_alarm(john).
alarm :– earthquake.
alarm :– burglary.
calls(X) :– alarm, hears_alarm(X).
```

Query

P(calls(mary))

ProbLog Inference

Answering a query in a ProbLog program happens in four steps

1. **Grounding the program w.r.t. the query (only relevant part !)**
2. Rewrite the ground logic program into a propositional logic formula
3. Compile the formula into an arithmetic circuit
4. Evaluate the arithmetic circuit

0.1 :: burglary.
0.5 :: hears_alarm(mary).
0.2 :: earthquake.
0.4 :: hears_alarm(john).
alarm :– earthquake.
alarm :– burglary.
calls(mary) :– alarm, hears_alarm(mary).
calls(john) :– alarm, hears_alarm(john).

Query

P(calls(mary))

ProbLog Inference

Answering a query in a ProbLog program happens in four steps

1. Grounding the program w.r.t. the query
2. **Rewrite the ground logic program into a propositional logic formula**
3. Compile the formula into an arithmetic circuit
4. Evaluate the arithmetic circuit

0.1 :: burglary.
0.5 :: hears_alarm(mary).
0.2 :: earthquake.
0.4 :: hears_alarm(john).
alarm :– earthquake.
alarm :– burglary.
calls(mary) :– alarm, hears_alarm(mary).
calls(john) :– alarm, hears_alarm(john).

calls(mary)

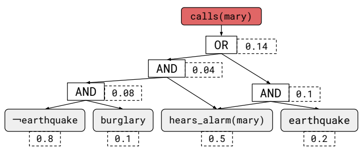
↔

hears_alarm(mary) ∧ (burglary ∨ earthquake)

ProbLog Inference

Answering a query in a ProbLog program happens in four steps

1. Grounding the program w.r.t. the query
2. Rewrite the ground logic program into a propositional logic formula
3. **Compile the formula into an arithmetic circuit (knowledge compilation)**
4. Evaluate the arithmetic circuit



calls(mary)

↔

hears_alarm(mary) ∧ (burglary ∨ earthquake)

Useful Semirings

task	$\mathcal{A}$	$e^{\oplus}$	$e^{\otimes}$	$\oplus$	$\otimes$	$\alpha(v)$	$\alpha(\neg v)$	ref
SAT	$\{true, false\}$	$false$	$true$	$\vee$	$\wedge$	$true$	$true$	B, BT, G, GK, K, L, M
#SAT	$\mathbb{N}$	0	1	$+$	$\cdot$	1	1	B, G, GK, K, L
WMC	$\mathbb{R}_{\geq 0}$	0	1	$+$	$\cdot$	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$	
PROB	$\mathbb{R}_{\geq 0}$	0	1	$+$	$\cdot$	$\in [0,1]$	$1 - \alpha(v)$	B, BT, E, G, K
SENS	$\mathbb{R}[\mathcal{V}]$	0	1	$+$	$\cdot$	v or $\in [0,1]$	$1 - \alpha(v)$	K
GRAD	$\mathbb{R}_{\geq 0} \times \mathbb{R}$	$(0,0)$	$(1,0)$	Eq. (4)	Eq. (5)	Eq. (2)	Eq. (3)	E, K
MPE	$\mathbb{R}_{\geq 0}$	0	1	max	$\cdot$	$\in [0,1]$	$1 - \alpha(v)$	B, BT, G, K, L, M
S-PATH	$\mathbb{N}^{\infty}$	∞	0	min	$+$	$\in \mathbb{N}$	0	BT, GK, K
W-PATH	$\mathbb{N}^{\infty}$	0	∞	max	min	$\in \mathbb{N}$	∞	BT
FUZZY	$[0,1]$	0	1	max	min	$\in [0,1]$	1	GK, M
kWEIGHT	$\{0,\ldots,k\}$	k	0	min	$+^{k}$	$\in \{0,\ldots,k\}$	$\in \{0,\ldots,k\}$	M
$\mathtt{OBDD}_{<}$	$\mathtt{OBDD}_{<}(\mathcal{V})$	$\mathtt{OBDD}_{<}(0)$	$\mathtt{OBDD}_{<}(1)$	$\vee$	$\wedge$	$\mathtt{OBDD}_{<}(v)$	$\neg\mathtt{OBDD}_{<}(v)$	K
WHY	$\mathcal{P}(\mathcal{V})$	$\emptyset$	$\emptyset$	$\cup$	$\cup$	$\{v\}$	n/a	GK
$\mathcal{RA}^{+}$	$\mathbb{N}[\mathcal{V}]$	0	1	$+$	$\cdot$	v	n/a	GK

Table 1: Examples of commutative semirings and labeling functions. The **WHY** and $\mathcal{RA}^{+}$ provenance semirings apply to positive literals only. Reference key: B (Bacchus et al., 2009), BT (Baras and Theodorakopoulos, 2010), E (Eisner, 2002), G (Goodman, 1999), GK (Green et al., 2007), K (Kimmig et al., 2011), L (Larrosa et al., 2010), M (Meseguer et al., 2006); more examples can be found in these references.

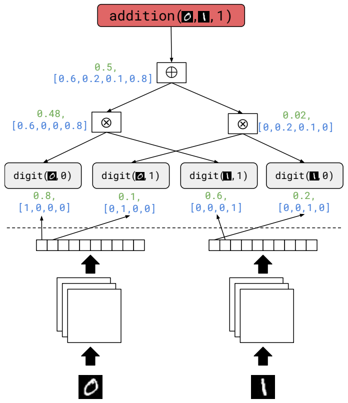
From Kimmig, Vanden Broeck and De Raedt, 2016

Gradient Semiring

```
nn(mnist_net, [X], Y, [0 ... 9] ) ::
    digit(X,Y).

addition(X,Y,Z) :-
   digit(X,N1),
   digit(Y,N2),
   Z is N1+N2.
```

The ACs are differentiable
and there is an interface
with the neural nets

(Pretty elegant in ProbLog
we use the “gradient” semi-ring)



Program Induction/Sketching

In Neural Symbolic methods

- Rule Induction — work with templates

  P(X) :- R(X,Y), Q(Y)

- and have the “predicate” variables / slots P,Q, R determined by the NN
- Simpler form, fill just a few slots / holes

Approach similar to ‘*Programming with a Differentiable Forth Interpreter*’ [1] ∂4

- Partially defined Forth program with slots / holes
- Slots are filled by neural network (encoder / decoder)
- Fully differentiable interpreter: NNs are trained with input / output examples

[1]: Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, Sebastian Riedel: Programming with a Differentiable Forth Interpreter.

Example DeepProbLog

neural predicate

```
hole(X,Y,X,Y):-
    swap(X,Y,0).

hole(X,Y,Y,X):-
    swap(X,Y,1).
```

bubble sort

```
bubble([X],[],X).
bubble([H1,H2|T],[X1|T1],X):-
    hole(H1,H2,X1,X2),
    bubble([X2|T],T1,X).

bubblesort([],L,L).

bubblesort(L,L3,Sorted) :-
    bubble(L,L2,X),
    bubblesort(L2,[X|L3],Sorted).

sort(L,L2) :- bubblesort(L,[],L2).
```

	Test Length	Sorting: Training length 2	3	4	5	6	Addition: training length 2	4	8
∂4 [Bošnjak et al., 2017]	8	100.0	100.0	49.22	–	–	100.0	100.0	100.0
	64	100.0	100.0	20.65	–	–	100.0	100.0	100.0
DeepProbLog	8	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	64	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

(a) Accuracy on the sorting and addition problems (results for ∂4 reported by Bošnjak et al. [2017]).

Training length ⟶	2	3	4	5	6
∂4 on GPU	42 s	160 s	–	–	–
∂4 on CPU	61 s	390 s	–	–	–
DeepProbLog on CPU	11 s	14 s	32 s	114 s	245 s

(b) Time until 100% accurate on test length 8 for the sorting problem.

Table 1: Results on the Differentiable Forth experiments

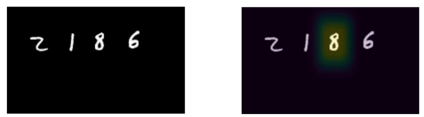
DeepSeaProbLog

discrete and continuous distributions [De Smet UAI 23]

useful for robotics and perception

dim is neural net returning parameters of normal distribution.

```
length(Obj) ~ normal(dim(Obj,Image)).

large(Obj) :- length(Obj) > 100.
```



determining order digits to determine year

DeepSeaProbLog

discrete and continuous distributions [De Smet UAI 23]

generative model with variational autoencoders (see also [Misoni et al NeurIPS 22])

So far from input  to output 11 so that **SUM(**  **,11)** holds

In DeepSeaProblog, you can query **SUM(**  **, X, 5)**

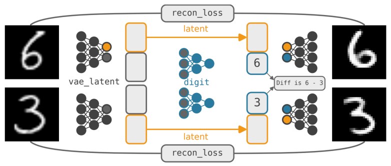
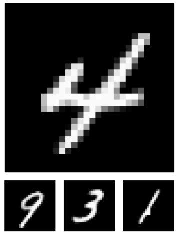


Figure 4: Given example pairs of images and the value of their subtraction, e.g., (6, 3) and 3, the CVAE encoder vae_latent first encodes each image into a multivariate normal NDF (latent) and a latent vector. The latter is the input of a categorical NDF digit, completing the CVAE latent space. Supervision is dual; generated images are compared to the original ones in a probabilistic reconstruction loss, while both digits need to subtract to the given value.



Neural Theorem Prover

Towards Neural Theorem Proving at Scale

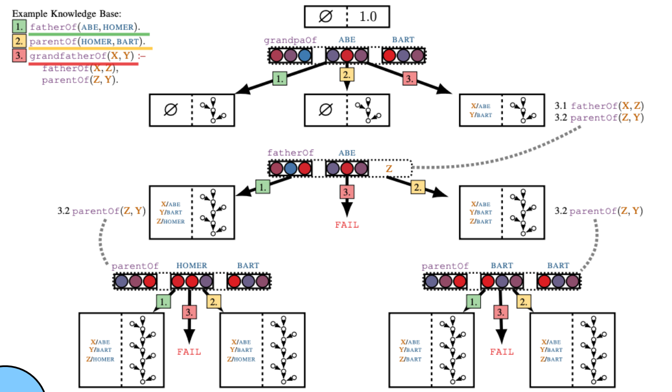


Figure 1. A visual depiction of the NTP' recursive computation graph construction, applied to a toy KB (top left). Dash-separated rectangles denote proof states (left: substitutions, right: proof score -generating neural network). All the non-FAIL proof states are aggregated to obtain the final proof success (depicted in Figure 2). Colours and indices on arrows correspond to the respective KB rule application.

Minervini Bosnjak Rocktäschel Riedel

Soft Unification

- NTP : "grandpa" **softly unifies** with "grandfather", as embeddings are close
- DeepProblog : define

  softunification(X,Y) :- embed(X,EX), embed(Y,EY), rbf(EX,EY).

  softunification(X,Y) returns 1 if X and Y unify

  otherwise returns $exp(\frac{-||e_X - e_Y||_2}{2\mu^2})$

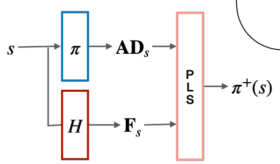
  grandPaOf(X,Y) :- softunification(grandPaOf,R), R(X,Y).

Probabilistic Logic Shield for Reinforcement Learning

Wen-chi Yang et al, IJCAI 23 Distinguished paper award

Assuming noisy sensors

$$\begin{cases} \pi(\texttt{accelerate} \mid s) = 0.5 \\ \pi(\texttt{left} \mid s) = 0.3 \\ \pi(\texttt{right} \mid s) = 0.2 \end{cases}$$



Shield

```
0.8 :: obstc(front).
0.2 :: obstc(left).
0.5 :: obstc(right).

0.5 :: act(accel);
0.3 :: act(left);
0.2 :: act(right)

0.9 :: crash:- obstc(front), act(accel).
0.4 :: crash:- obstc(left), act(left).
0.4 :: crash:- obstc(right), act(right).
safe:- ¬crash.
```

DeepProbLog Theory (Manhaeve et al. AIJ)

Will stay undamaged?

$$\mathbf{P}(\texttt{safe} \mid a, s) = \begin{cases} \texttt{accelerate} & \to 0.28 \\ \texttt{left} & \to 0.92 \\ \texttt{right} & \to 0.8 \end{cases}$$

Probability of staying safe if following π?

$$\mathbf{P}_{\pi}(\texttt{safe} \mid s) = 0.576$$

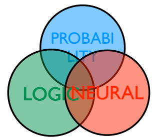
What is a safer policy π^+?

$$\begin{cases} \pi^+(\texttt{accelerate} \mid s) = 0.24 \\ \pi^+(\texttt{left} \mid s) = 0.48 \\ \pi^+(\texttt{right} \mid s) = 0.28 \end{cases}$$

DeepStochLog : Neural Definite Clause Grammars

PART 3 B

DeepStochLog : Neural Definite Clause Grammars

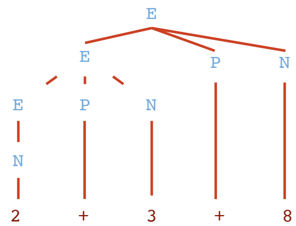


DeepStochLog

- Little sibling of DeepProbLog [Winters, Marra, et al AAAI 22]
- Based on a different semantics
  - **probabilistic graphical models vs grammars**
  - **random graphs vs random walks**
- Underlying StarAI representation is **Stochastic Logic Programs** (Muggleton, Cussens)
  - close to Probabilistic Definite Clause Grammars, ako probabilistic unification based grammar formalism
  - again the idea of neural predicates
- Scales better, is faster than DeepProbLog

CFG: Context-Free Grammar

```
E --> N
E --> E, P, N

P --> ["+"]

N --> ["0"]
N --> ["1"]
...
N --> ["9"]
```

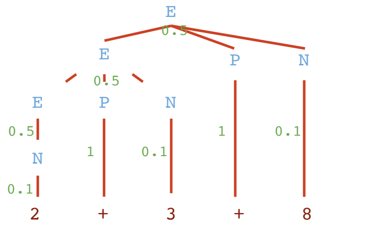


Useful for:
- Is sequence an element of the specified language?
- What is the *"part of speech"*-tag of a terminal
- Generate all elements of language

PCFG: Probabilistic Context-Free Grammar

```
0.5 :: E --> N
0.5 :: E --> E, P, N

1.0 :: P --> ["+"]

0.1 :: N --> ["0"]
0.1 :: N --> ["1"]
       …
0.1 :: N --> ["9"]
```

Always sums to 1 per non-terminal



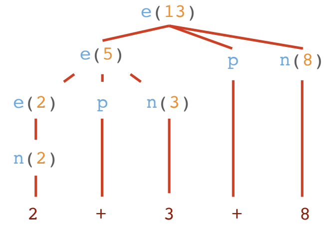
*Probability of this parse = 0.5*0.5*0.5*0.1*1*0.1*1*0.1 = 0.000125*

Useful for:

- What is the most likely parse for this sequence of terminals? *(useful for ambiguous grammars)*
- What is the probability of generating this string?

DCG: Definite Clause Grammar

```
e(N) --> n(N).
e(N) --> e(N1), p, n(N2),
         {N is N1 + N2}.
p    --> ["+"].

n(0) --> ["0"].
n(1) --> ["1"].
…
n(9) --> ["9"].
```

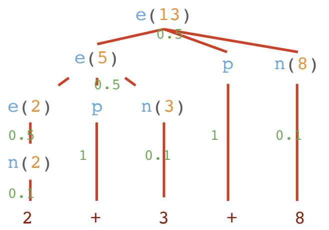


Useful for:

- Modelling more complex languages *(e.g. context-sensitive)*
- Adding constraints between non-terminals thanks to Prolog power *(e.g. through unification)*
- Extra inputs & outputs aside from terminal sequence *(through unification of input variables)*

SDCG: Stochastic Definite Clause Grammar

```
0.5 :: e(N) --> n(N).
0.5 :: e(N) --> e(N1), p, n(N2),
                {N is N1 + N2}.
1.0 :: p    --> ["+"].

0.1 :: n(0) --> ["0"].
0.1 :: n(1) --> ["1"].
       …
0.1 :: n(9) --> ["9"].
```



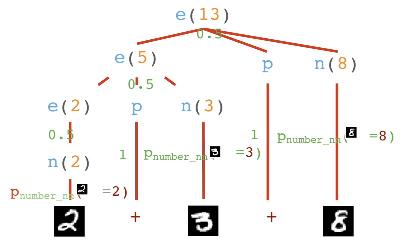
*Probability of this parse = 0.5*0.5*0.5*0.1*1*0.1*1*0.1 = 0.000125*

Useful for:

- Same benefits as PCFGs give to CFG *(e.g. most likely parse)*
- But: loss of probability mass possible due to failing derivations

NDCG: Neural Definite Clause Grammar (= DeepStochLog)

```
0.5 :: e(N) --> n(N).
0.5 :: e(N) --> e(N1), p, n(N2),
                {N is N1 + N2}.
1.0 :: p    --> ["+"].
nn(number_nn,[X],[Y],[digit])::
        n(Y) -> [X].
digit(Y) :-
      member(Y,[0,1,2,3,4,5,6,7,8,9]).
```



Probability of this parse =

$0.5*0.5*0.5*p_{number\_nn}(\ =2)*1\ p_{number\_nn}(\ =3)*1*p_{number\_nn}(\ =8)$

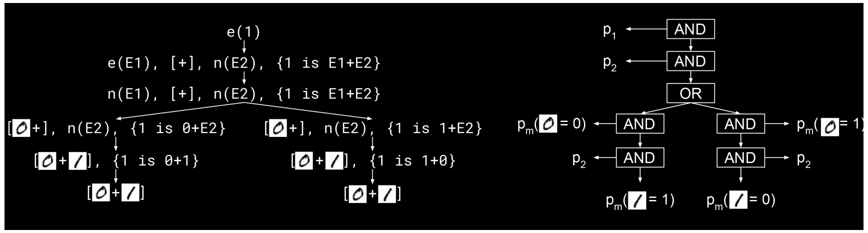
Useful for:

- Subsymbolic processing: e.g. tensors as terminals
- Learning rule probabilities using neural networks

DeepStochLog Inference

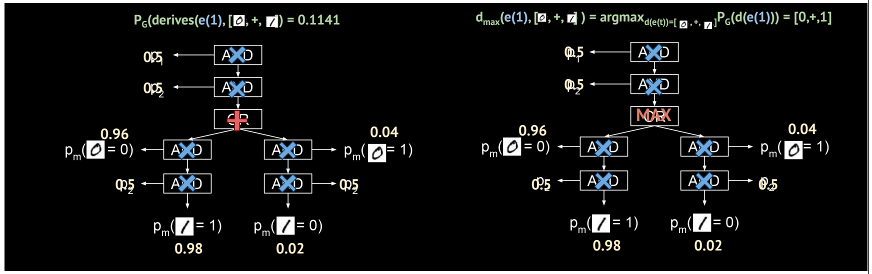
Deriving probability of goal for given terminals in NDCG

Proof derivations d(e(1), [0+1])   then turn it into and/or tree



And/Or tree + semiring for different inference types

Probability of goal   Most likely derivation



Inference optimisation

- Inference is **optimized** using
  - **SLG resolution**: Prolog tables the returned proof tree(s), and thus creates forest
    → Allows for reusing probability calculation results from intermediate nodes

Table 6: **Q4** Parsing time in seconds (**T2**). Comparison of the DeepStochLog with and without tabling (SLD vs SLG resolution).

Lengths	# Answers	No Tabling	Tabling
1	10	0.067	0.060
3	95	0.081	0.096
5	1066	3.78	0.95
7	10386	30.42	10.95
9	68298	1494.23	132.26
11	416517	timeout	1996.09

  - **Batched network calls**: Evaluate all the required neural network queries first
    → Very natural for neural networks to evaluate multiple instances at once using batching
    & less overhead in logic & neural network communication

Mathematical expression outcome

T1: Summing MNIST numbers with pre-specified # digits

 **= 137**

T2: Expressions with images representing operator or single digit number.

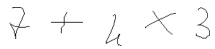
 **= 19**

Table 1: The test accuracy (%) on the MNIST addition (**T1**).

Methods	Number of digits per number (N) 1	2	3	4
NeurASP	97.3 ± 0.3	93.9 ± 0.7	timeout	timeout
DeepProbLog	97.2 ± 0.5	95.2 ± 1.7	timeout	timeout
DeepStochLog	97.9 ± 0.1	96.4 ± 0.1	94.5 ± 1.1	92.7 ± 0.6

Table 2: The accuracy (%) on the HWF dataset (**T2**).

Method	Expression length 1	3	5	7
NGS	90.2 ± 1.6	85.7 ± 1.0	91.7 ± 1.3	20.4 ± 37.2
DeepProbLog	90.8 ± 1.3	85.6 ± 1.1	timeout	timeout
DeepStochLog	90.8 ± 1.0	86.3 ± 1.9	92.1 ± 1.4	94.8 ± 0.9

Classic grammars, but with MNIST images as terminals

T3: Well-formed brackets as input (without parse). Task: predict parse.

→ parse = ( ) ((( ) ( ))

T4: inputs are strings $a^k b^l c^m$ (or permutations of [a,b,c], and (k+l+m) mod 3=0). Predict 1 if k=l=m, otherwise 0.

 **= 1**

 **= 0**

Table 3: The parse accuracy (%) on the well-formed parentheses dataset (**T3**).

Method	Maximum expression length 10	14	18
DeepProbLog	100.0 ± 0.0	99.4 ± 0.5	99.2 ± 0.8
DeepStochLog	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0

Table 4: The accuracy (%) on the $a^n b^n c^n$ dataset (**T4**).

Method	Expression length 3-12	3-15	3-18
DeepProbLog	99.8 ± 0.3	timeout	timeout
DeepStochLog	99.4 ± 0.5	99.2 ± 0.4	98.8 ± 0.2

Citation networks

T5: Given scientific paper set with only few labels & citation network, find all labels

Table 5: **Q3** Accuracy (%) of the classification on the test nodes on task **T5**

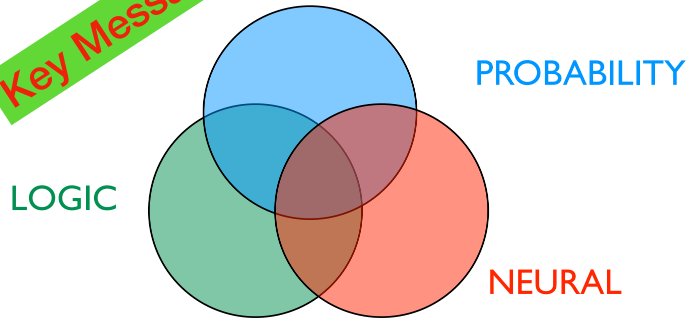
Method	**Citeseer**	**Cora**
ManiReg	60.1	59.5
SemiEmb	59.6	59.0
LP	45.3	68.0
DeepWalk	43.2	67.2
ICA	69.1	75.1
GCN	70.3	81.5
DeepProbLog	timeout	timeout
DeepStochLog	65.0	69.4

Challenges

- For NeSy,
  - scaling up
  - which models and which knowledge to use
  - large scale life applications
  - peculiarities of neural nets & fuzzy logic
  - dynamics / continuous
- This is an excellent area for starting researchers / PhDs

Neurosymbolic =
Neuro + Logic + Probability

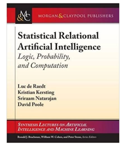
Key Message 1



see Manhaeve et al. NeSy Book

interpret PROBABILITY broadly (including fuzzy)

Key Message 2



StarAI and NeSy share similar problems and thus similar solutions apply

See also [De Raedt et al., IJCAI 20; Marra et al, arxiv]

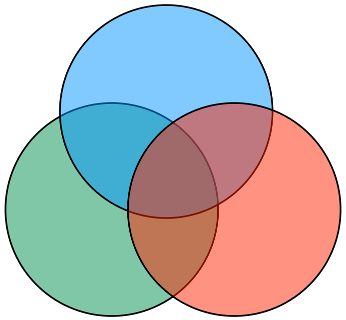
Key Message 3

Provide recipe for

Kautz

Neural : : Symbolic

“an interface layer (<> pipeline) between neural & symbolic components”



THANKS