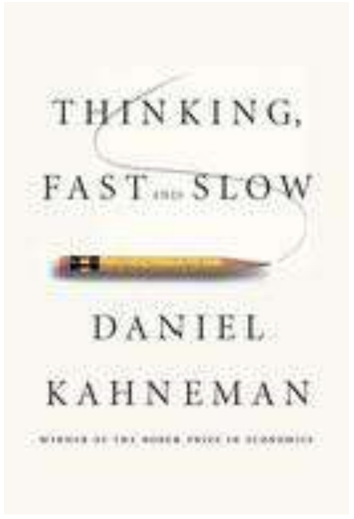
From Statistical Relational AI to Neural Symbolic Computation

Luc De Raedt
luc.deraedt@cs.kuleuven.be

joint work with Robin Manhaeve, Angelika Kimmig, Giuseppe Marra, Sebastijan Dumancic, Thomas De Meester, Thomas Winters

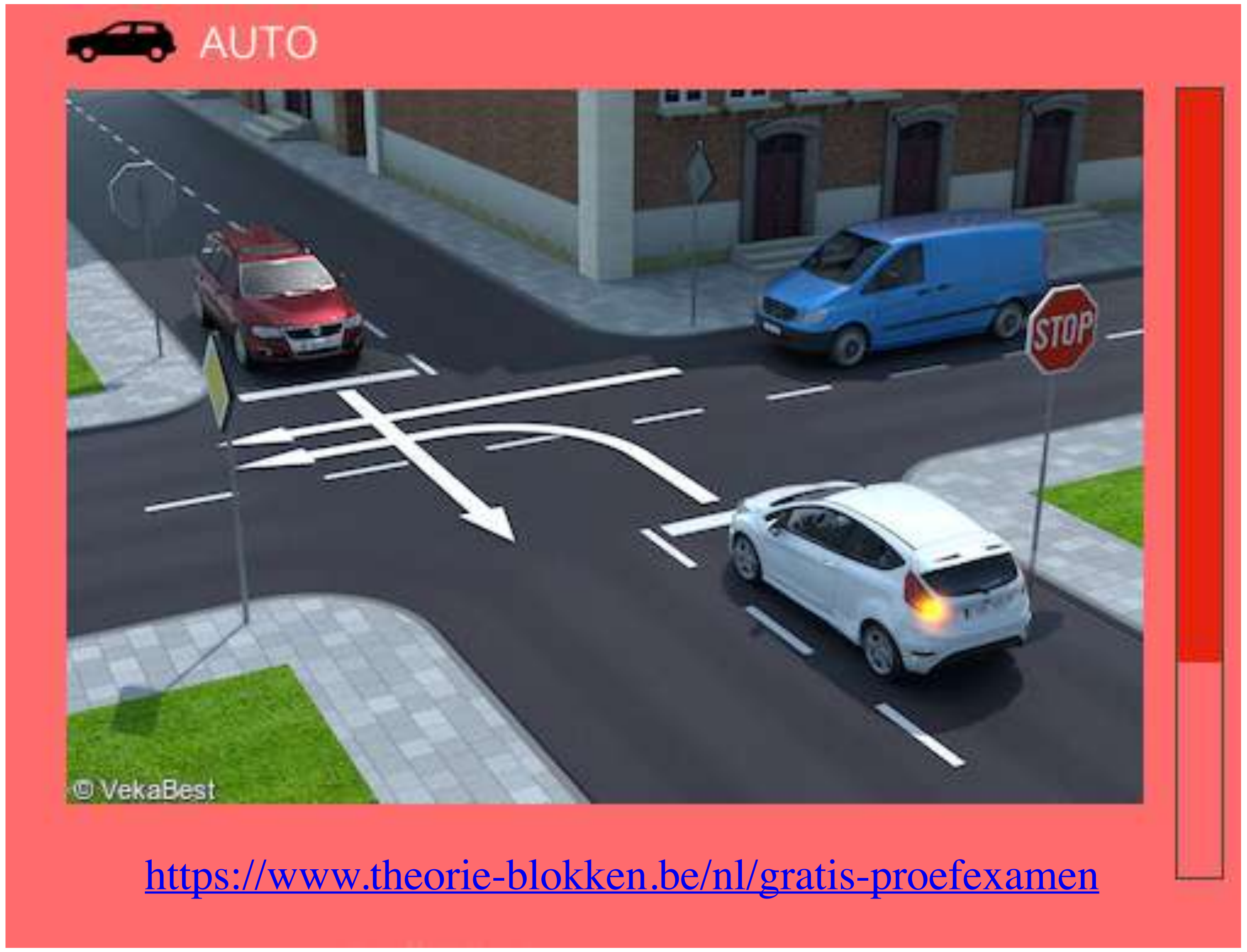
Learning and Reasoning both needed



- 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

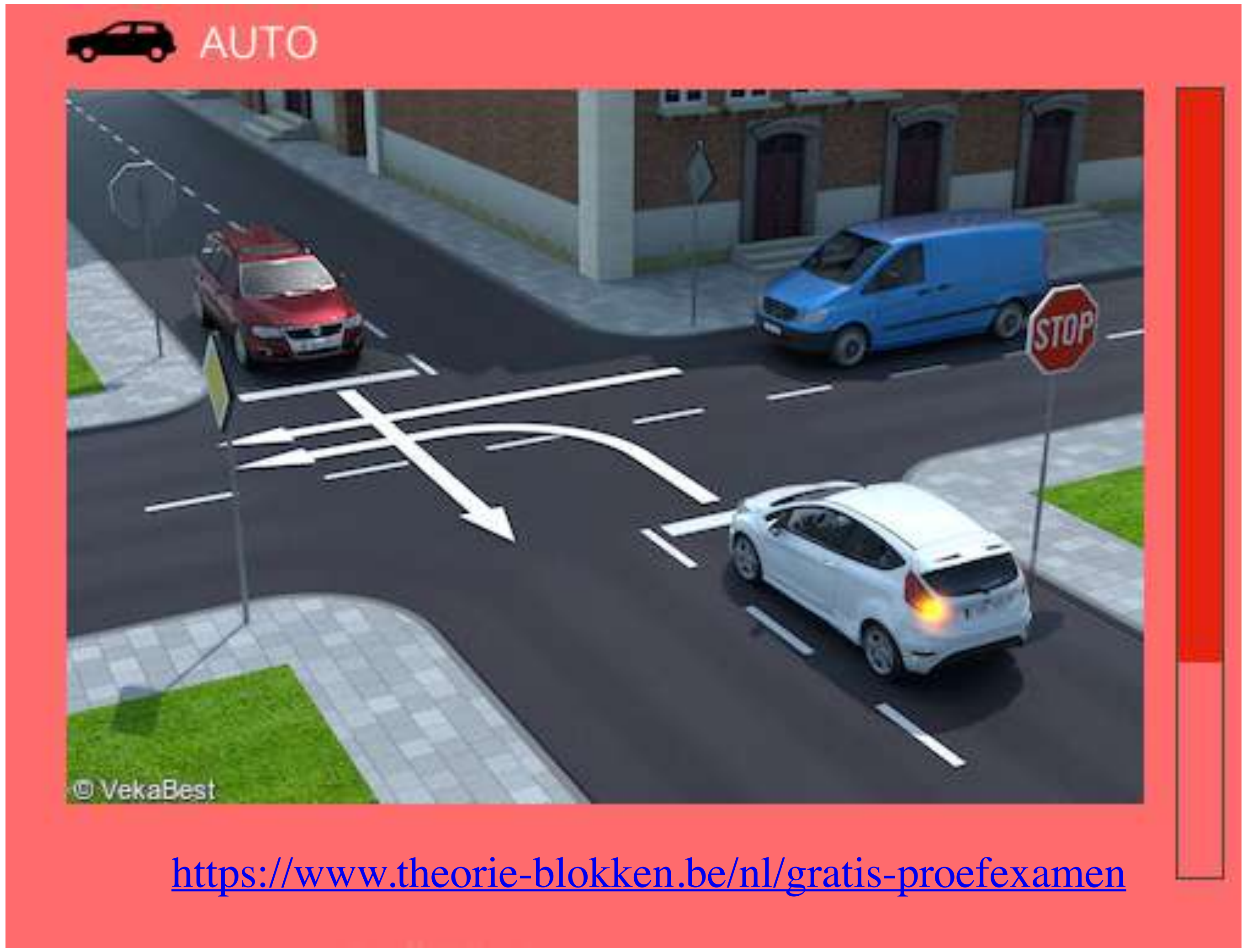
Who can go first ?

A. The red car

B. The blue van

C. The white car

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

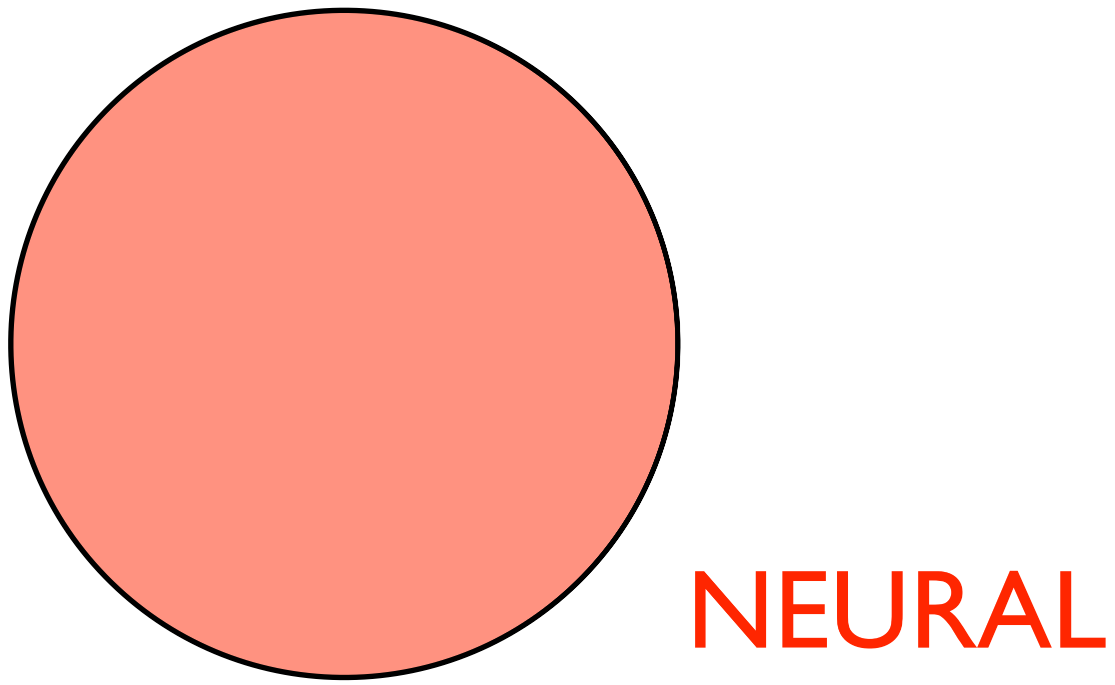
C. The white car

Reasoning

Sub-symbolic perception

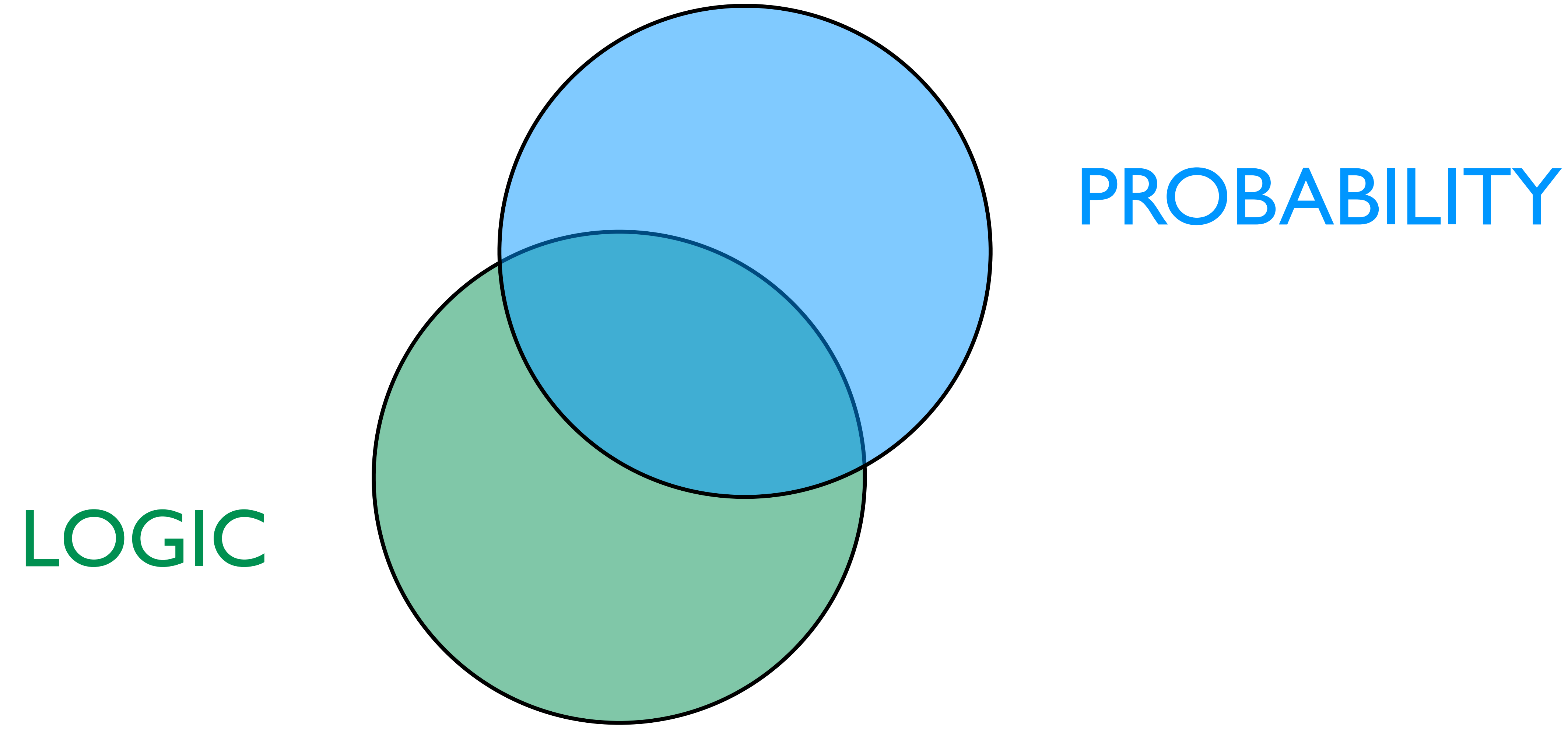
Thinking fast

MAIN PARADIGM in AI
Focus on Learning



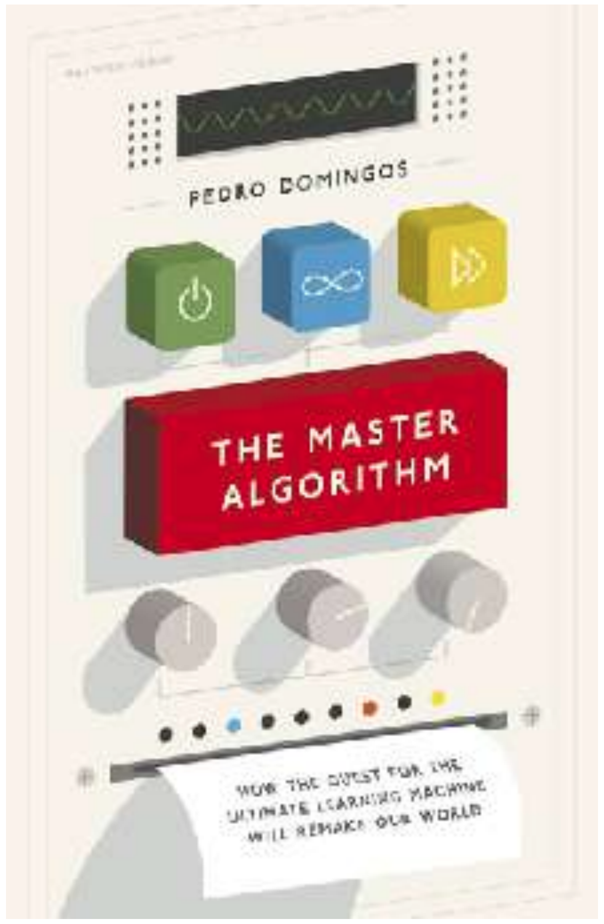
Thinking slow = reasoning

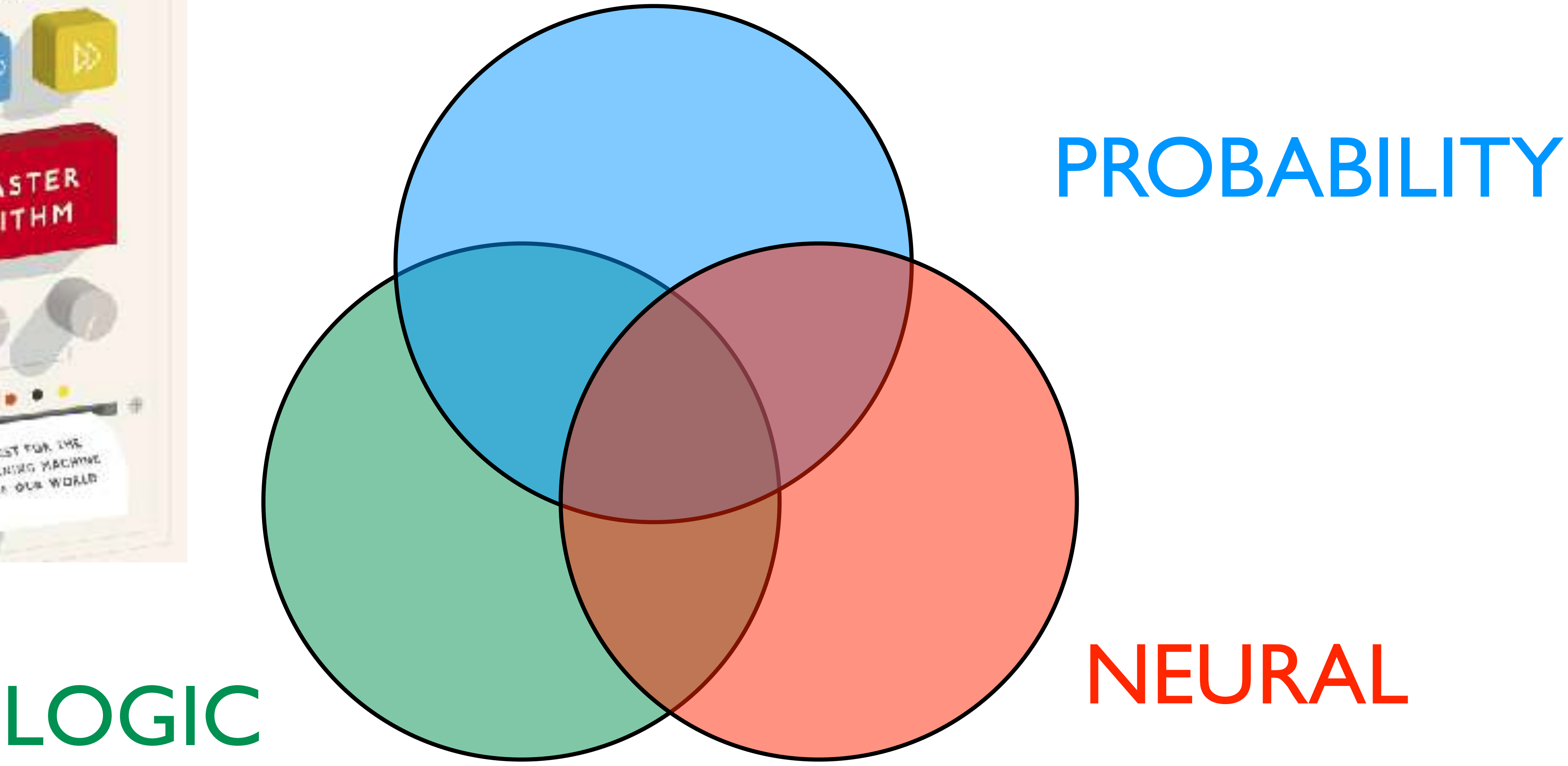
TWO MAIN PARADIGMS in AI



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

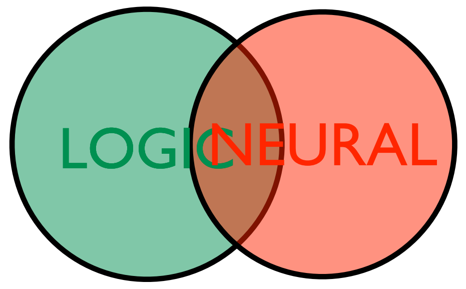
Integrating learning and reasoning


PEDRO DOMINGOS
THE MASTER ALGORITHM
HOW THE QUEST FOR THE ULTIMATE LEARNING MACHINE WILL REMAKE OUR WORLD

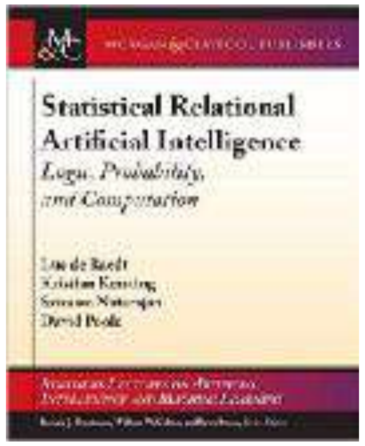


How to integrate these three paradigms in AI ?

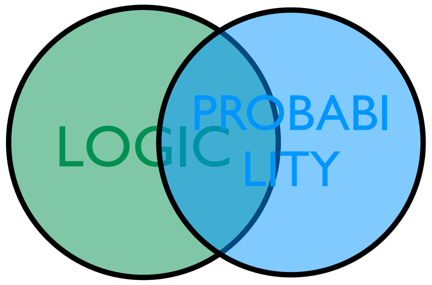
Neural Symbolic Computation:

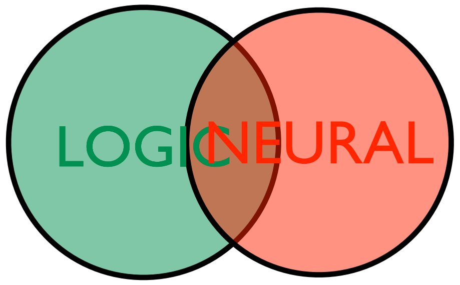


- **Neural symbolic computation** is the area combining logic / symbolic reasoning and neural networks


Statistical Relational Artificial Intelligence
Logic, Probability, and Computation
Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

Key Message 1

FROM 

TO 

StarAI and NeSy share similar problems and thus similar solutions apply

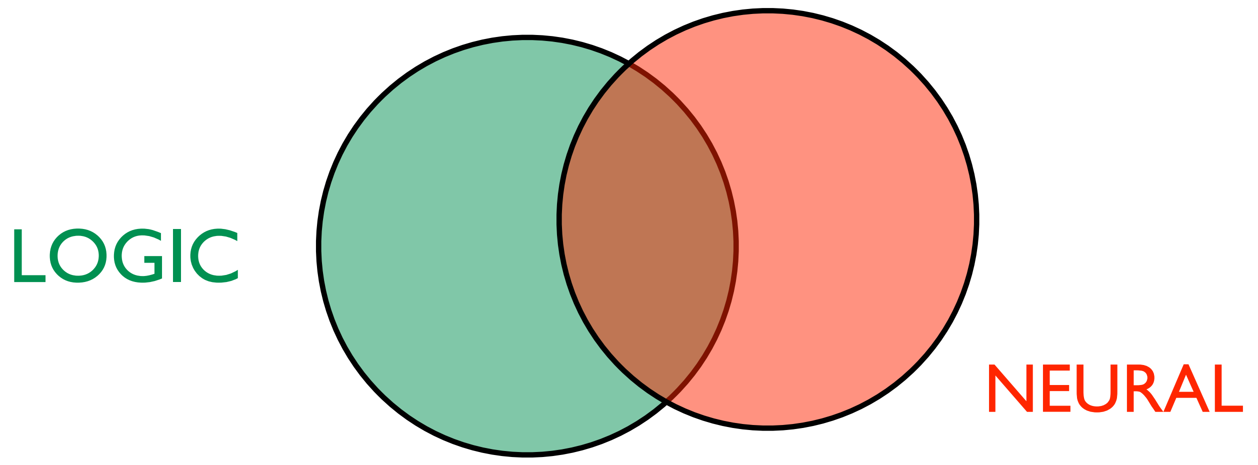
WARNING

TALK MAY NOT COVER ALL of NESY

PART 1 of the talk

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

Neural Symbolic Computation: state-of-the-art



- Neural symbolic computation is the area combining logic / symbolic reasoning and neural networks

- **Most NeSy approaches :** inject the logic/knowledge into neural networks, and let the neural network do the rest

- **Downside :** relies only on neural networks -> the power of reasoning, explanation and trust is (at least partly) lost

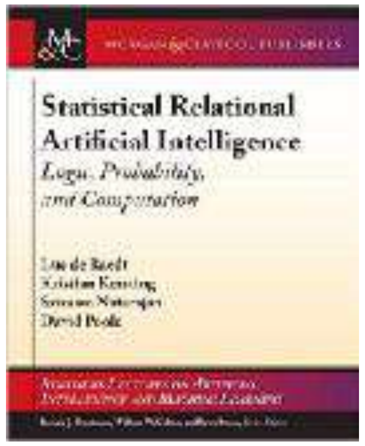
Key Message 2

A different approach

A true integration T of X and Y should allow to reconstruct X and Y as special cases of T

Thus, Neural Symbolic approaches should have both logic and neural networks as special cases

PART 2 of the talk — illustration with DeepProbLog [NeurIPS 2018]



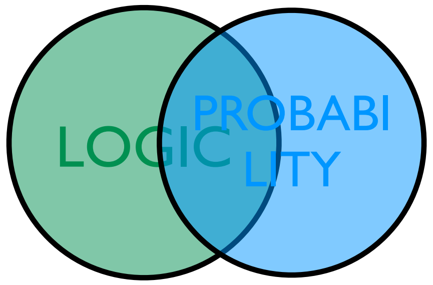
Statistical Relational Artificial Intelligence

Logic, Probability, and Computation

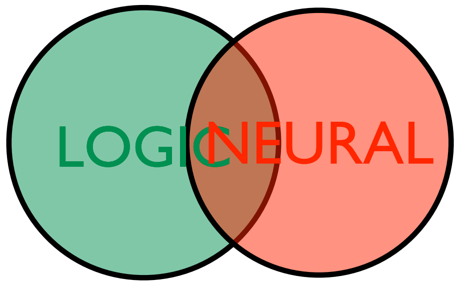
Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

PART 1

FROM



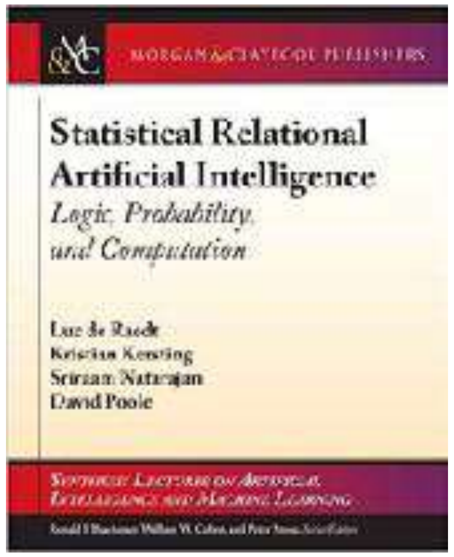
TO



Key Message 1

StarAI and NeSy share similar problems and thus similar solutions apply

There are two basic types of (uses of) logic, graphical models, and neural symbolic models


MORGAN & CLAYPOOL PUBLISHERS

Statistical Relational Artificial Intelligence

Logic, Probability, and Computation

Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

Synthesis Lectures on Artificial Intelligence and Machine Learning

Ronald J. Brachman, William W. Cohen, and Peter Stone, Series Editors

Logic Programs

as in the programming language Prolog

Propositional logic program

```
burglary.
hears_alarm_mary.


earthquake.
hears_alarm_john.
```

facts :
burglary = true

```
alarm :– earthquake.
alarm :– burglary.

calls_mary :– alarm, hears_alarm_mary.
calls_john :– alarm, hears_alarm_john.
```

LOGIC

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.
```

rule:
calls_mary =true IF alarm = true AND hears_alarm_mary = true

LOGIC

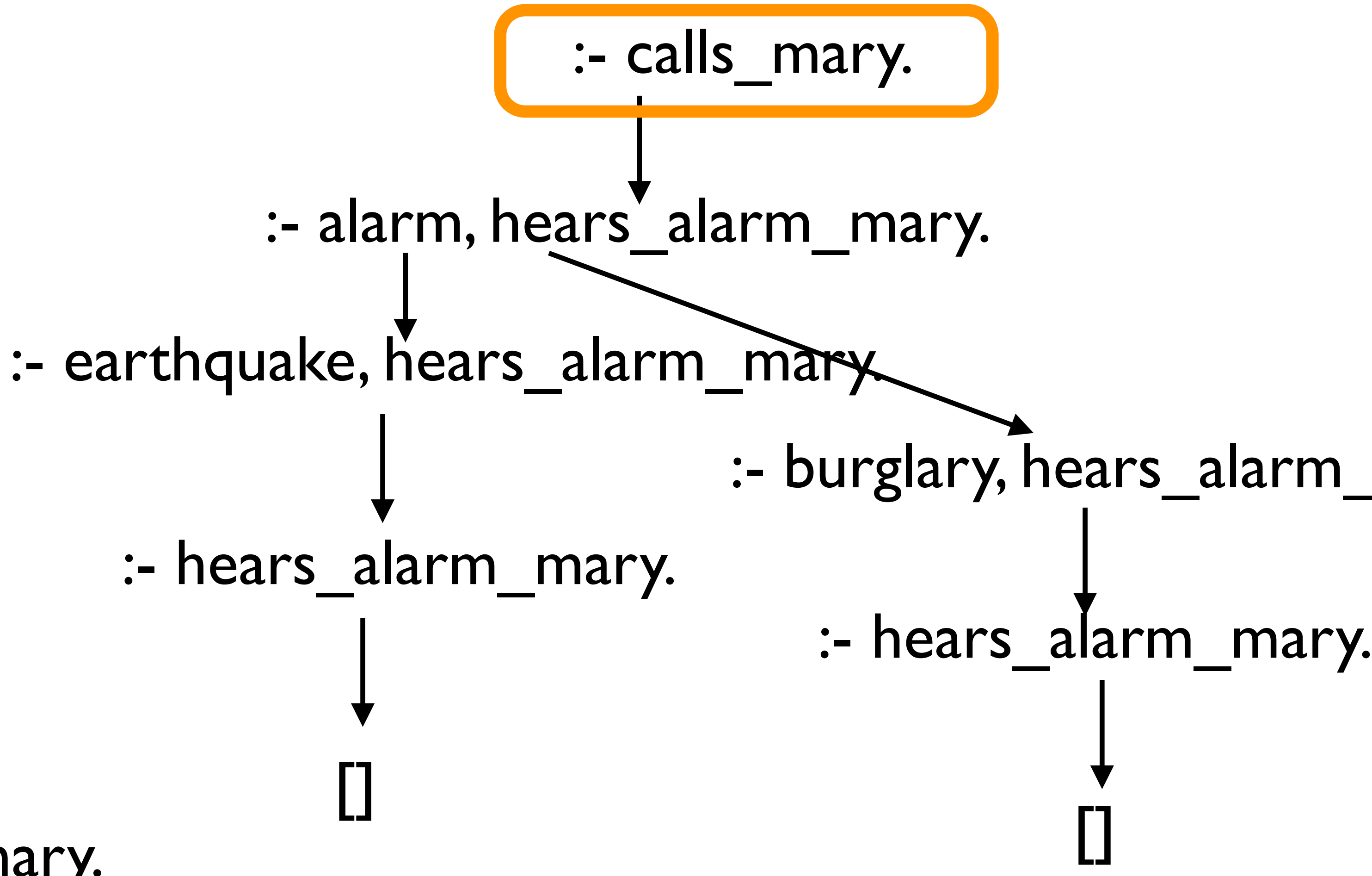
Logic Programs

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

LOGIC

Logic as constraints

as in SAT solvers

Propositional logic

IFF AND

calls(mary) $\leftrightarrow$ hears_alarm(mary) $\wedge$ alarm

calls(john) $\leftrightarrow$ hears_alarm(john) $\wedge$ alarm

OR

alarm $\leftrightarrow$ earthquake v burglary

Model / Possible World

{ burglary,

hears_alarm(john),

alarm,

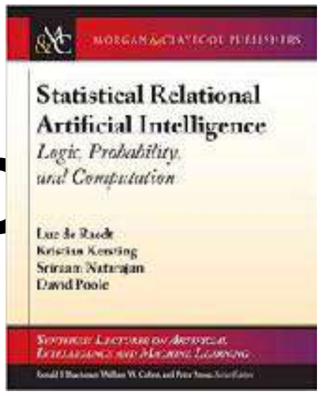
calls(john)}

the facts that are true
in this model / possible world

A model-theoretic view

LOGIC

Two types of probabilistic graphic models and StarAI systems


Morgan & Claypool Publishers
Statistical Relational Artificial Intelligence
Logic, Probability, and Computation
Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole
Synthesis Lectures on Artificial Intelligence and Machine Learning

0.1 :: burglary.

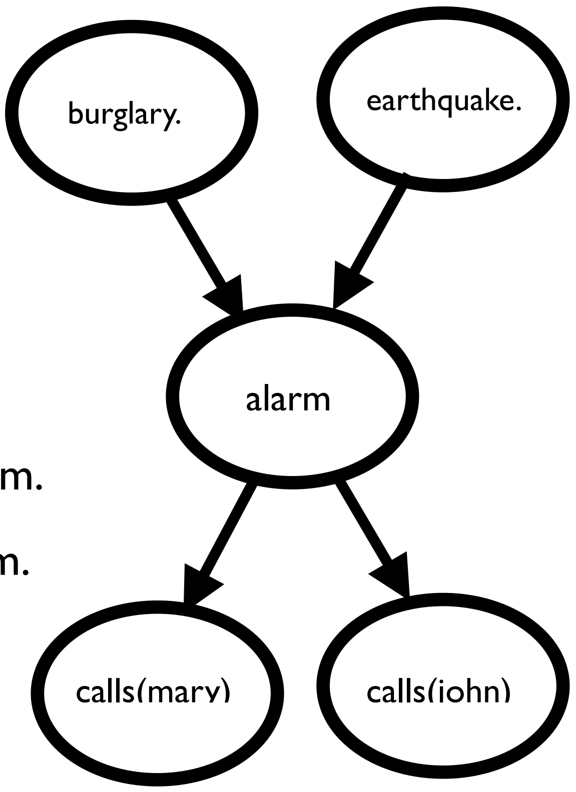
0.05 :: earthquake.

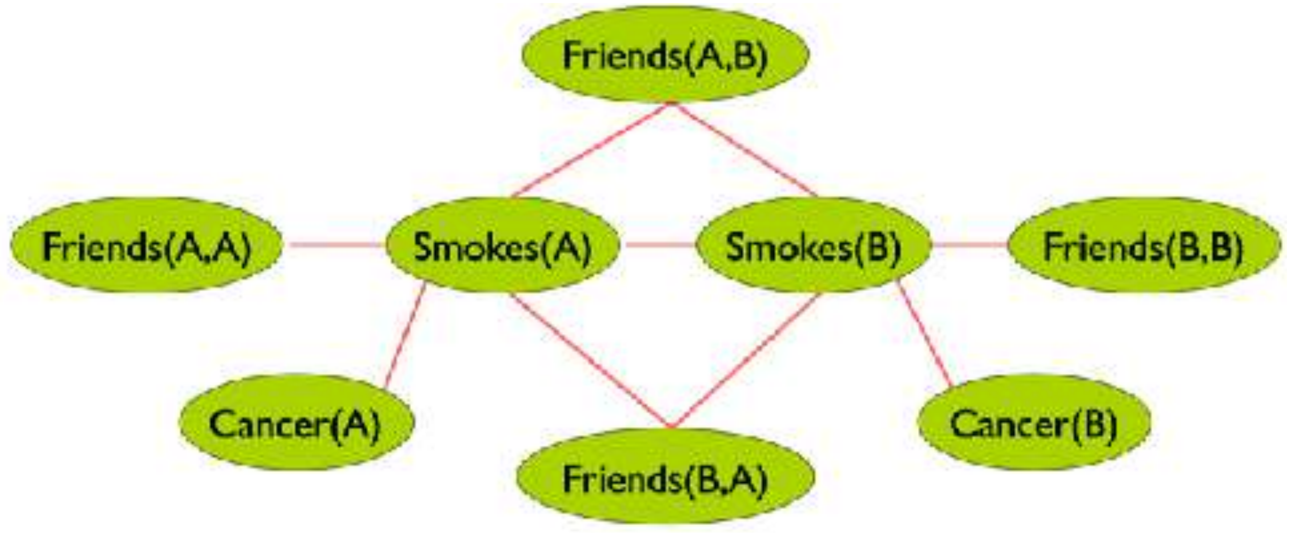
alarm :– earthquake.

alarm :– burglary.

0.7::calls(mary) :– alarm.

0.6::calls(john) :– alarm.





$$1.5 \quad \forall x \; Smokes(x) \Rightarrow Cancer(x)$$
$$1.1 \quad \forall x, y \; Friends(x,y) \Rightarrow \big(Smokes(x) \Leftrightarrow Smokes(y)\big)$$

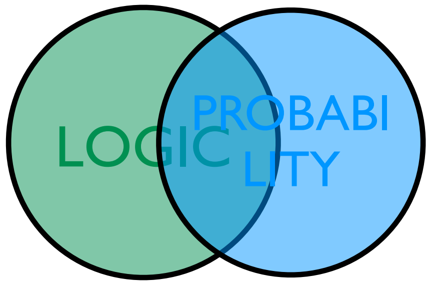
Probabilistic Logic Programs
ProbLog

directed
Bayesian Net

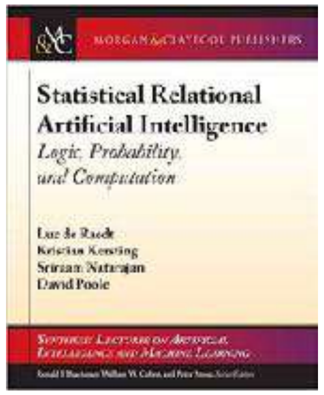
Markov Logic

undirected
Markov Net
model theoretic

key representatives



Two types of Neural Symbolic Systems


MORGAN & CLAYPOOL PUBLISHERS

Statistical Relational Artificial Intelligence

Logic, Probability, and Computation

Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

Synthesis Lectures on Artificial Intelligence and Machine Learning

Just like in StarAI

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

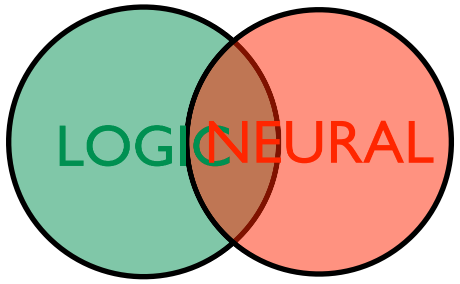
directed StarAI approach and logic programs

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 as a neural program

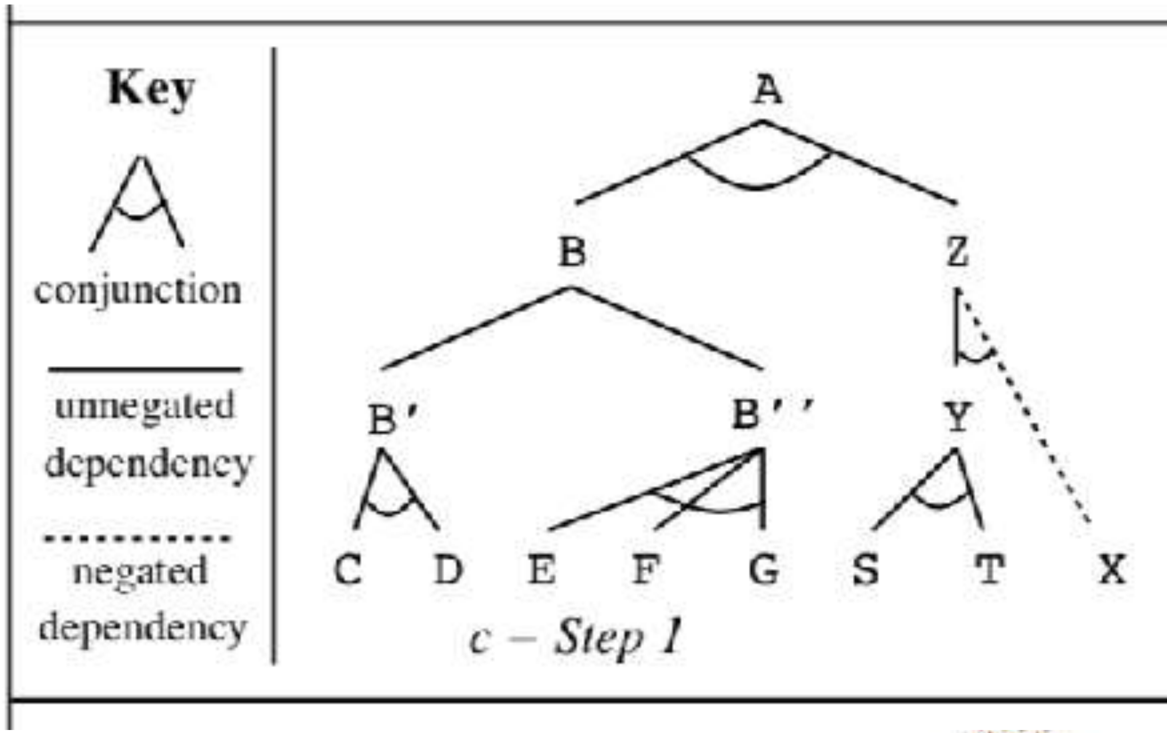
directed StarAI approach and logic programs

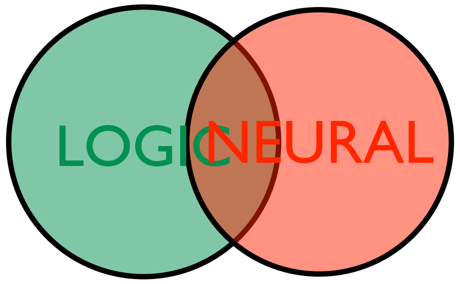
- KBANN (Towell and Shavlik AIJ 94)
- Turn a (propositional) Prolog program into a neural network and learn

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

REWRITE

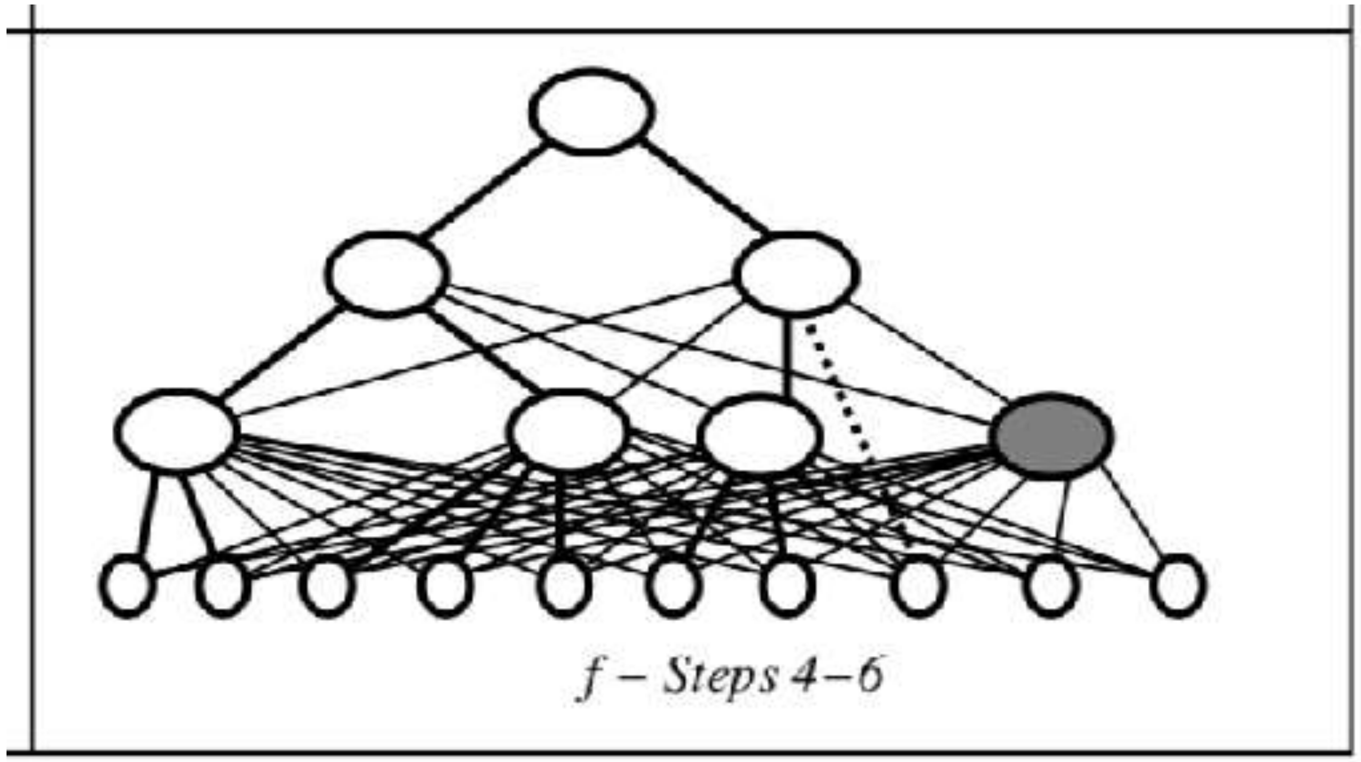
```
A    :- B, Z.
B    :- B'.
B    :- B''.
B'   :- C, D.
B''  :- E, F, G.
Z    :- Y, not X.
Y    :- S, T.
```

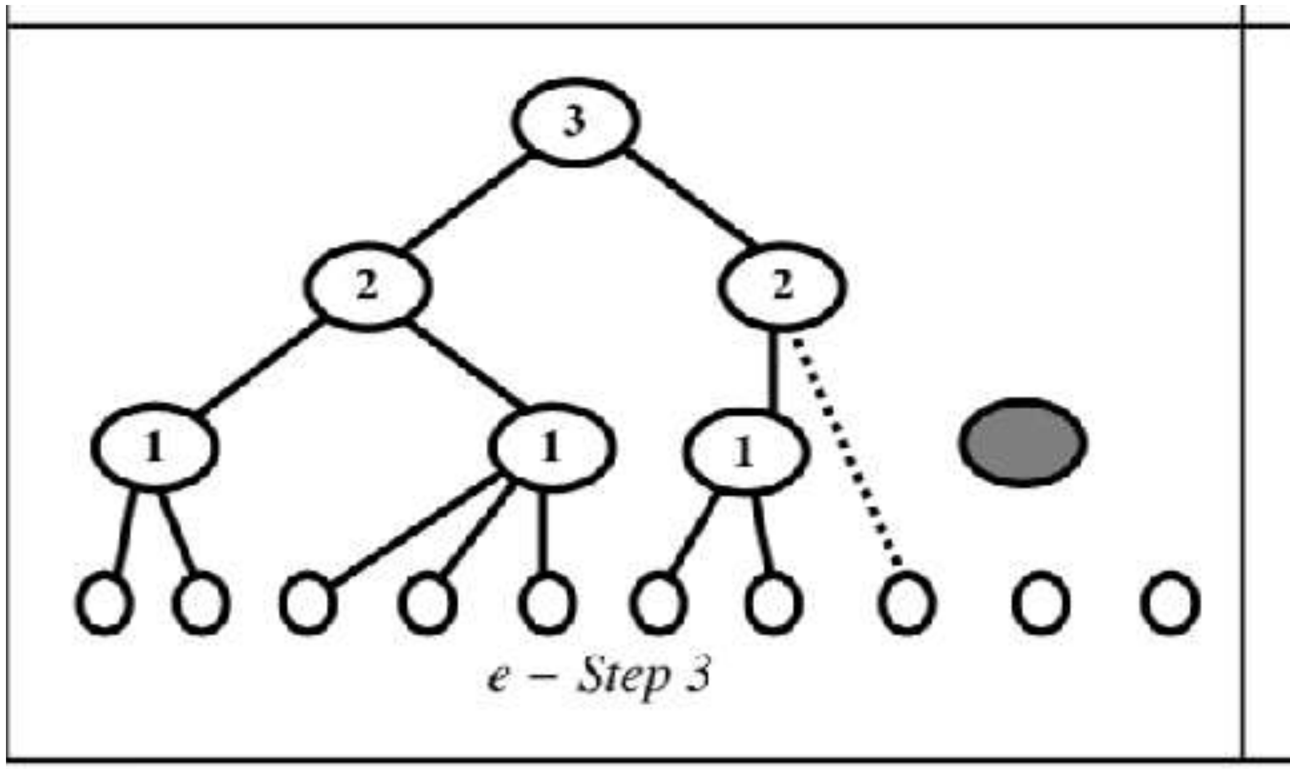




Logic as a neural program

directed StarAI approach and logic programs





ADD LINKS — ALSO SPURIOUS ONES

HIDDEN UNIT

and then learn

(Details of activation & loss functions not mentioned)

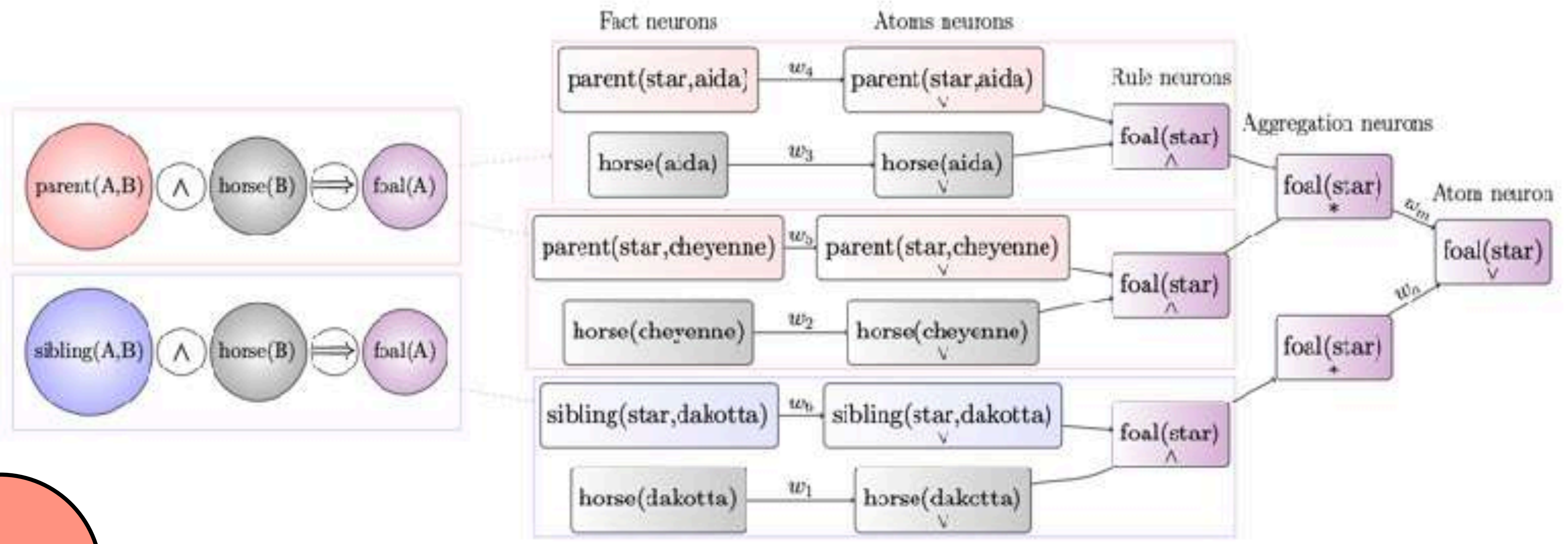
LOGIC NEURAL

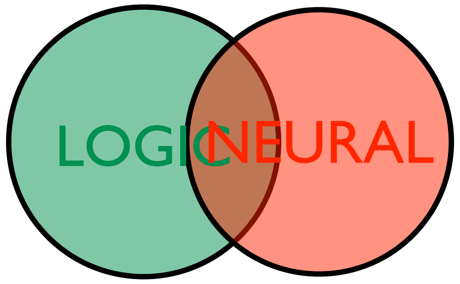
erc

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)

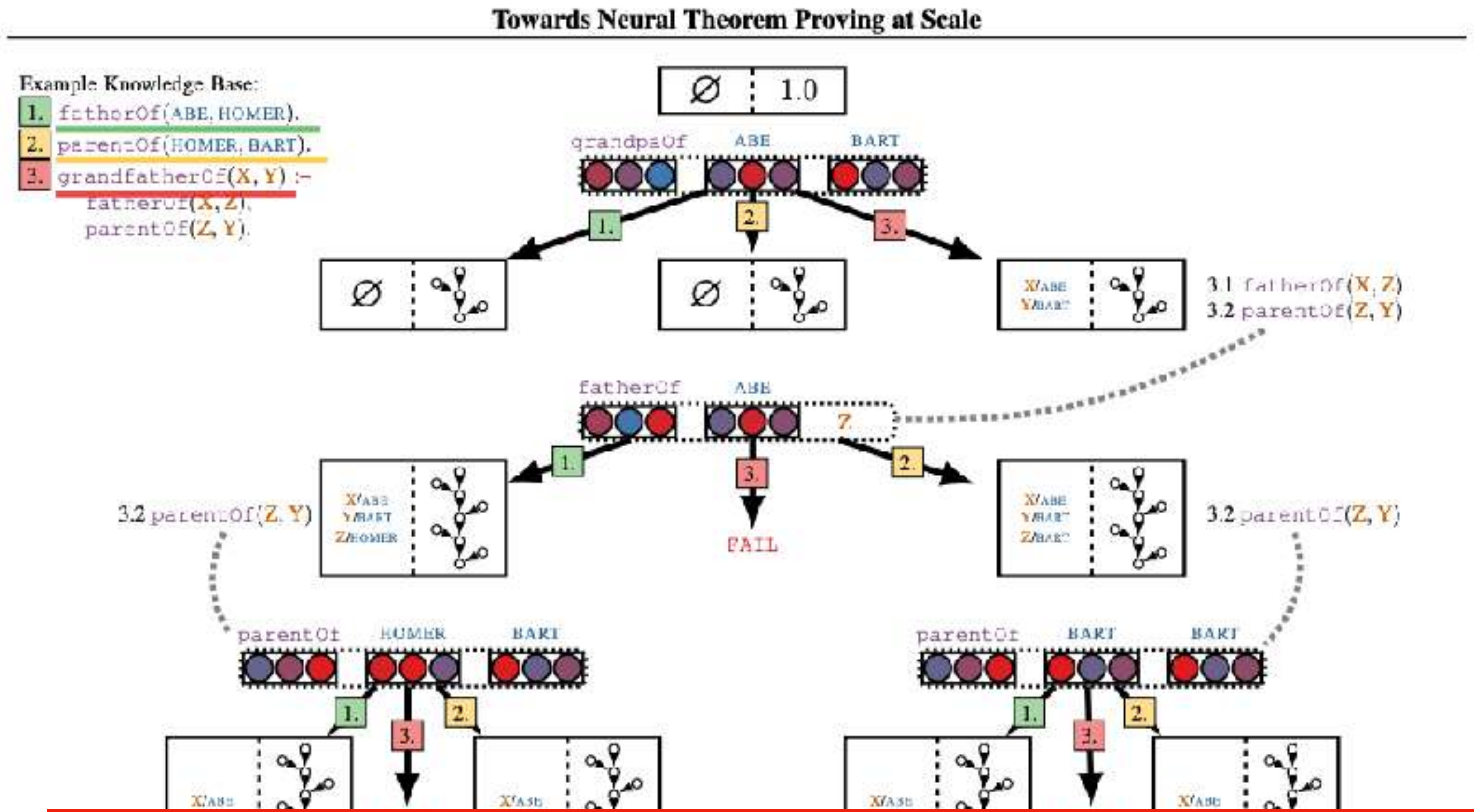




[Sourek, Kuzelka, et al JAIR]

directed StarAI approach and logic programs

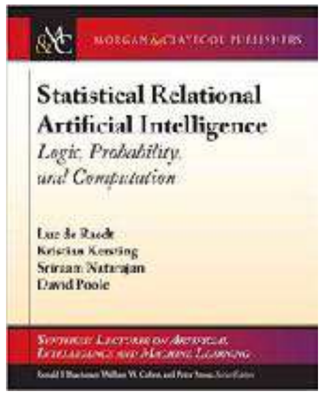
Neural Theorem Prover



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

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

Two types of Neural Symbolic Systems


MORGAN & CLAYPOOL PUBLISHERS

Statistical Relational Artificial Intelligence
Logic, Probability, and Computation

Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

Synthesis Lectures on Artificial Intelligence and Machine Learning

Just like in StarAI

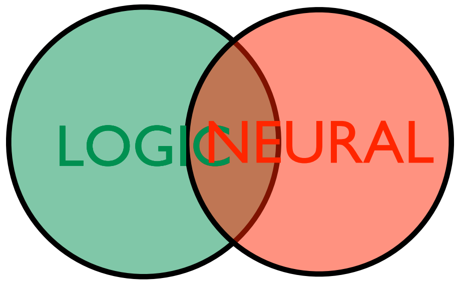
Logic as a kind of ***neural program***

directed StarAI approach and logic programs

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

undirected StarAI approach and (soft) constraints

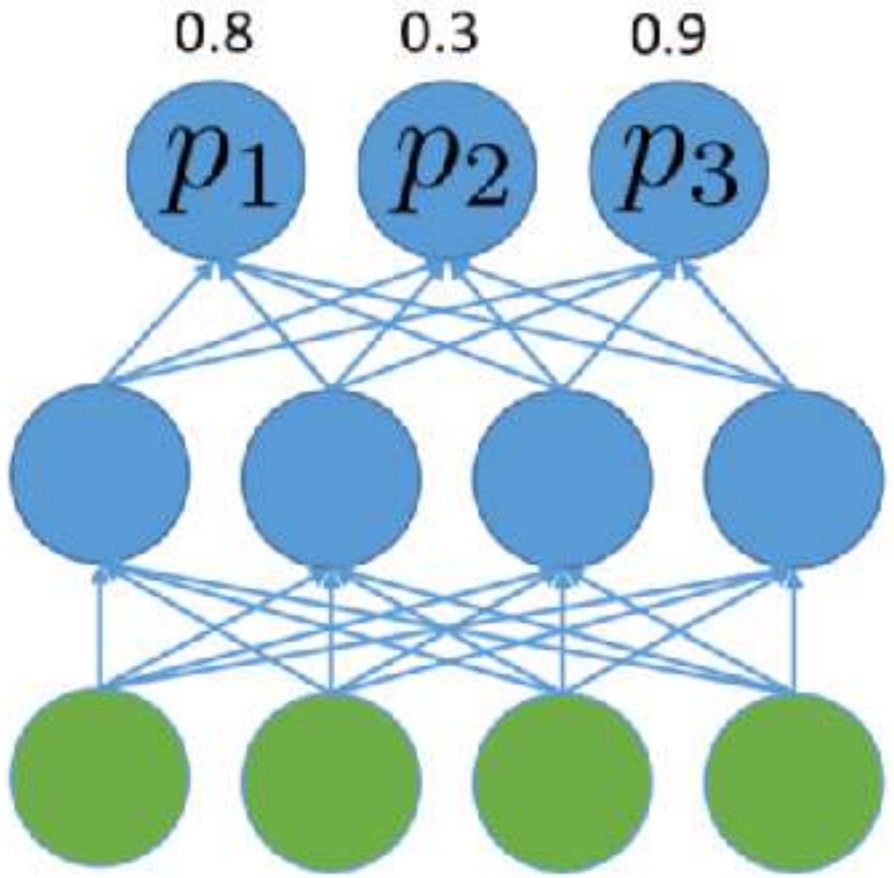
Also, many NeSy systems are doing
knowledge based model construction KBMC
where logic is used as a template



Logic as constraints

undirected StarAI approach and (soft) constraints

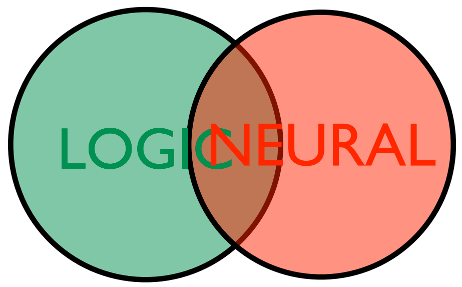
multi-class classification



This constraint should be satisfied

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

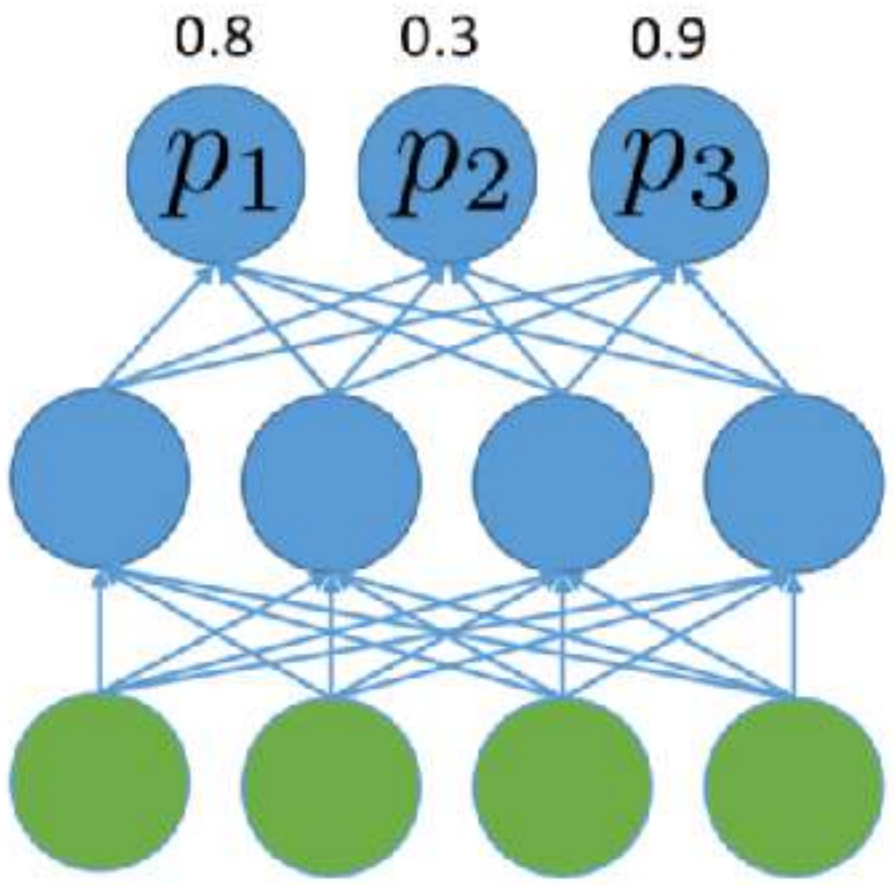
figures and example from Xu et al., ICML 2018



Logic as constraints

undirected StarAI approach and (soft) constraints

multi-class classification

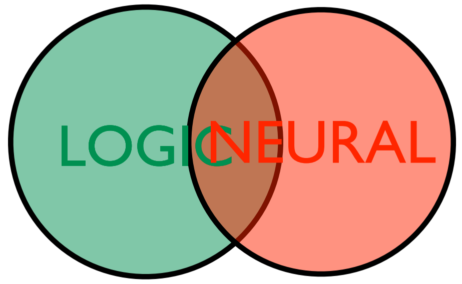


Probability that constraint is satisfied

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

basis for SEMANTIC LOSS

(weighted model counting)



Logic as a regularizer

undirected StarAI approach and (soft) constraints

Semantic Loss:

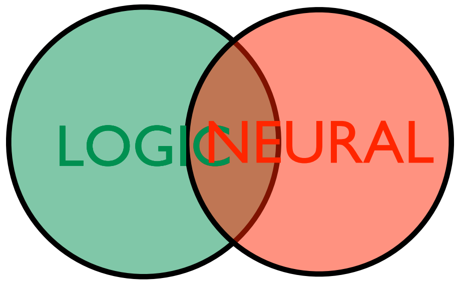
- Use logic as constraints (very much like "propositional MLNs)
- Semantic loss

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

- Used as regulariser

$$Loss = TraditionalLoss + w.SLoss$$

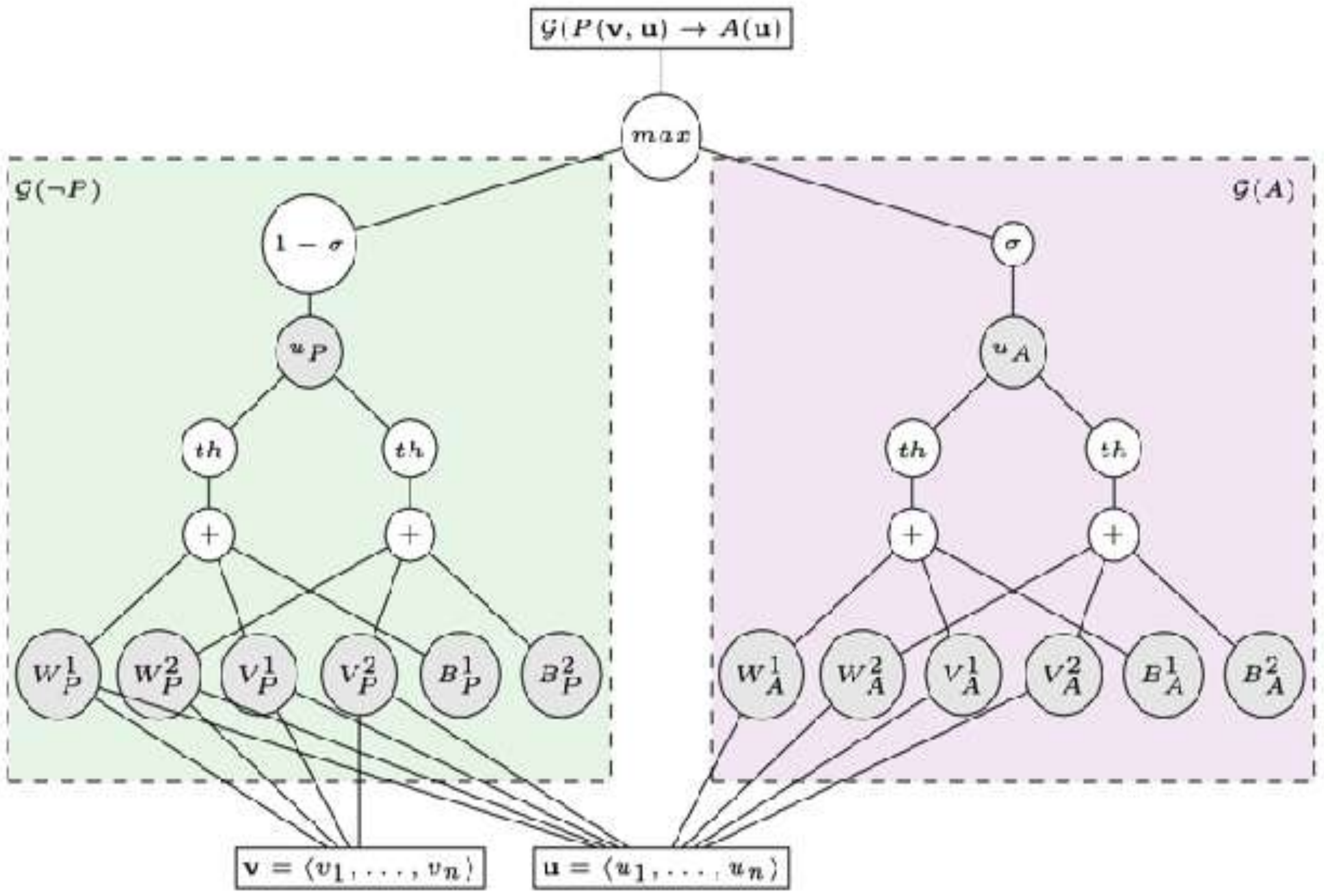
- Use weighted model counting , close to StarAI



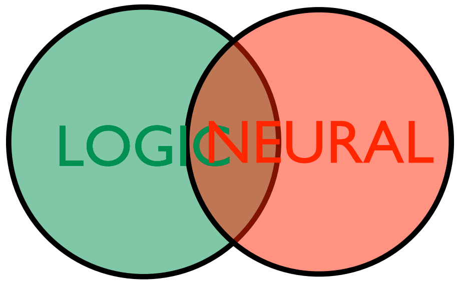
Logic Tensor Networks

undirected StarAI approach and (soft) constraints

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



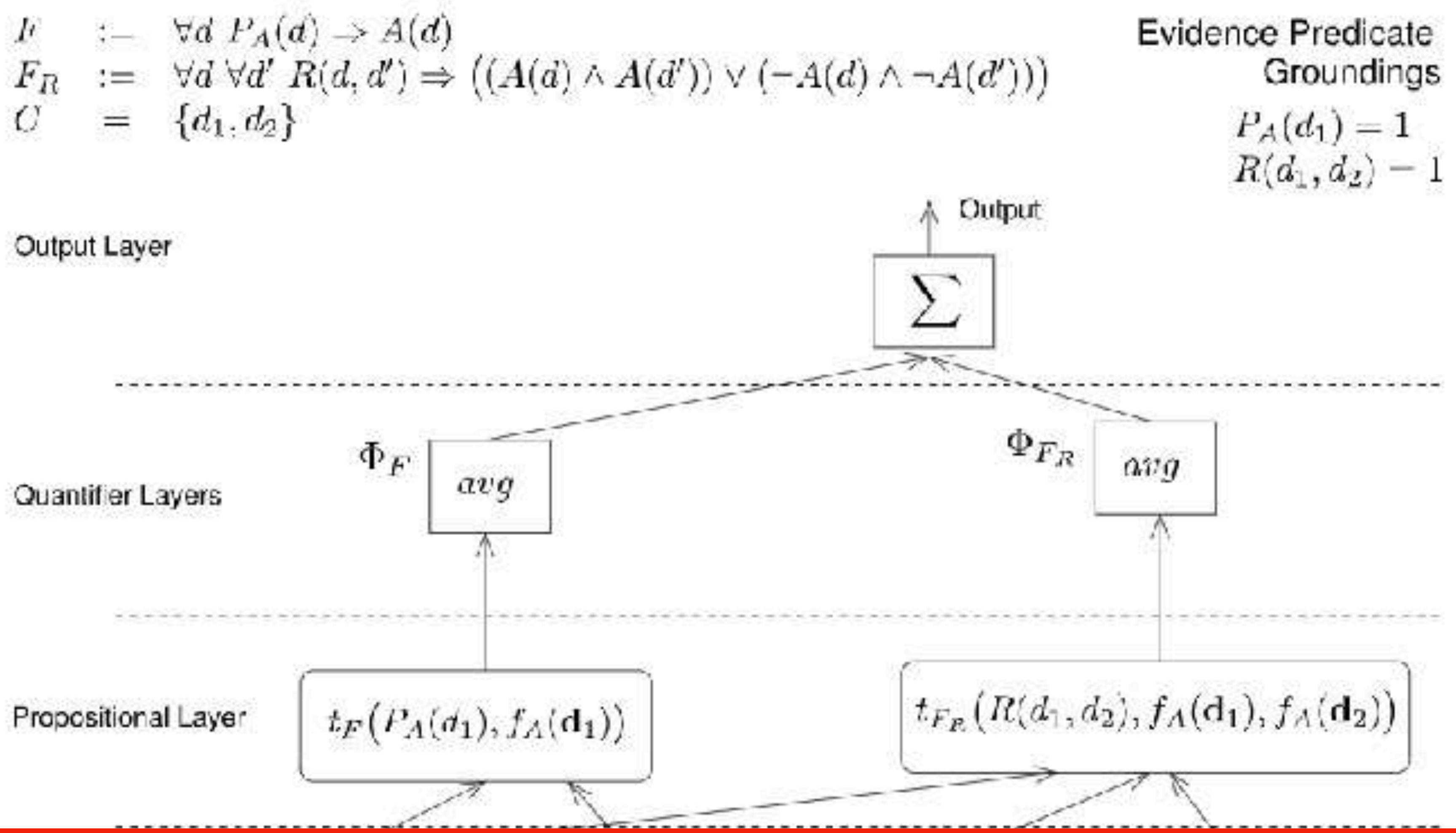
$\mathcal{G}(P(\mathbf{v},\mathbf{u}) \rightarrow A(\mathbf{u}))$
max
$\mathcal{G}(\neg P)$
$\mathcal{G}(A)$
$1-\sigma$
σ
u_P
u_A
th
+
W_P^1 W_P^2 V_P^1 V_P^2 B_P^1 B_P^2
W_A^1 W_A^2 V_A^1 V_A^2 B_A^1 B_A^2
$\mathbf{v} = \langle v_1, \ldots, v_n \rangle$
$\mathbf{u} = \langle u_1, \ldots, u_n \rangle$



Serafini & Garcez

Semantic Based Regularization

undirected StarAI approach and (soft) constraints

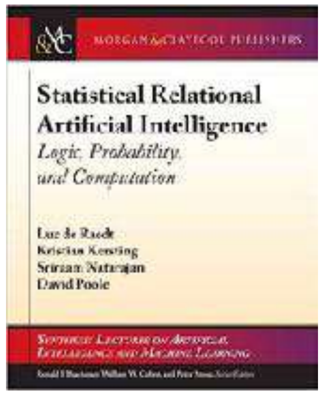


$F := \forall d\ P_A(d) \rightarrow A(d)$
$F_R := \forall d\ \forall d'\ R(d,d') \Rightarrow ((A(d) \wedge A(d')) \vee (\neg A(d) \wedge \neg A(d')))$
$C = \{d_1, d_2\}$
Evidence Predicate Groundings
$P_A(d_1) = 1$
$R(d_1, d_2) = 1$
Output
Output Layer
$\sum$
Quantifier Layers
Φ_F
avg
Φ_{F_R}
avg
Propositional Layer
$t_F(P_A(d_1), f_A(\mathbf{d}_1))$
$t_{F_R}(R(d_1,d_2), f_A(\mathbf{d}_1), f_A(\mathbf{d}_2))$

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

Diligenti et al. AIJ

Two types of Neural Symbolic Systems



Statistical Relational Artificial Intelligence
Logic, Probability, and Computation
Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

Just like in StarAI

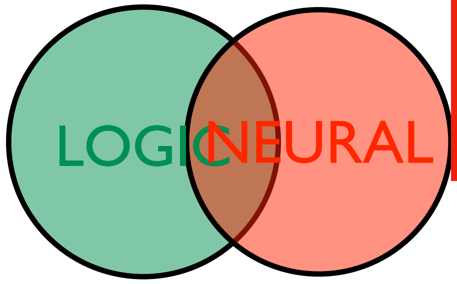
Logic as a kind of ***neural program***

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



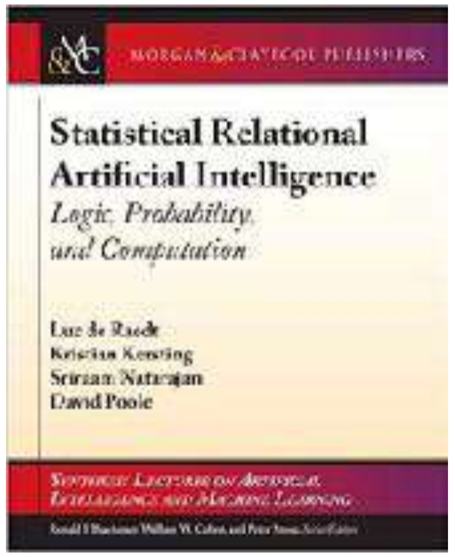
Key Message 1

StarAI and NeSy share similar problems and thus similar solutions apply

What do the numbers mean ?

Three possible choices:
Logic,
Probability &
Fuzzy

Just like in StarAI


MORGAN & CLAYPOOL PUBLISHERS

Statistical Relational Artificial Intelligence

Logic, Probability, and Computation

Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

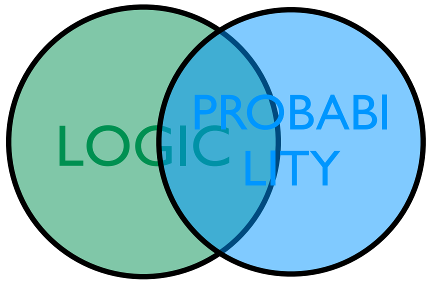
Synthesis Lectures on Artificial Intelligence and Machine Learning

Logic, Probability and Fuzzy

Just like in StarAI

Three types of approaches to NeSy:

- Purely Logic — keep everything logical (e.g., Dai et al, NeurIPS 19)
  - difficult to optimise
- Probabilistic
  - use e.g. arithmetic circuits and knowledge compilation
- Fuzzy
  - easy to translate in neural networks and optimise (but not really logical)



Logic as constraints

Propositional logic

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

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

alarm <- earthquake v burglary

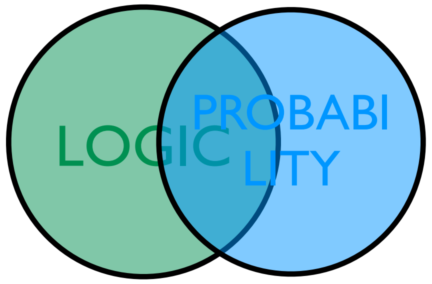
Model / Possible World

0.1	{ burglary,
0.4	hears_alarm(john),
…	alarm,
…	calls(john)}

probability of world ∽ 0.1 x 0.4 x …

SEMANTIC LOSS =
probability that a random possible world satisfies the formula

using weighted model counting (WMC)
weights/probabilities are on the literals



Logic as soft constraints
Markov Logic

Propositional logic

Model / Possible World

10 : f1 <-> calls(mary) <- hears_alarm(mary) ∧ alarm

20 : f2 <-> calls(john) <- hears_alarm(john) ∧ alarm

30 : f3 <-> alarm <- earthquake v burglary

e^10 { f1,
e^20 f2,
e^30 f3,
burglary, hears_alarm(john),
alarm, calls(john),}

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

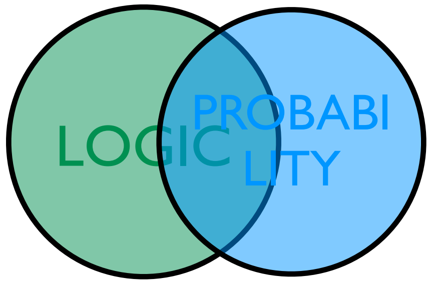
using weighted model counting (WMC)

weights/probabilities are on the formulae (soft constraints)

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

w(f1) = e^10	w(not f1) =e^0 = 1
w(f2) = e^20	w(not f2) =e^0 = 1
w(f3) = e^30	w(not f3) =e^0 = 1

(need to normalise to get probability distribution)



Logic as soft constraints

Probabilistic Soft Logic [Bach & Getoor]

Propositional logic

10 : calls(mary) <- hears_alarm(mary) ∧ alarm

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

30 : alarm <- earthquake v burglary

Model / Possible World

{0.7 burglary,
0.8 hears_alarm(john),
0.5 alarm,
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

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 $B \leq A$

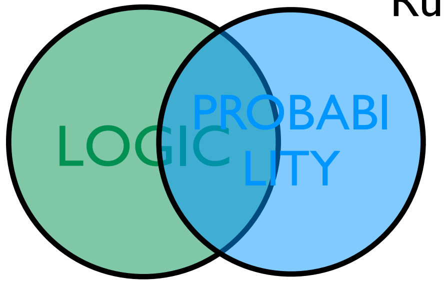
calls(john) <- hears_alarm(john) ∧ alarm

≥ 0.5 0.7 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

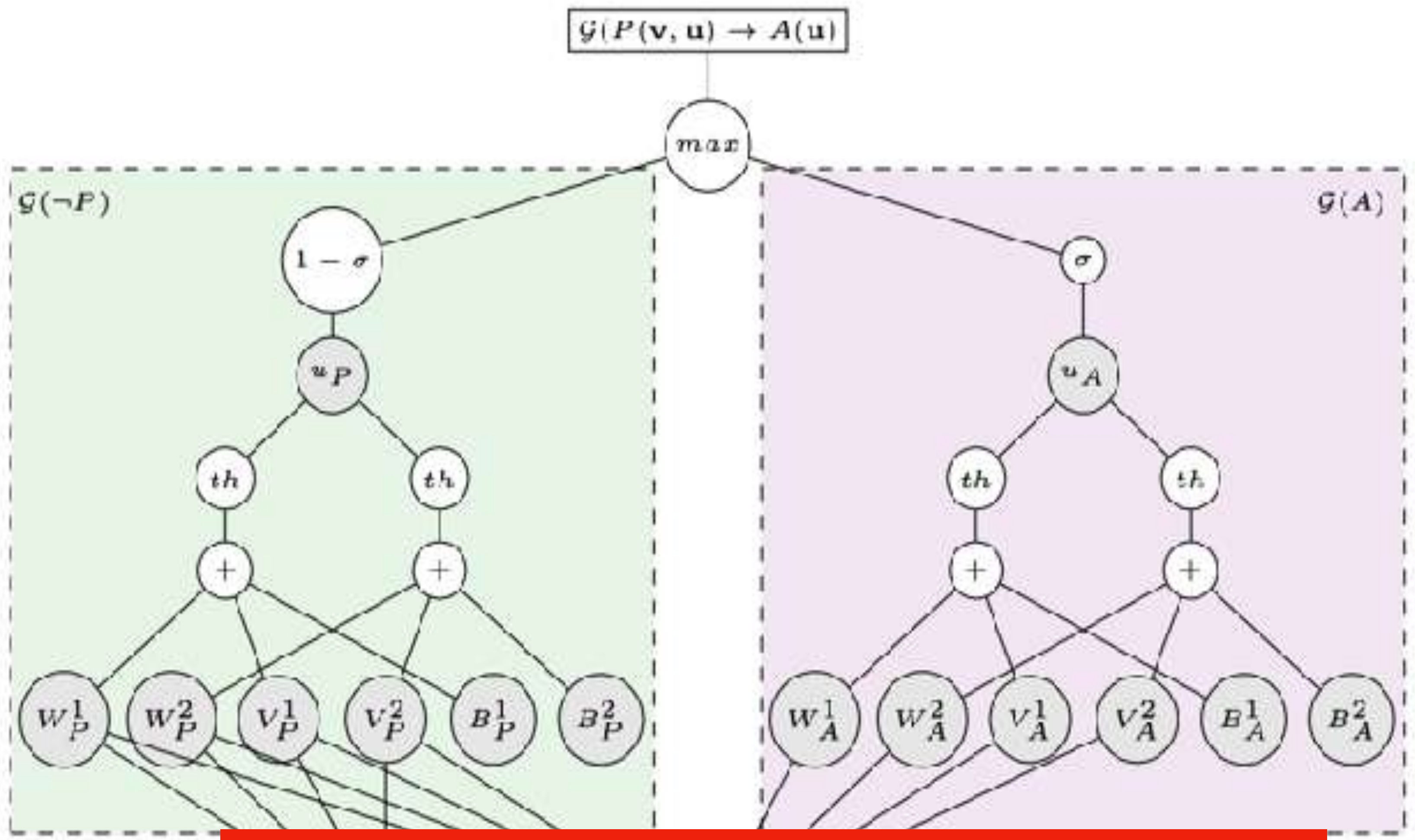
w= e^ [—20 x (1-0.8)]



Logic Tensor Networks

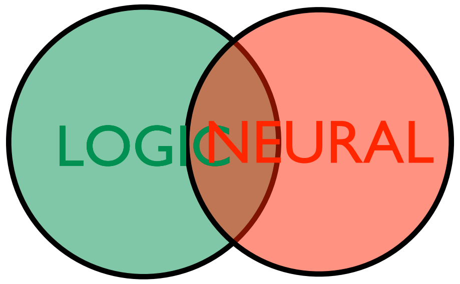
undirected StarAI approach and (soft) constraints

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



$\mathcal{G}(P(\mathbf{v},\mathbf{u}) \rightarrow A(\mathbf{u}))$
max
$\mathcal{G}(\neg P)$
$\mathcal{G}(A)$
$1-\sigma$
σ
u_P
u_A
th
$+$
W_P^1 W_P^2 V_P^1 V_P^2 B_P^1 B_P^2
W_A^1 W_A^2 V_A^1 V_A^2 B_A^1 B_A^2

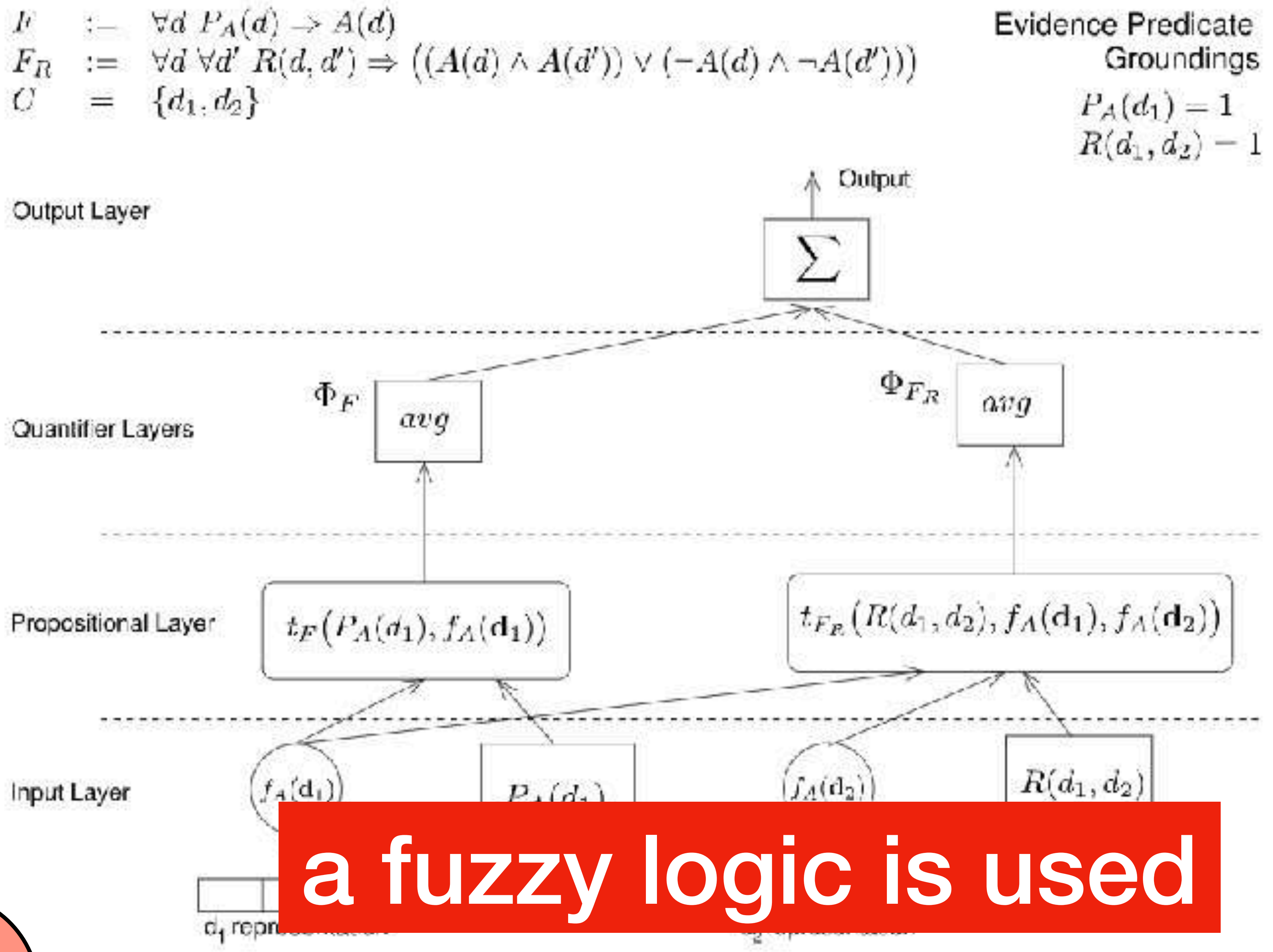
a fuzzy logic is used



Serafini & Garcez

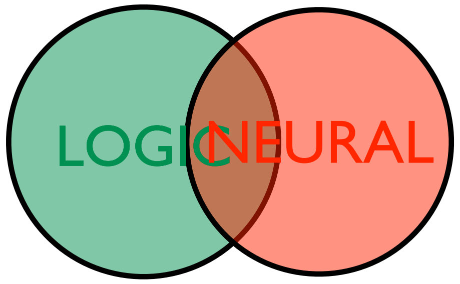
Semantic Based Regularization

undirected StarAI approach and (soft) constraints



$F := \forall d\ P_A(d) \to A(d)$
$F_R := \forall d\ \forall d'\ R(d,d') \Rightarrow ((A(d) \wedge A(d')) \vee (\neg A(d) \wedge \neg A(d')))$
$C = \{d_1, d_2\}$
Evidence Predicate Groundings
$P_A(d_1) = 1$
$R(d_1, d_2) = 1$
Output
Output Layer
Σ
Φ_F
avg
Φ_{F_R}
avg
Quantifier Layers
Propositional Layer
$t_F(P_A(d_1), f_A(\mathbf{d}_1))$
$t_{F_R}(R(d_1,d_2), f_A(\mathbf{d}_1), f_A(\mathbf{d}_2))$
Input Layer
$f_A(\mathbf{d}_1)$
$P_A(d_1)$
$f_A(\mathbf{d}_2)$
$R(d_1, d_2)$
d_1 repr

a fuzzy logic is used

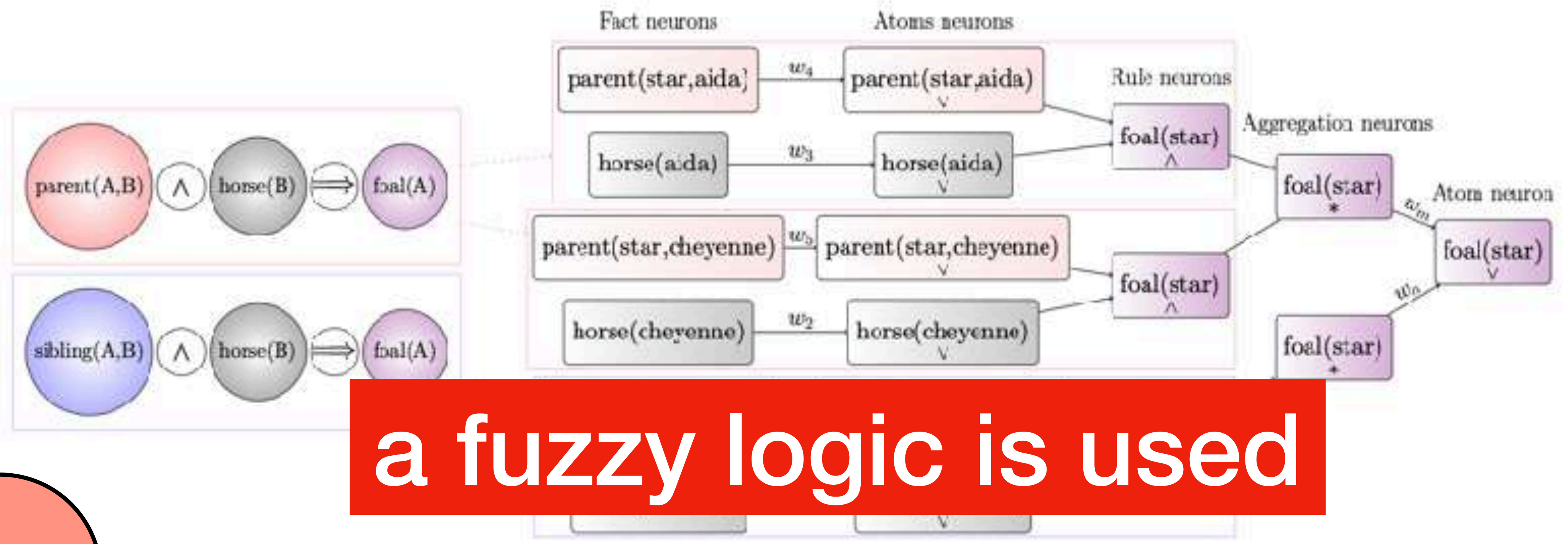


Diligenti et al. AIJ

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)



a fuzzy logic is used

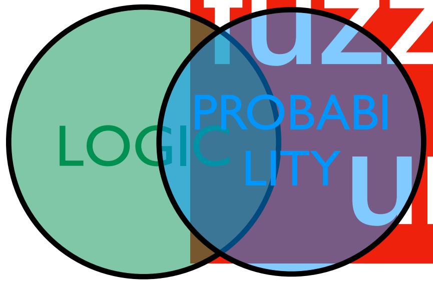
[Sourek, Kuzelka, et al JAIR]

Logic, Probability and Fuzzy

Three types of approaches to NeSy:

- Purely Logic - keep everything logical (e.g., Dai et al, NeurIPS)
  - difficult to optimise
- Probabilistic with e.g. arithmetic circuits and knowledge compilation
  - knowledge compilation (hard to compile, fast inference and learning afterwards)
- Fuzzy

Consequence :
faster / convex optimisation
fuzzy logic differs from traditional logic
unexpected behaviours can occur



Key Message 2

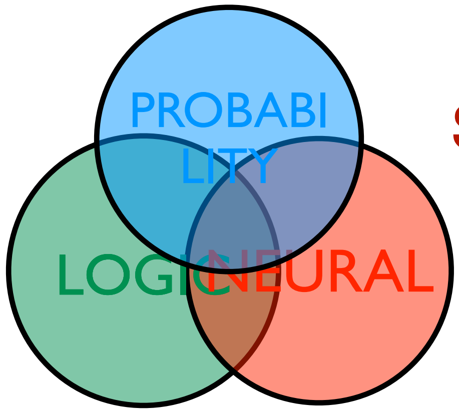
A different approach

A true integration T of X and Y should allow to reconstruct X and Y as special cases of T

Thus, Neural Symbolic approaches should have both logic and neural networks as special cases

Our approach: “an interface layer (<> pipeline) between neural & symbolic components” will be illustrated with DeepProbLog

See also [Manhaeve et al., NeurIPS 18; arXiv: 1907.08194]



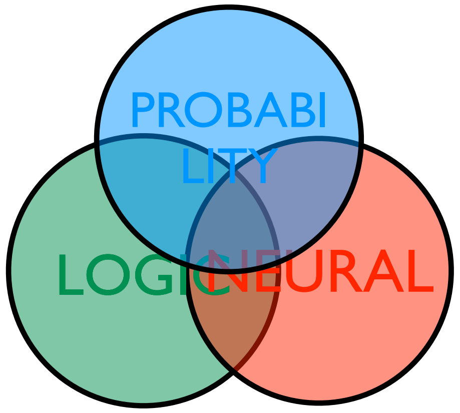
Part 2 of the talk — illustration with DeepProbLog [NeurIPS 2018]

DeepProbLog

DeepProbLog = Probability + Logic + Neural Network

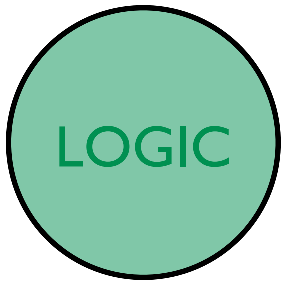
DeepProbLog = ProbLog + Neural Network

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 network
Fuzzy logic	Probabilistic logic programming
Language semantics unclear	Clear semantics

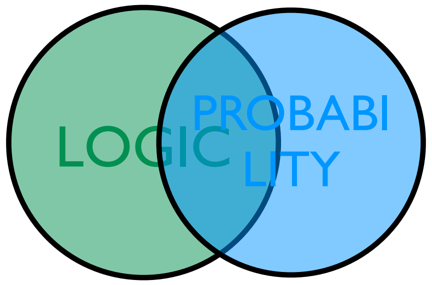


PART 2

FROM

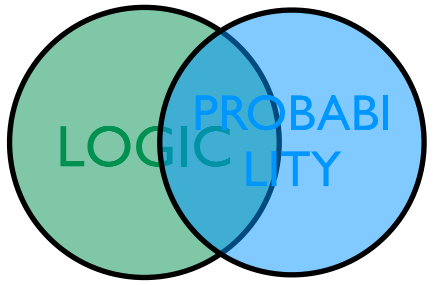


TO

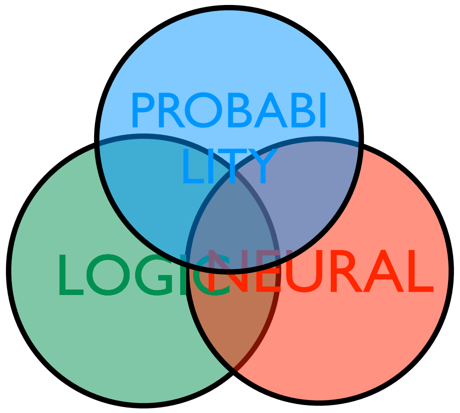


ProbLog

FROM



TO



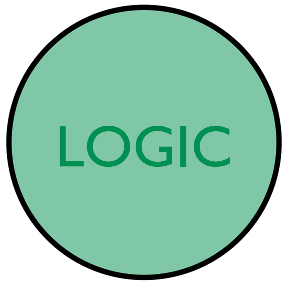
DeepProbLog

a logic programming perspective

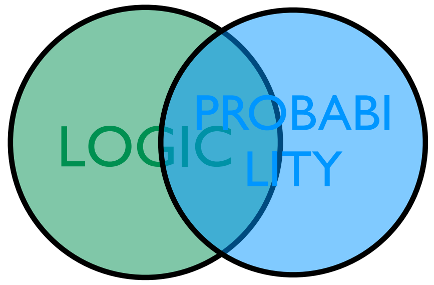
PART 2 A

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)

$$\begin{tikzcd}
 & \text{:- calls(mary).} \arrow[d] & \\
 & \text{:- alarm, hears\_alarm(mary).} \arrow[dl] \arrow[dr] & \\
\text{:- earthquake, hears\_alarm(mary).} \arrow[d] & & \text{:- burglary, hears\_alarm(} \arrow[d] \\
\text{:- hears\_alarm(mary).} \arrow[d] & & \text{:- hears\_alarm(mary)} \arrow[d] \\
\text{[]} & & \text{[]}
\end{tikzcd}$$

A proof-theoretic view

LOGIC

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).
```

Probabilistic facts

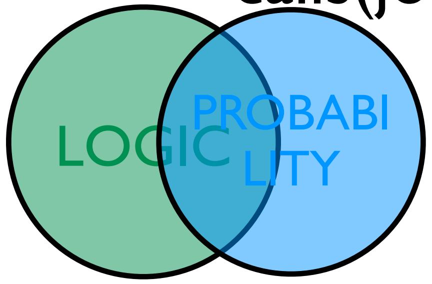
```
alarm :– earthquake.
alarm :– burglary.

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



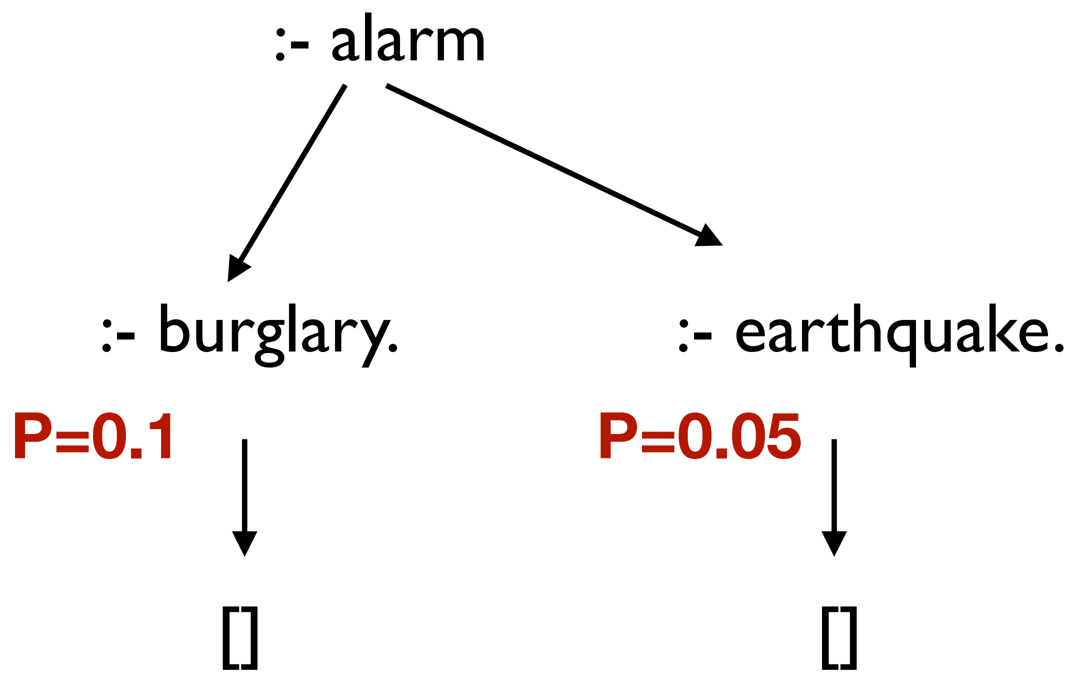
Probabilistic Logic Programs

as in the probabilistic programming language ProbLog

Propositional logic program

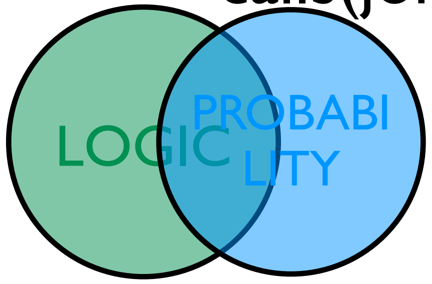
```
0.1 :: burglary.
0.3 ::hears_alarm(mary).


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)



$$\begin{tikzcd}
 & \text{:- alarm} \arrow[dl] \arrow[dr] & \\
\text{:- burglary.} \arrow[d, "P=0.1"'] & & \text{:- earthquake.} \arrow[d, "P=0.05"'] \\
{[]} & & {[]}
\end{tikzcd}$$

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



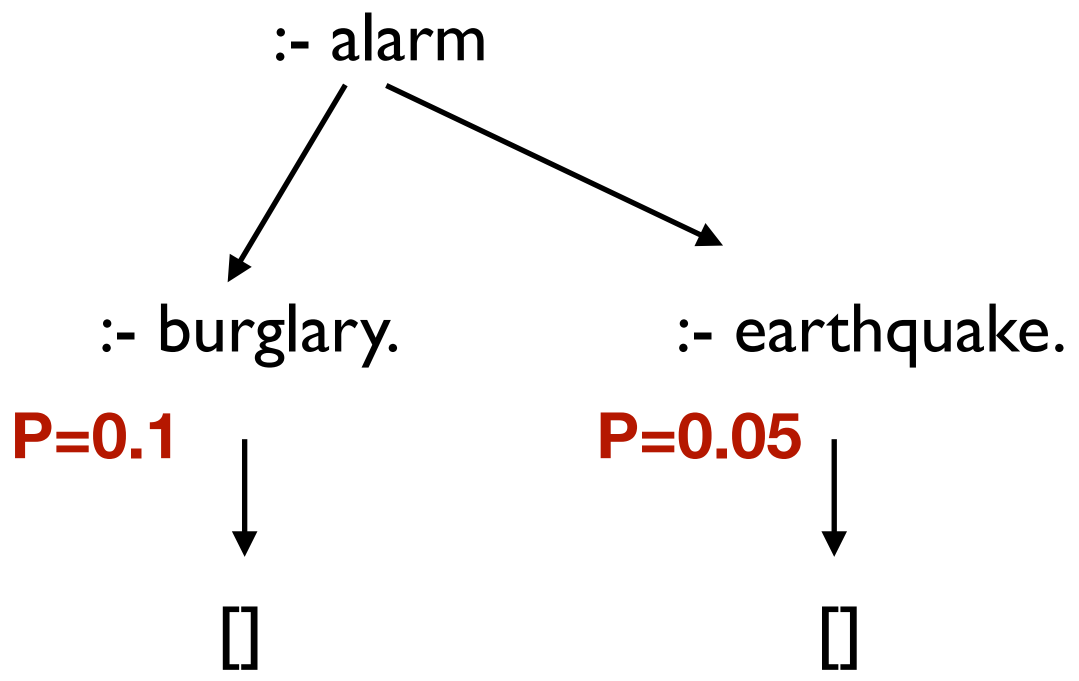
Probabilistic Logic Programs

as in the probabilistic programming language ProbLog

Propositional logic program

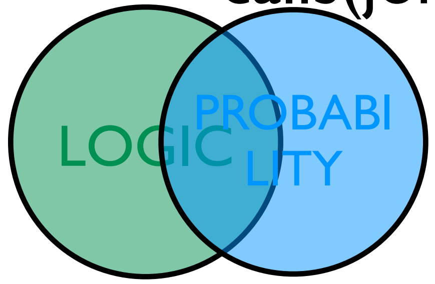
```
0.1 :: burglary.
0.3 ::hears_alarm(mary).


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)

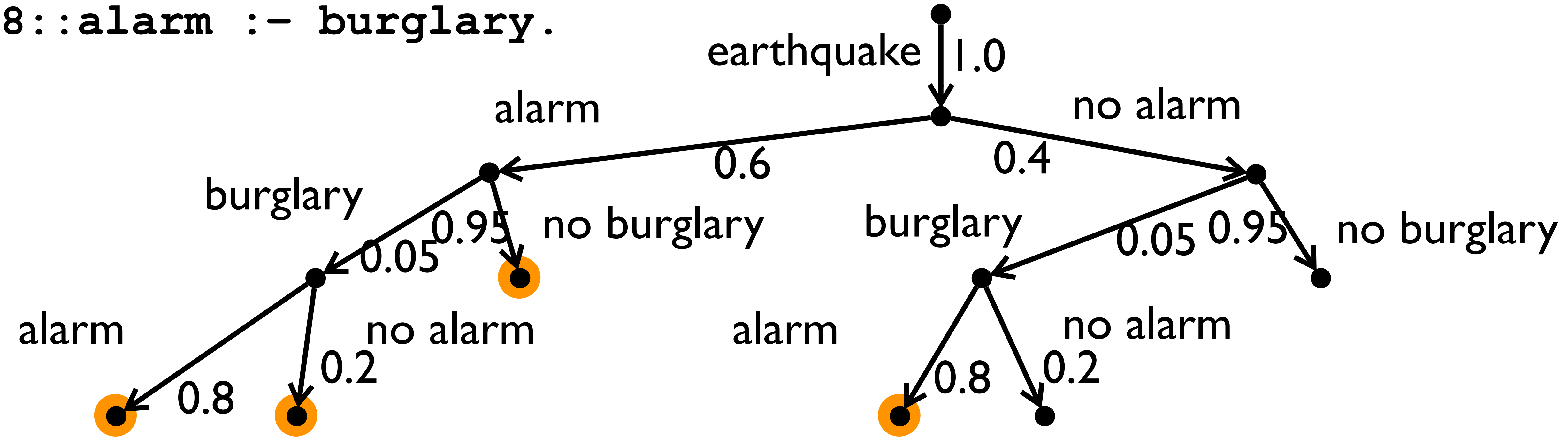


Probabilistic Logic Program Semantics

[Vennekens et al, ICLP 04]

```
earthquake.
0.05::burglary.

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

probabilistic causal laws



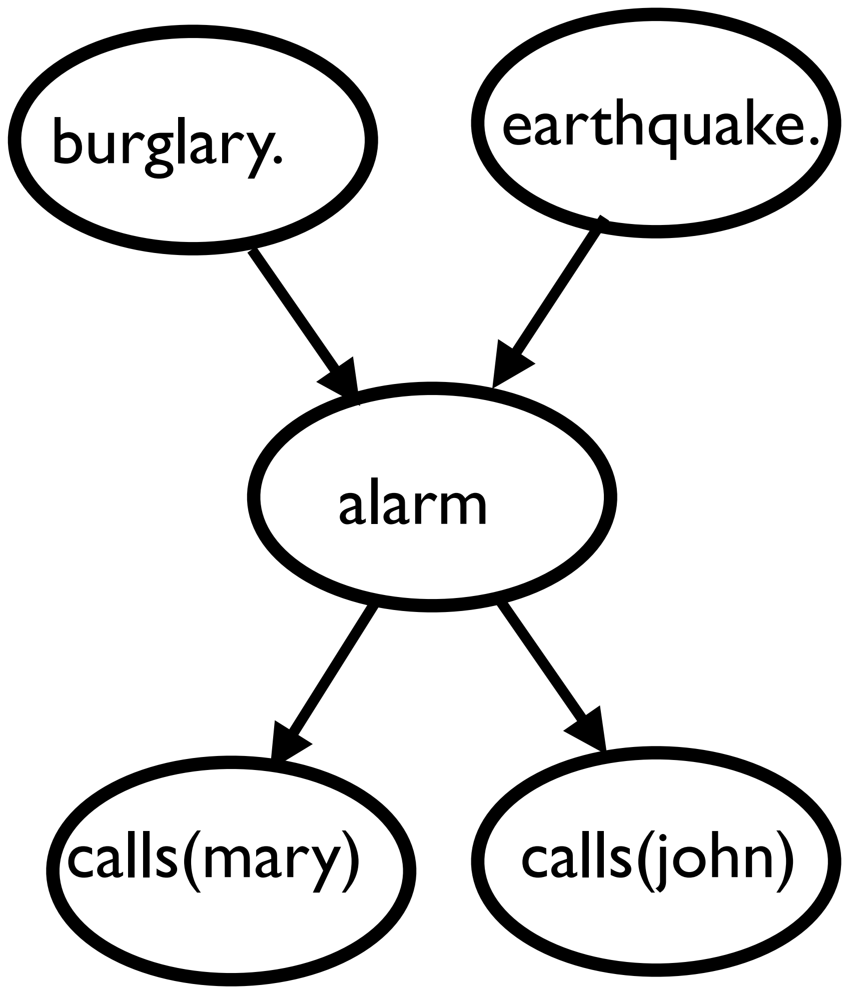
P(alarm)=0.6×0.05×0.8+0.6×0.05×0.2+0.6×0.95+0.4×0.05×0.8

Probabilistic Logic Program Semantics

Propositional logic program

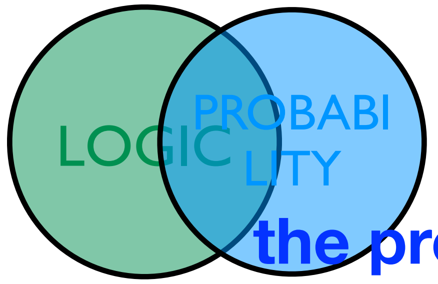
```
0.1 :: burglary.
0.05 :: earthquake.

alarm :– earthquake.
alarm :– burglary.

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

Bayesian Network

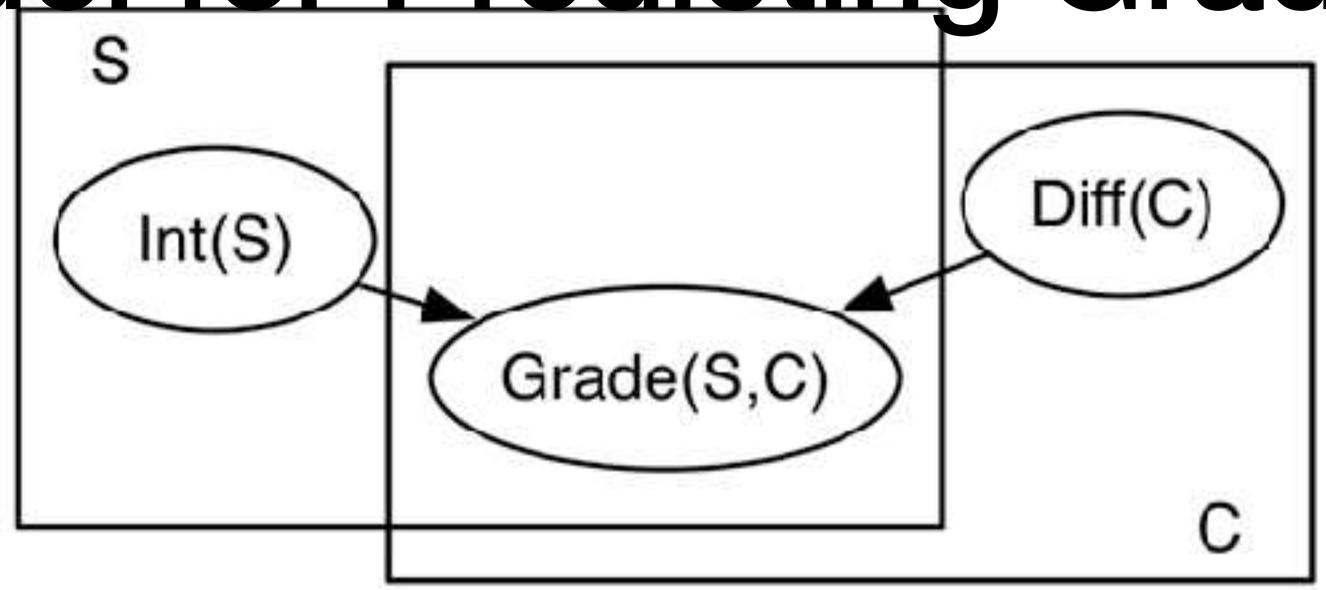


Bayesian net encoded as Probabilistic Logic Program
PLPs correspond to directed graphical models

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



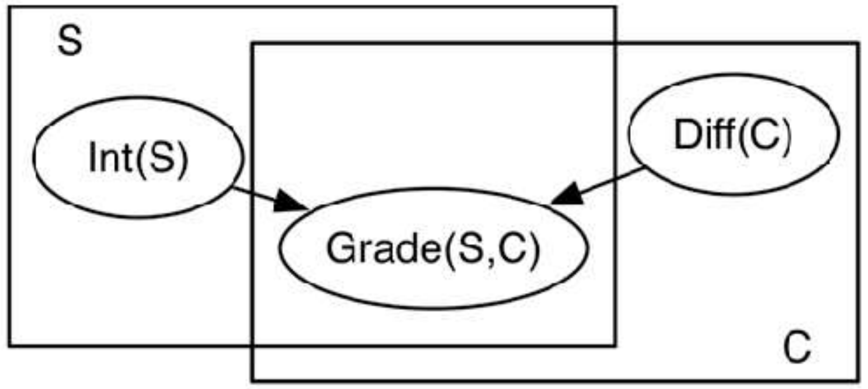
"Program" Abstraction:

- 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

Probabilistic Logic Programs



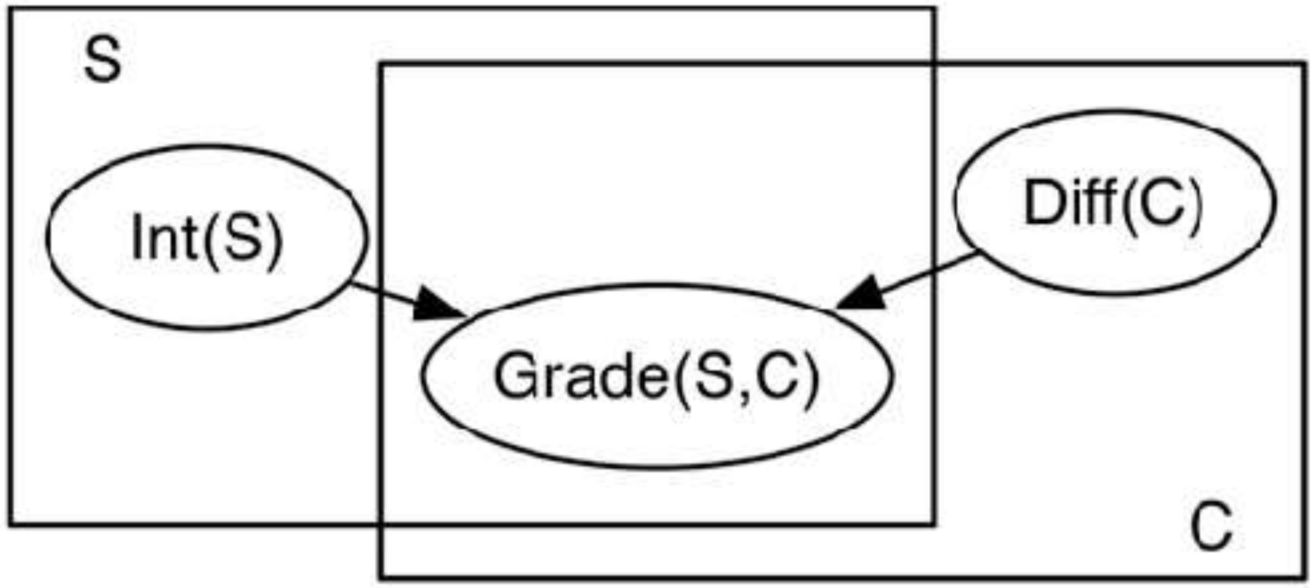
```
0.4 :: int(S) :- student(S).
0.5 :: diff(C):- course(C).

student(john).  student(anna).  student(bob).
course(ai).     course(ml).     course(cs).

gr(S,C,a) :- int(S), not diff(C).
0.3::gr(S,C,a); 0.5::gr(S,C,b);0.2::gr(S,C,c) :-   int(S), diff(C).
0.1::gr(S,C,b); 0.2::gr(S,C,c); 0.2::gr(S,C,f) :-
          student(S), course(C),
          not int(S), not diff(C).
0.3::gr(S,C,c); 0.2::gr(S,C,f) :-
          not int(S), diff(C).
```

ProbLog by example: Grading

```
unsatisfactory(S) :- student(S), grade(S,C,f).

excellent(S):- student(S), not(grade(S,C1,G),below(G,a)), grade(S,C2,a).
```

```
0.4 :: int(S) :- student(S).
0.5 :: diff(C):- course(C).

student(john).  student(anna).  student(bob).
course(ai).       course(ml).       course(cs).

gr(S,C,a) :- int(S), not diff(C).
0.3::gr(S,C,a); 0.5::gr(S,C,b);0.2::gr(S,C,c) :-   int(S), diff(C).
0.1::gr(S,C,b); 0.2::gr(S,C,c); 0.2::gr(S,C,f) :-
          student(S), course(C),
           not int(S), not diff(C).
0.3::gr(S,C,c); 0.2::gr(S,C,f) :-  not int(S), diff(C).
```

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

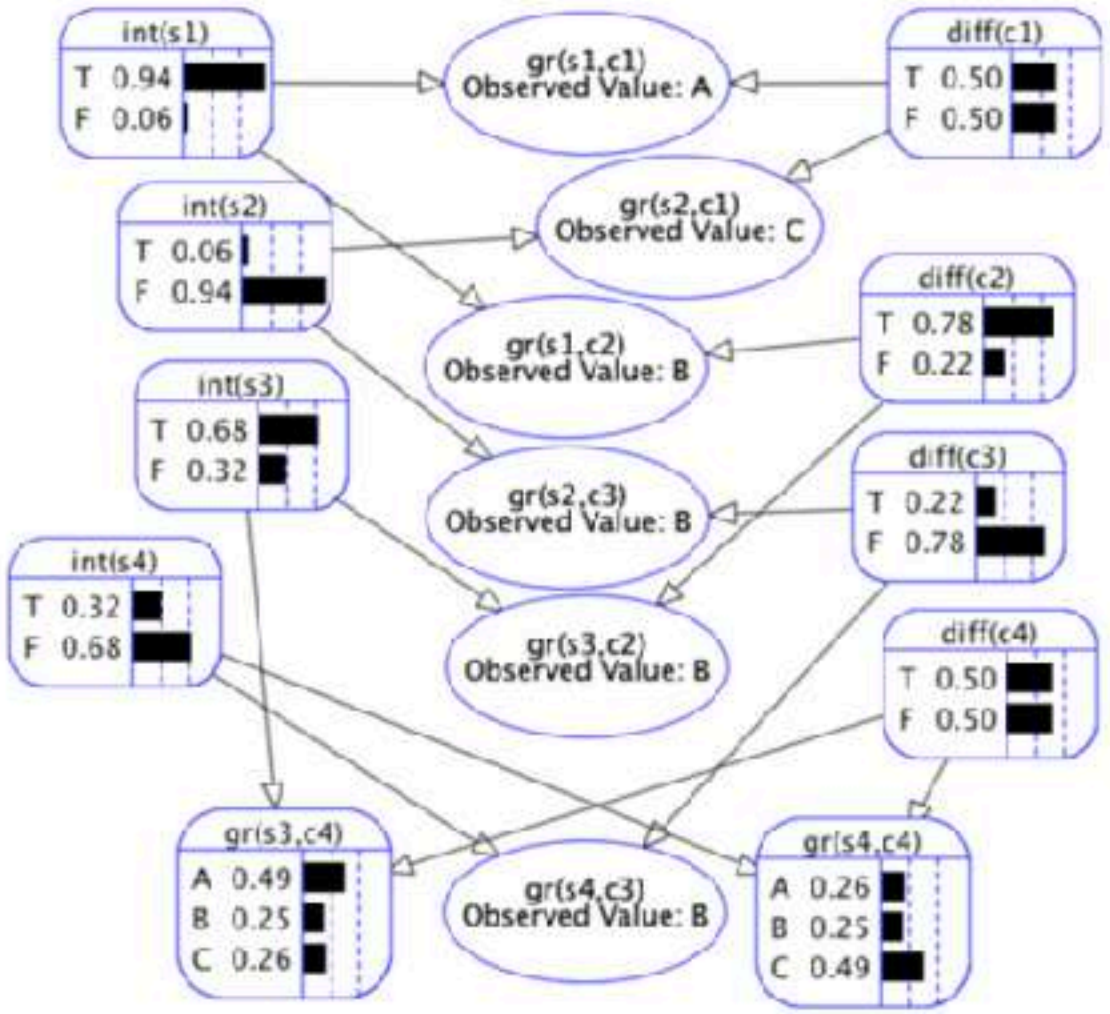
build and learn compact models,

from one set of individuals - > other sets;

reason also about exchangeability,

build even more complex models,

incorporate background knowledge



ProbLog Inference

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)

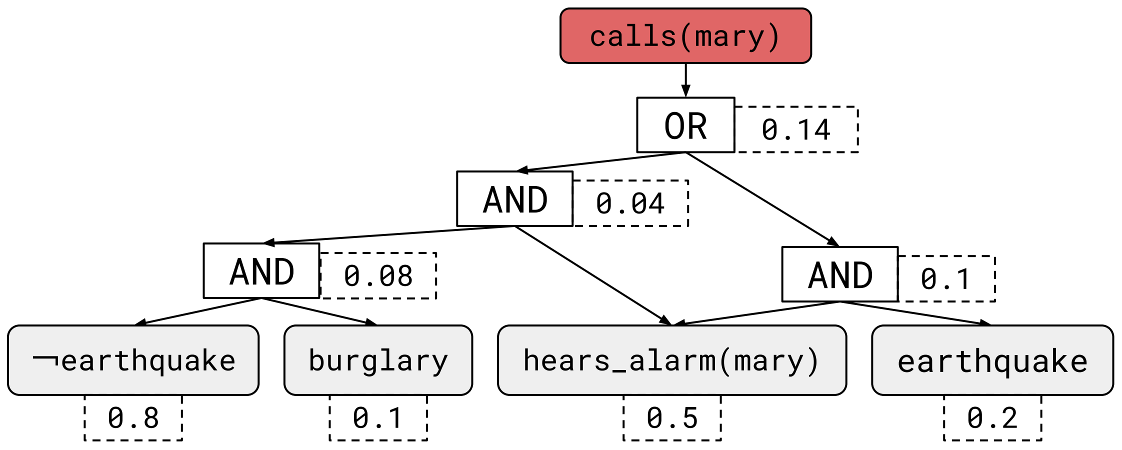
$\leftrightarrow$

hears_alarm(mary) ∧ (burglary ∨ earthquake)

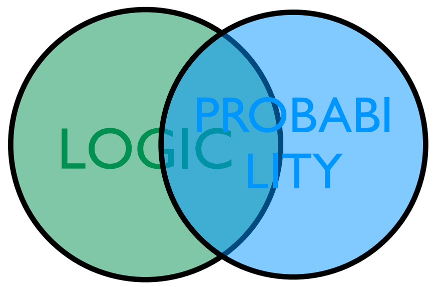
ProbLog Inference

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



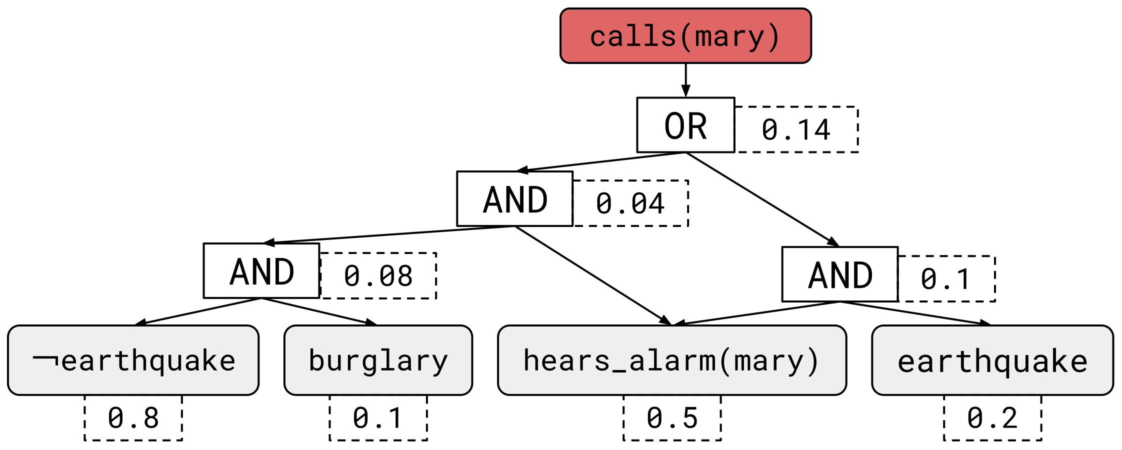
$$\text{calls(mary)} \leftrightarrow \text{hears\_alarm(mary)} \wedge (\text{burglary} \vee \text{earthquake})$$



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 - replace AND by X and OR by +**

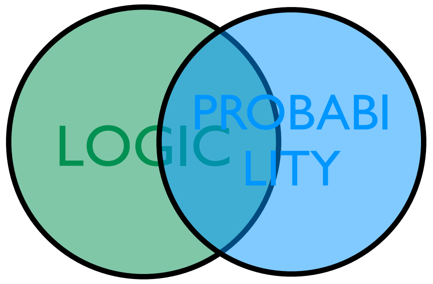


calls(mary)

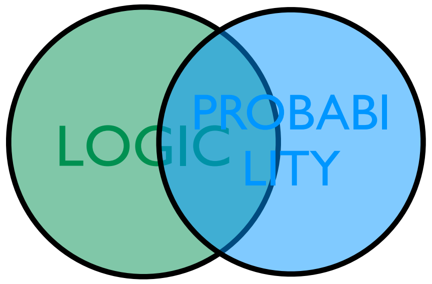
$\leftrightarrow$

hears_alarm(mary) ∧ (burglary ∨ earthquake)

The AC deals with the disjoint sum problem



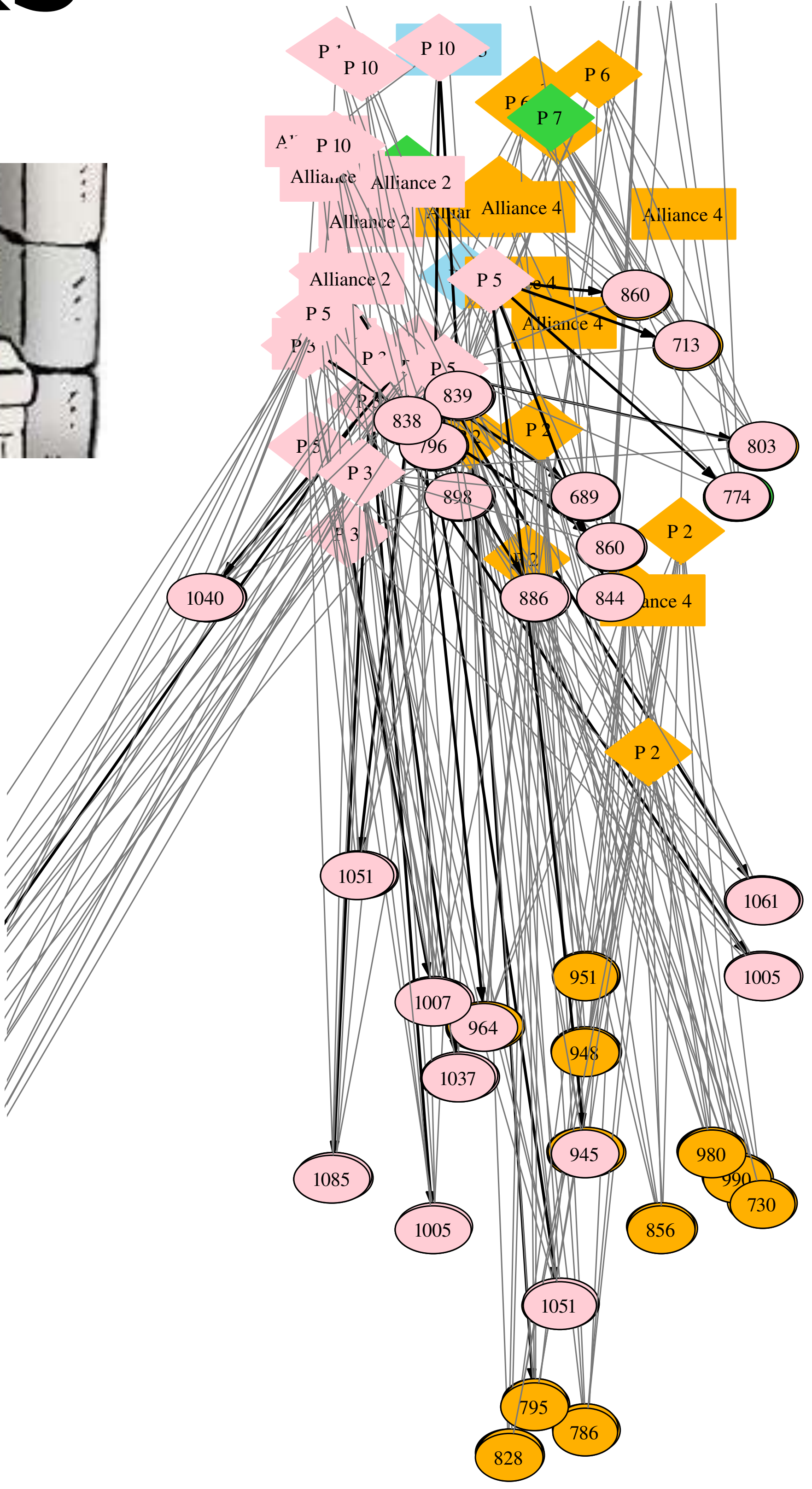
ProbLog applications

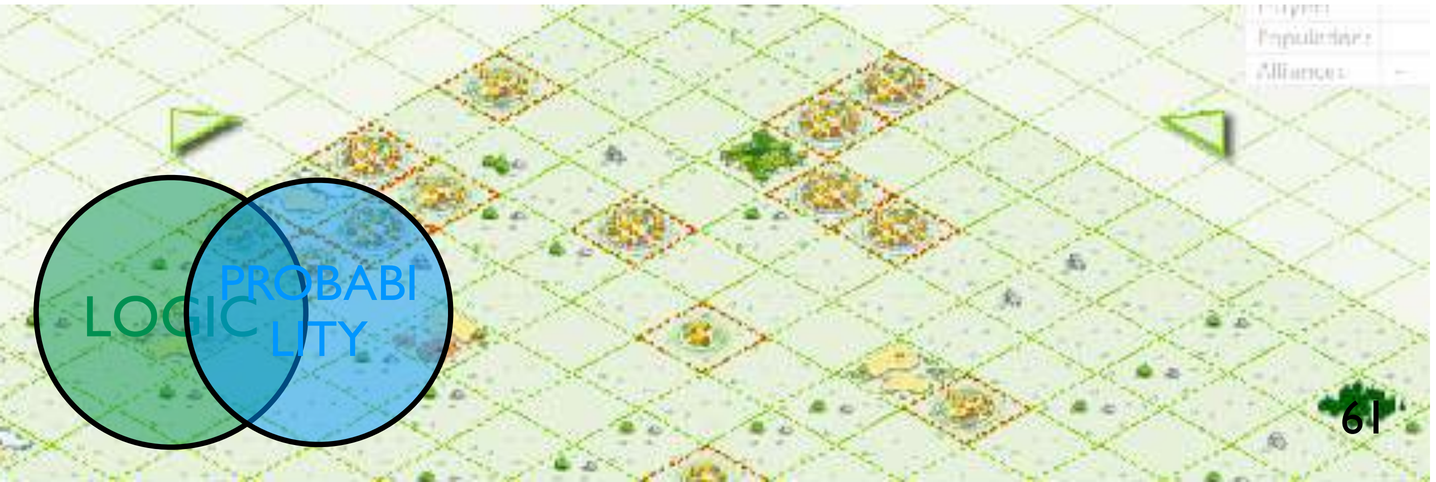


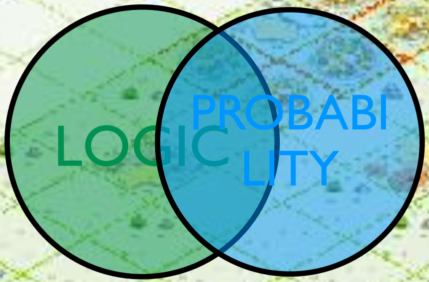
Dynamic networks

Travian: A massively multiplayer real-time strategy game

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

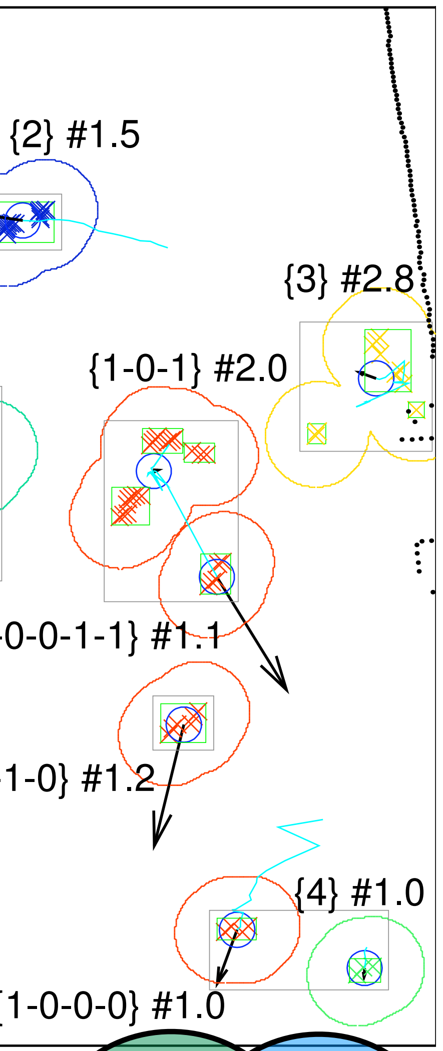






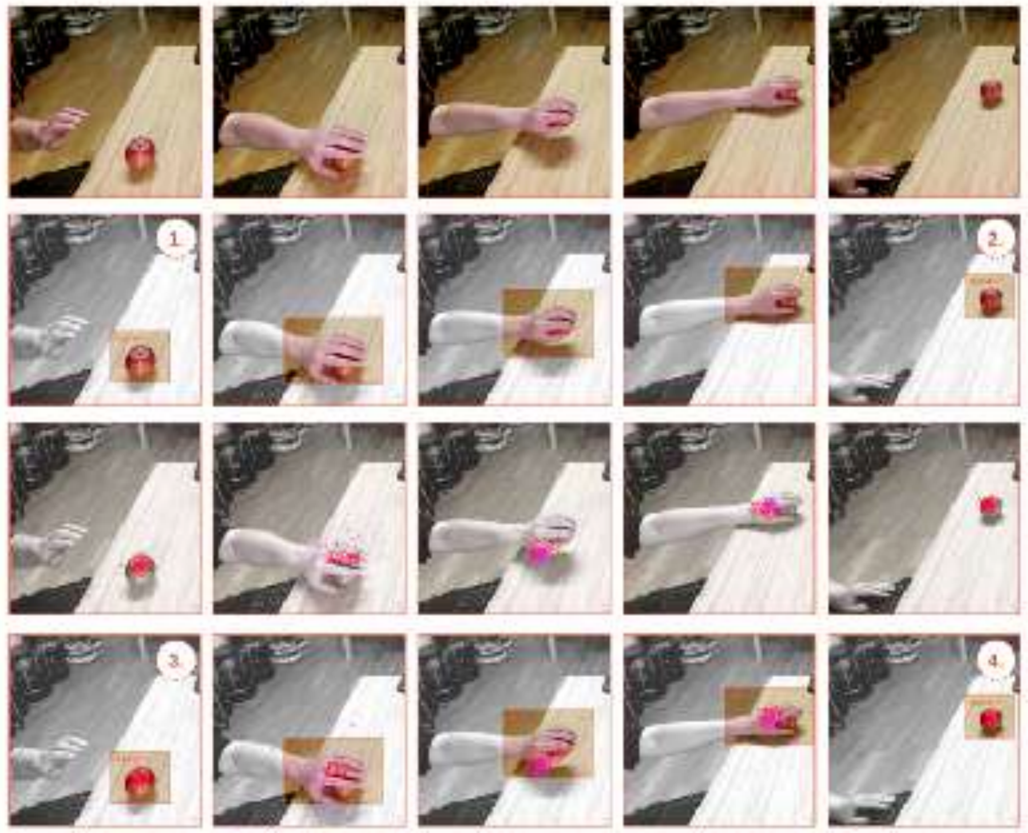
[Thon et al, MLJ 11]

Activity analysis and tracking

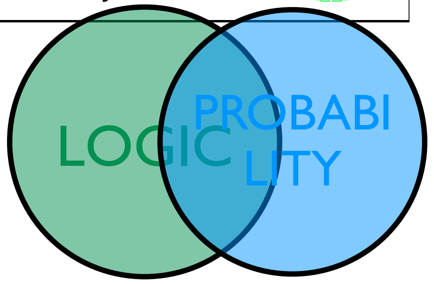


- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?

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

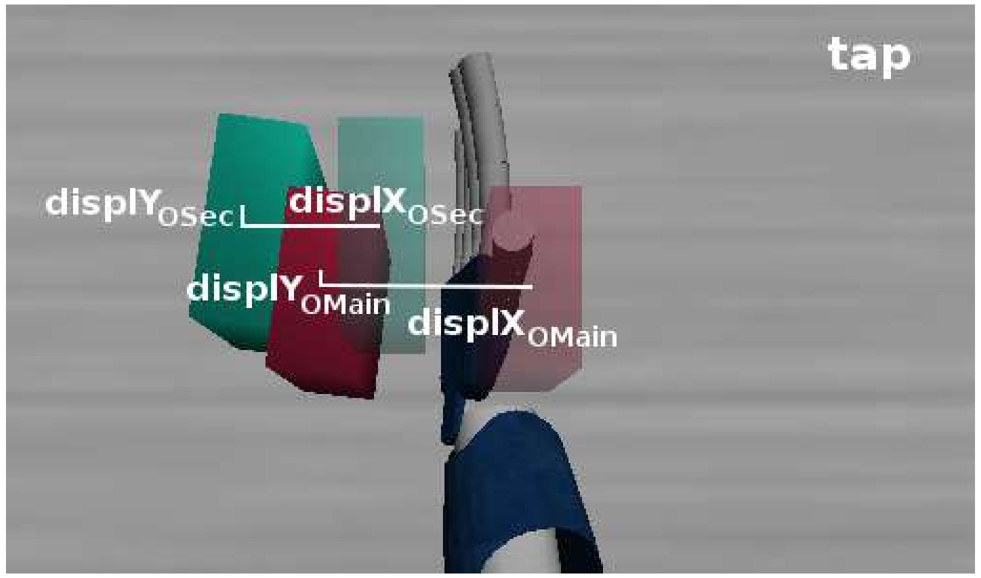


[Persson et al, IEEE Trans on Cogn. & Dev. Sys. 19; IJCAI 20]

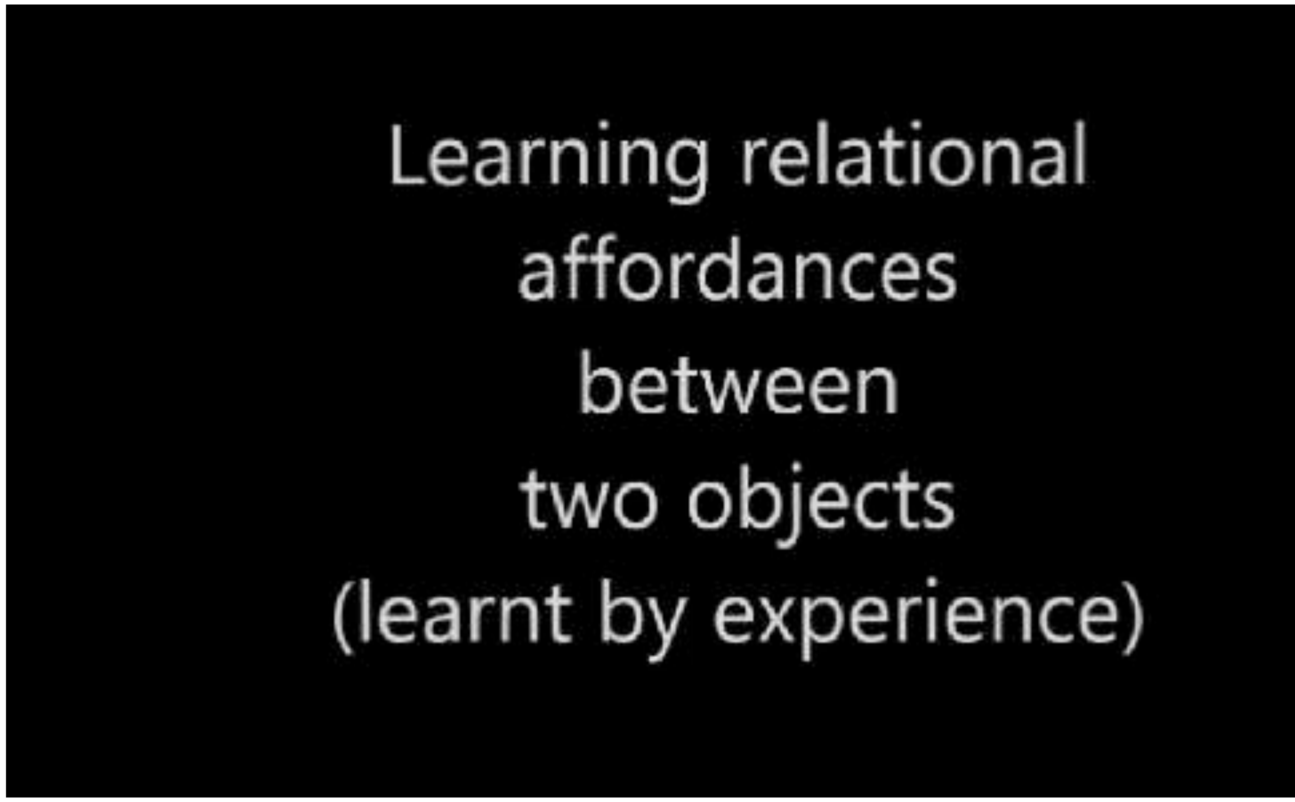


Learning relational affordances

Learn probabilistic model



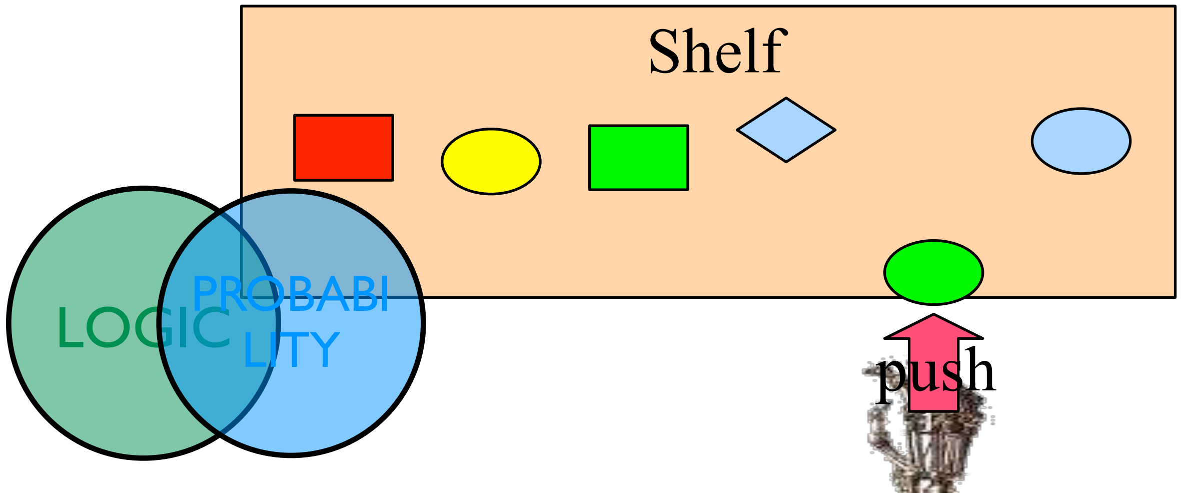
similar to probabilistic Strips (with continuous distributions)

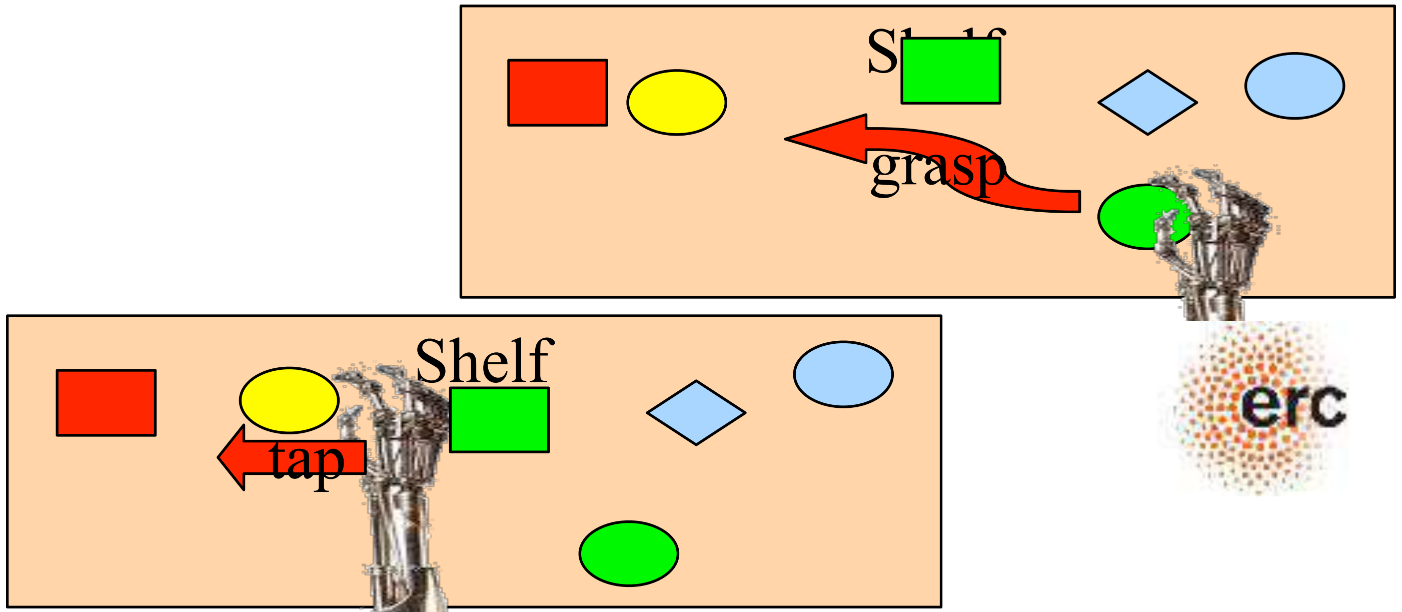

Learning relational affordances between two objects (learnt by experience)

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

From two object interactions

Generalize to N





Biology

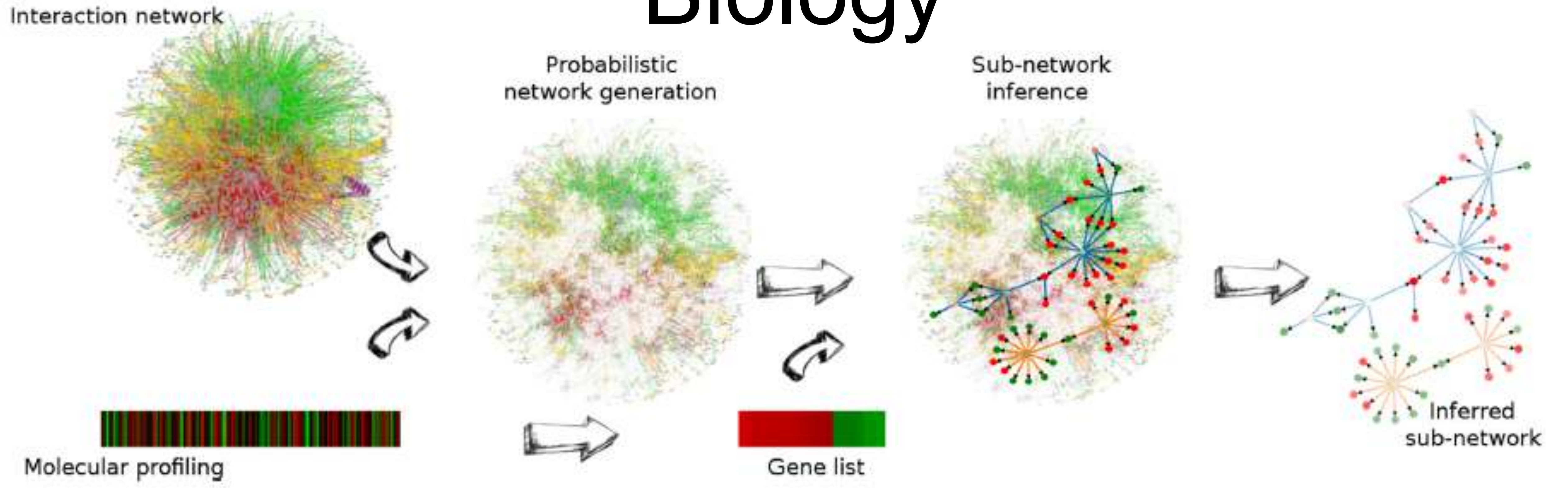
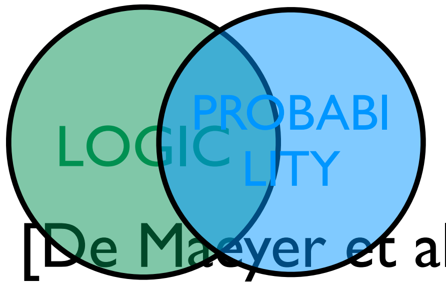


Figure 1. Overview of PheNetic, a web service for network-based interpretation of 'omics' data. The web service uses as input a genome wide interaction network for the organism of interest, a user generated molecular profiling data set and a gene list derived from these data. Interaction networks for 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.

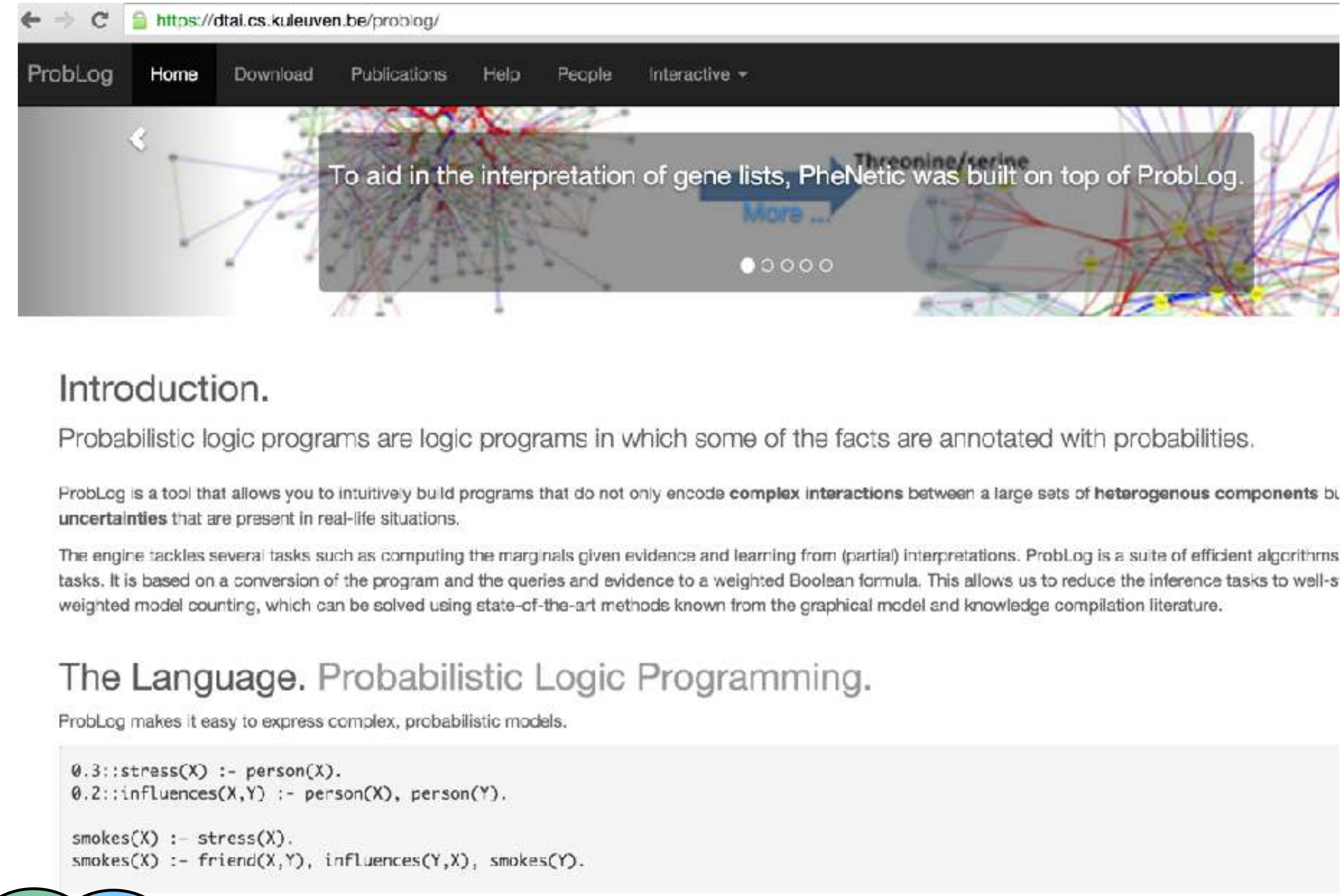
- Causes: Mutations
  - All related to similar phenotype
- Effects: Differentially expressed genes
- 27 000 cause effect pairs

- 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]


https://dtai.cs.kuleuven.be/problog/
ProbLog
Home
Download
Publications
Help
People
Interactive
To aid in the interpretation of gene lists, PheNetic was built on top of ProbLog.
More ...

Introduction.

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

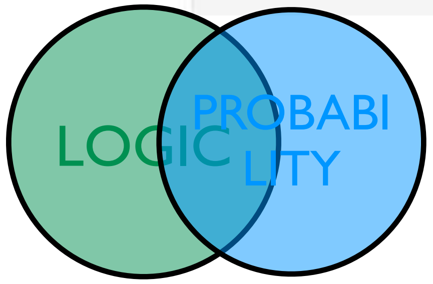
ProbLog is a tool that allows you to intuitively build programs that do not only encode **complex interactions** between a large sets of **heterogenous components** bu
uncertainties that are present in real-life situations.

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.

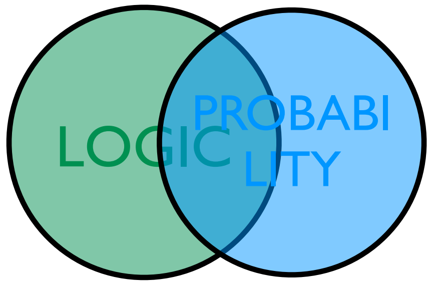
```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).

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



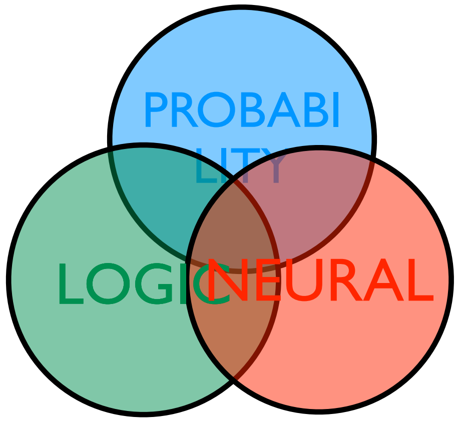
PART 2 B

From ProbLog to DeepProbLog

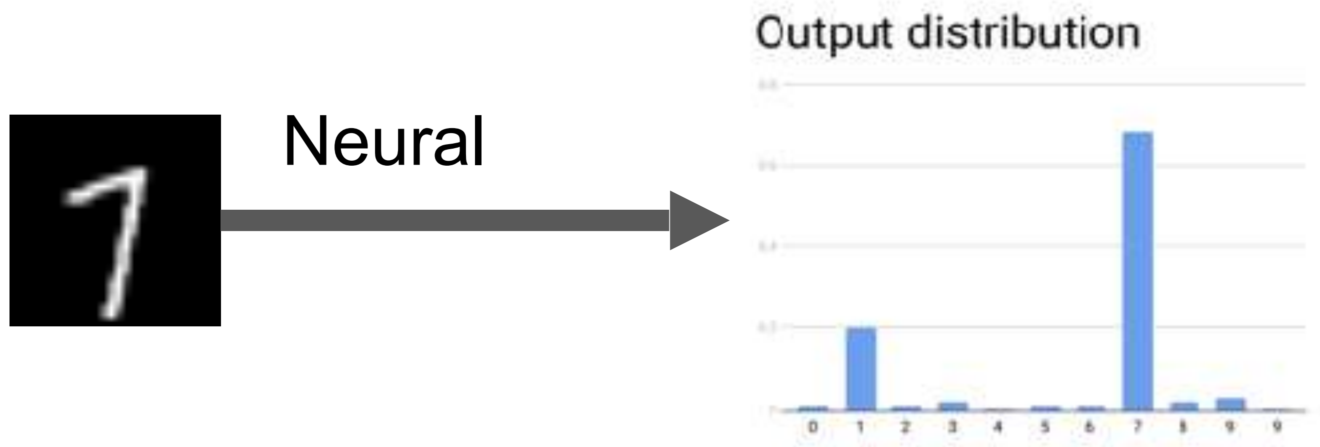
FROM



TO



Neural predicate



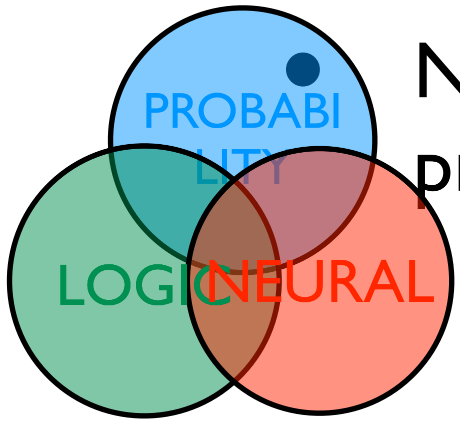
- Neural networks have uncertainty in their predictions
- A normalized output can be interpreted as a probability distribution
- Neural predicate models the output as probabilistic facts
- No changes needed in the probabilistic host language

Key Idea DeepProbLog

unify the basic concepts in logic and neural networks:

neural predicate ~ neural net

an interface between logic and neural nets



The neural predicate

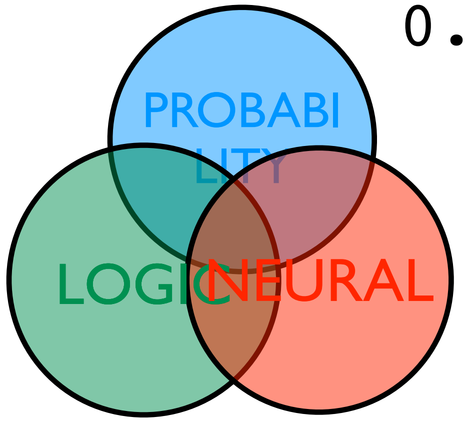
The output of the neural network is probabilistic facts in DeepProbLog

Example:

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

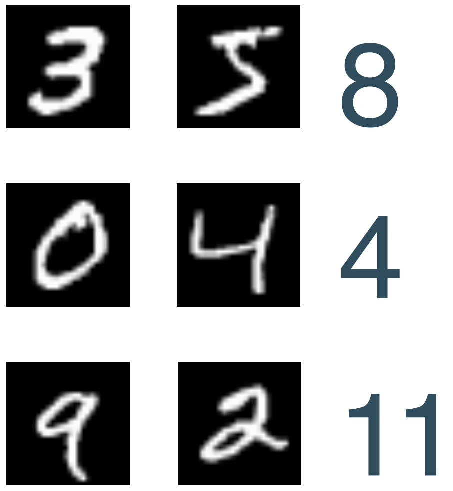
Instantiated into a (neural) Annotated Disjunction:

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



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

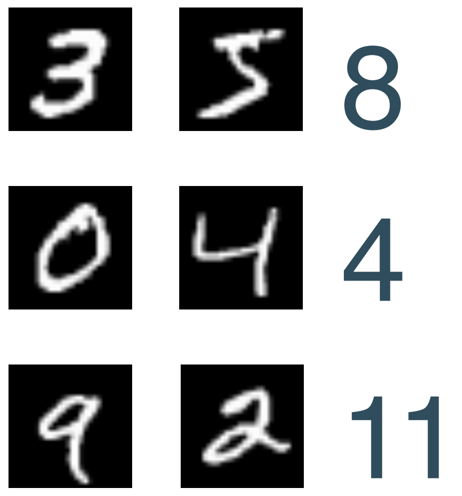
```
nn(mnist_net, [X], Y, [0 ... 9] ) :: digit(X,Y).

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.
```

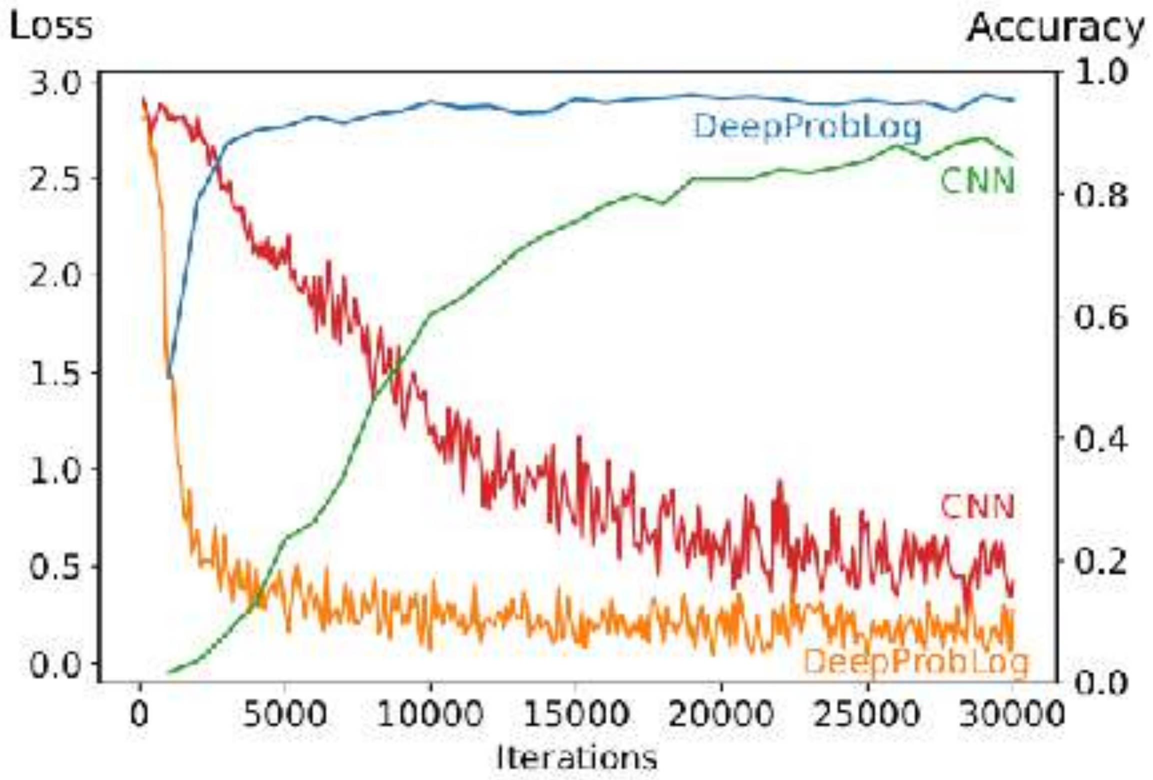
addition(, ,8) :- digit(,N1), digit(,N2), 8 is N1 + N2.

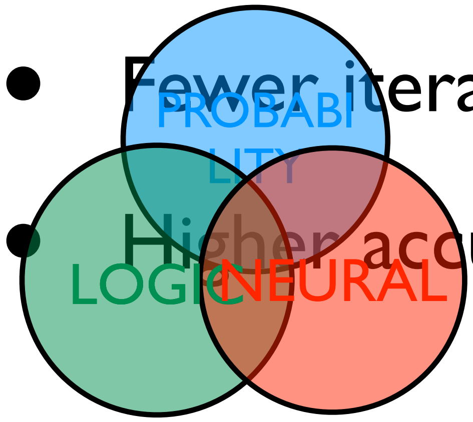
Examples:

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





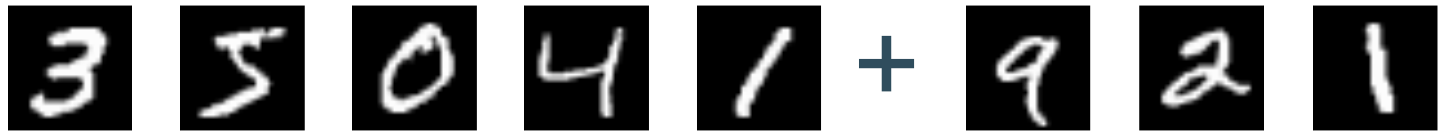
Example

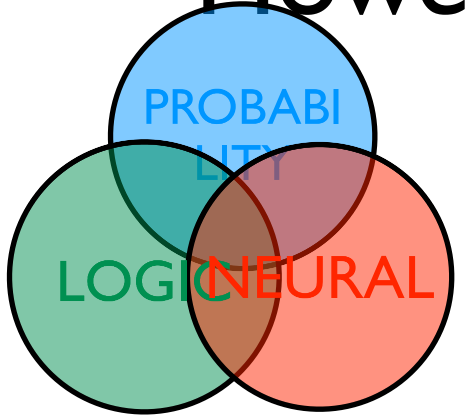
Learn to classify the sum of pairs of MNIST digits

Individual digits are not labeled!

E.g. (, , 8)

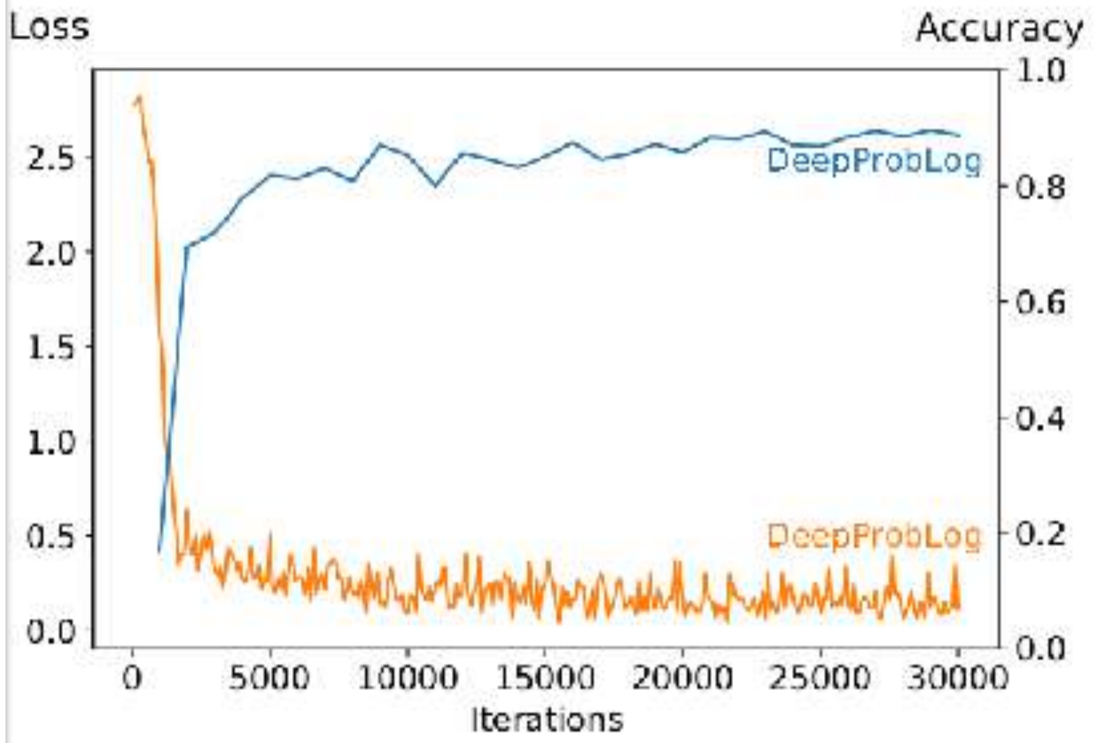
Could be done by a CNN: classify the concatenation of both images into 19 classes

However: 

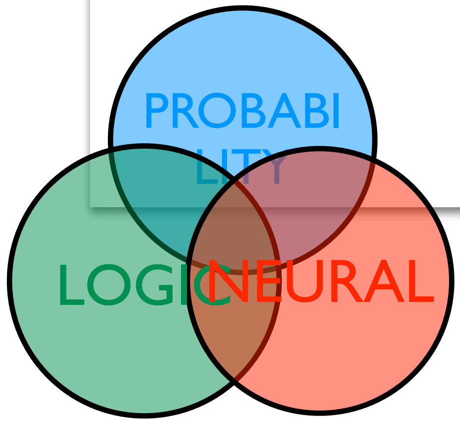


Multi-digit MNIST addition with MNIST

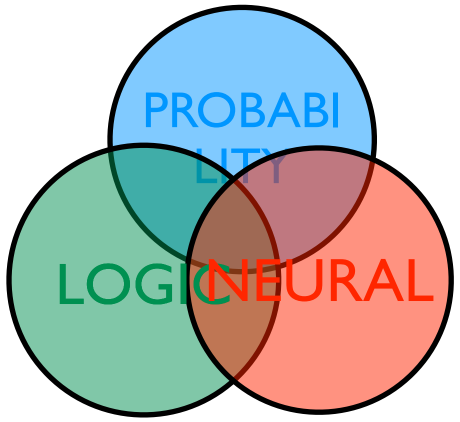
```
number ( [ ] , Result , Result ) .
number ( [H | T ] , Acc , Result) :–
  digit(H, Nr ), Acc2 is Nr +10*Acc ,
  number ( T , Acc2 , Result ) .
number (X,Y) :– number (X, 0 ,Y ) .

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



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



Inference & Learning

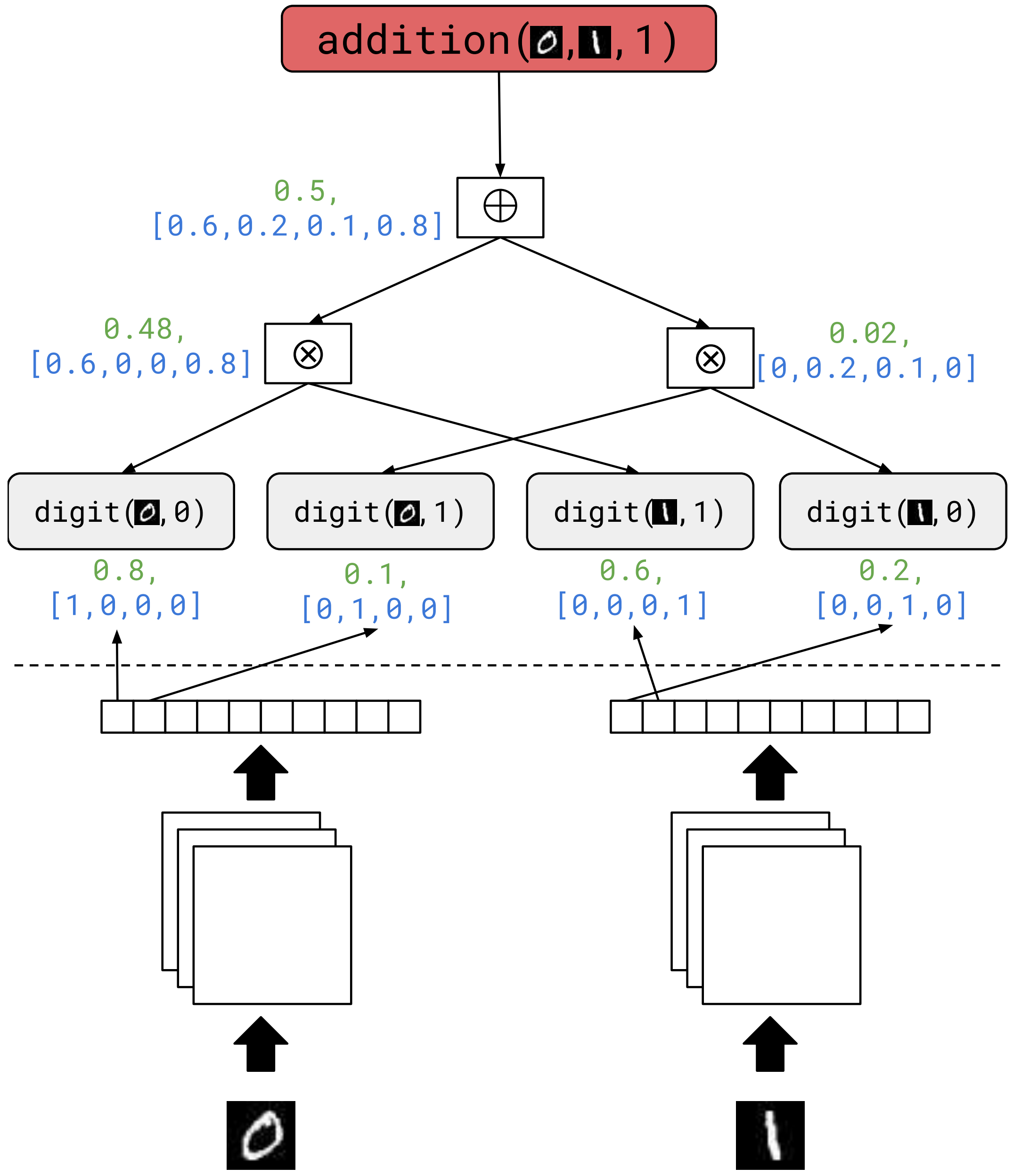


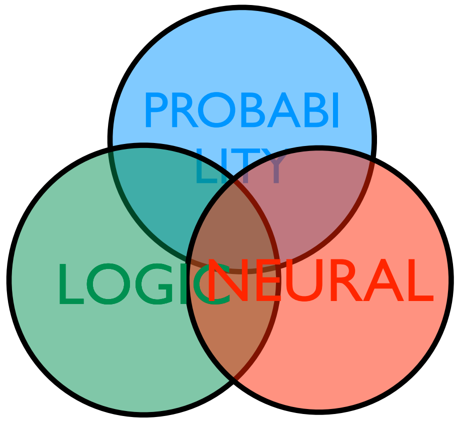
Gradient Semiring

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

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

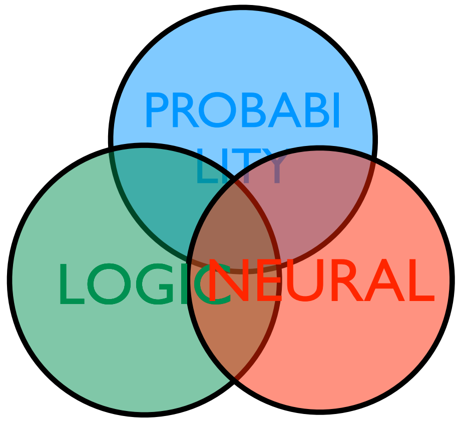
The ACs are differentiable
and there is an interface
with the neural nets

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





Experiments



Program Induction/Sketching

In Neural Symbolic methods

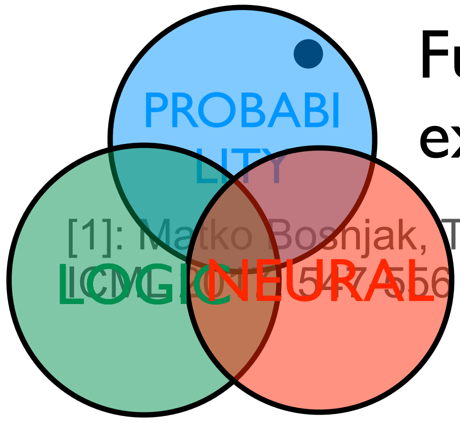
- Rule Induction — work with templates

  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

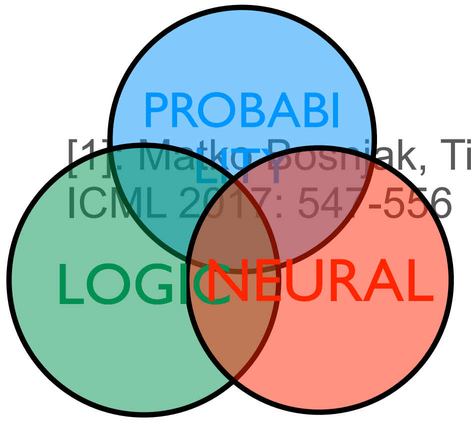


[1]: Matko Bosnjak, Tim Rocktäschel, Jason Naradowsky, Sebastian Riedel: Programming with a Differentiable Forth Interpreter. ICML 2017: 547-556

Tasks[1]

- Sorting
  - Sort lists of numbers using Bubble sort
  - Hole: Swap or don’t swap when comparing two numbers
- Addition
  - Add two numbers and a carry
  - Hole: What is the resulting digit and carry on each step
  - (Note: not MNIST digits, but actual numbers)
- Word Algebra Problems
  - E.g. “Ann has 8 apples. She buys 4 more. She distributes them equally among her 3 kids. How many apples does each child receive?
  - Hole: Sequence of permuting, swapping and performing operations on the three numbers

[1]: Matko Bosnjak, Tim Rocktäschel, Jason Naradowsky, Sebastian Riedel: Programming with a Differentiable Forth Interpreter. ICML 2017: 547-556



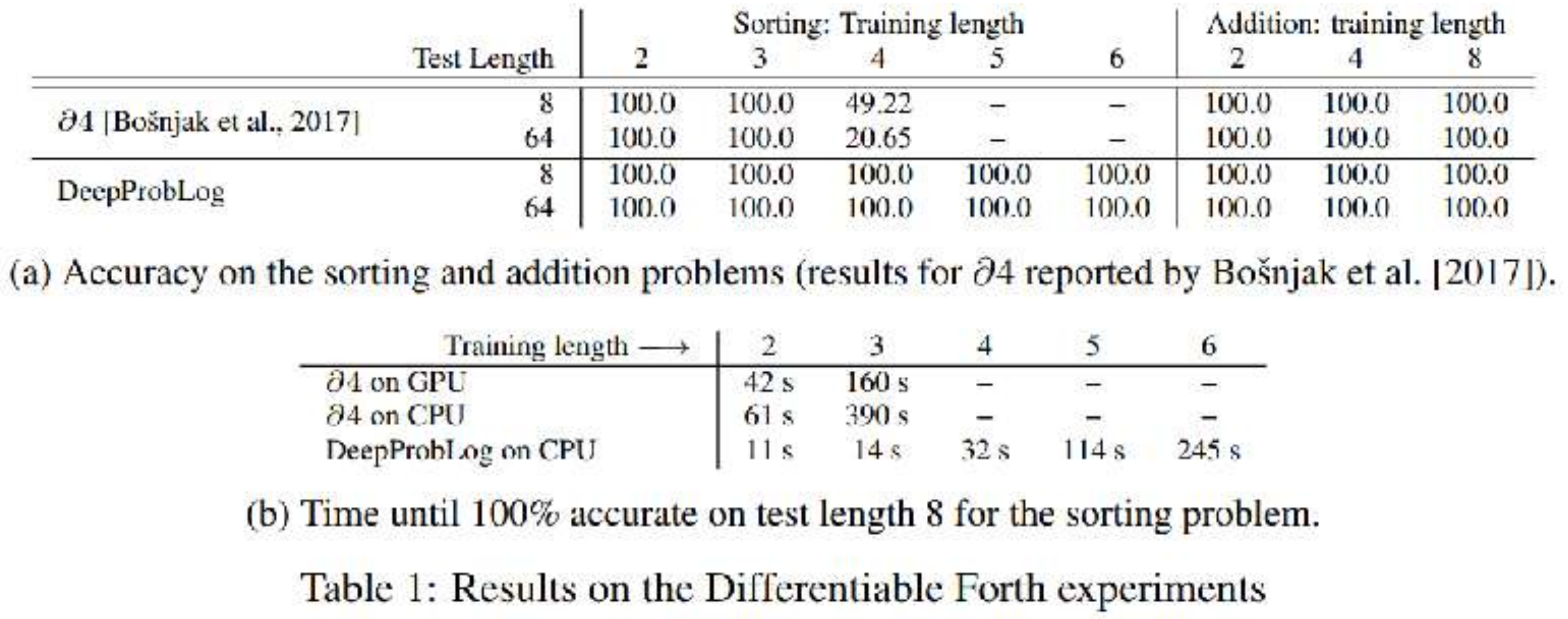
Example DeepProbLog

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).
```



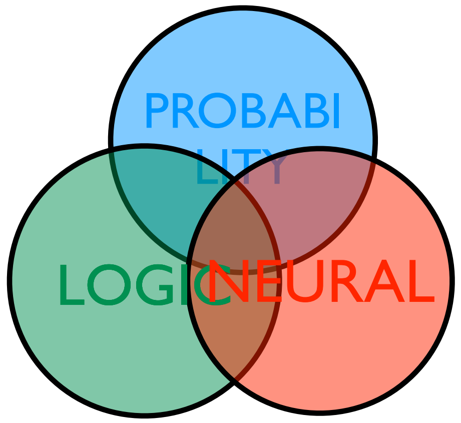
	Test Length	Sorting: Training length 2	3	4	5	6	Addition: training length 2	4	8
∂4 [Bošnjak et al., 2017]	8	100.0	100.0	49.22	–	–	100.0	100.0	100.0
	64	100.0	100.0	20.65	–	–	100.0	100.0	100.0
DeepProbLog	8	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	64	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

(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



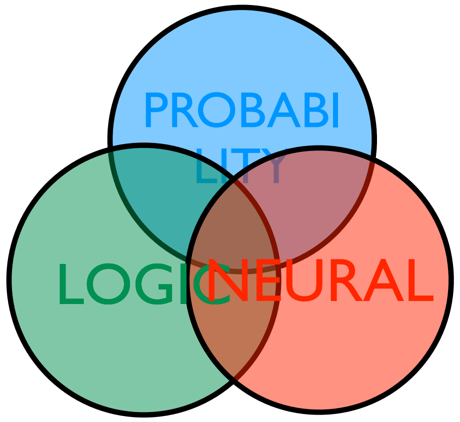
Noisy Addition

```
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.
```

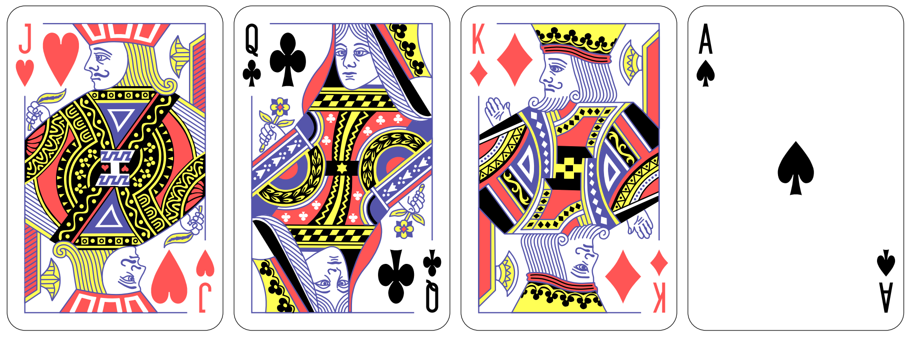
(a) The DeepProbLog program.

	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**.

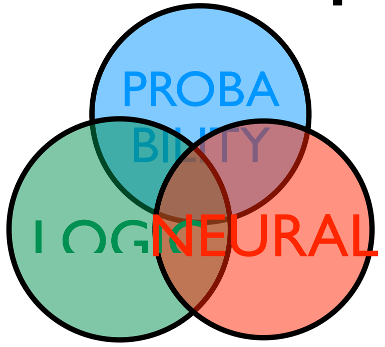


Simplified Poker



- dealing with uncertainty
- ignore suits and just with A, J, Q and K
- two players, two cards, and one community card
  - train the neural network to recognize the four cards
  - reason probabilistically about the non-observed card
  - learn the distribution of the unlabeled community card
- 0.8 :: poker([Q♡, Q♢, A♢, K♣], loss)   poker([Q♡, Q♢, A♢, K♣], A♢, loss).

in 6/10 experiments



Distribution	Jack	Queen	King	Ace
Actual	0.2	0.4	0.15	0.25
Learned	0.203 ± 0.002	0.396 ± 0.002	0.155 ± 0.003	0.246 ± 0.002

Table 8: The results for the Poker experiment (**T9**).

Neural Theorem Prover

Towards Neural Theorem Proving at Scale

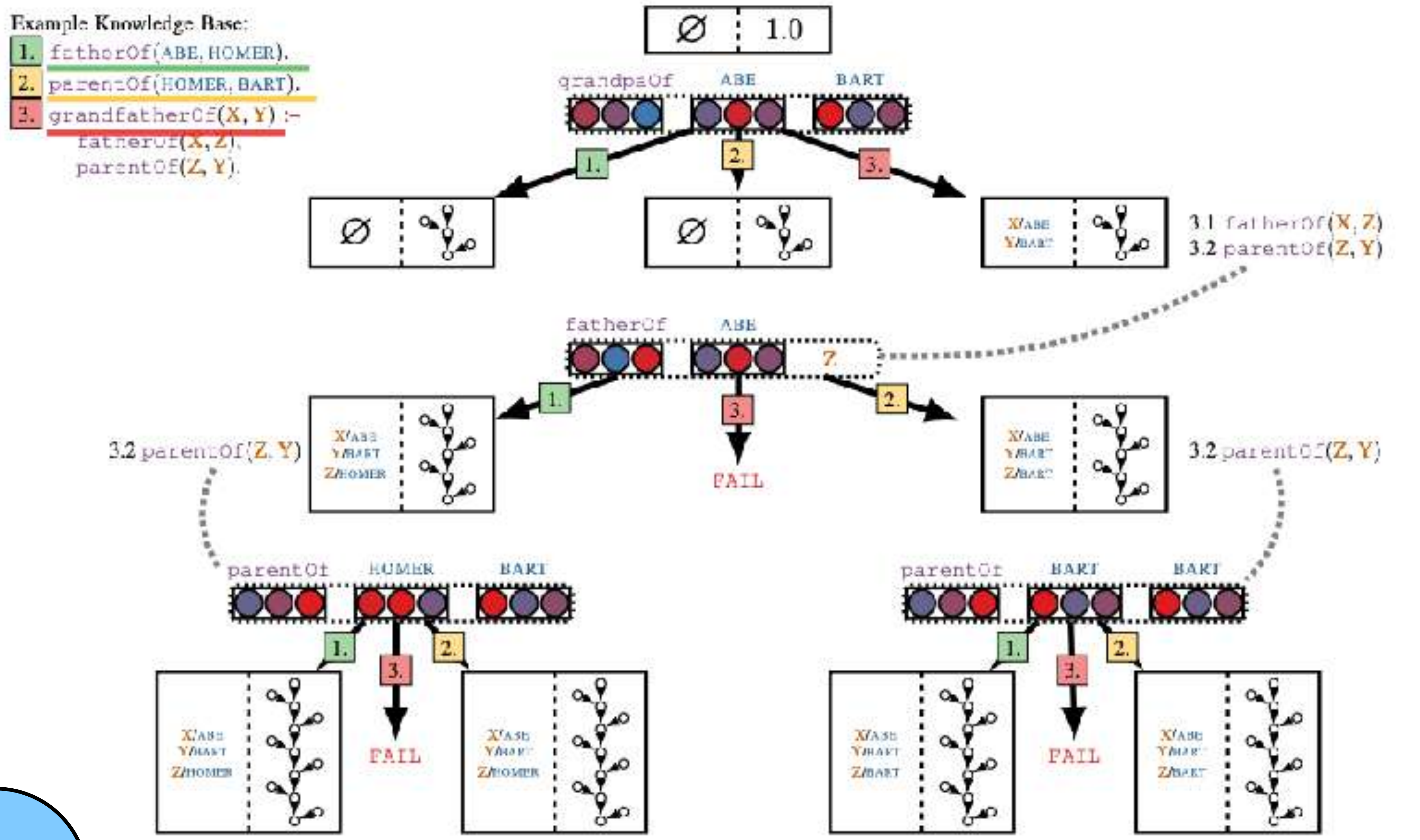
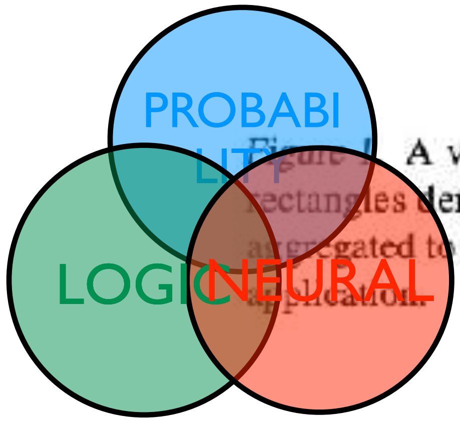


Figure 1. A visual depiction of the NTP' recursive computation graph construction, applied to a toy KB (top left). Dash-separated rectangles denote proof states (left: substitutions, right: proof score -generating neural network). All the non-FAIL proof states are aggregated to obtain the final proof 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).

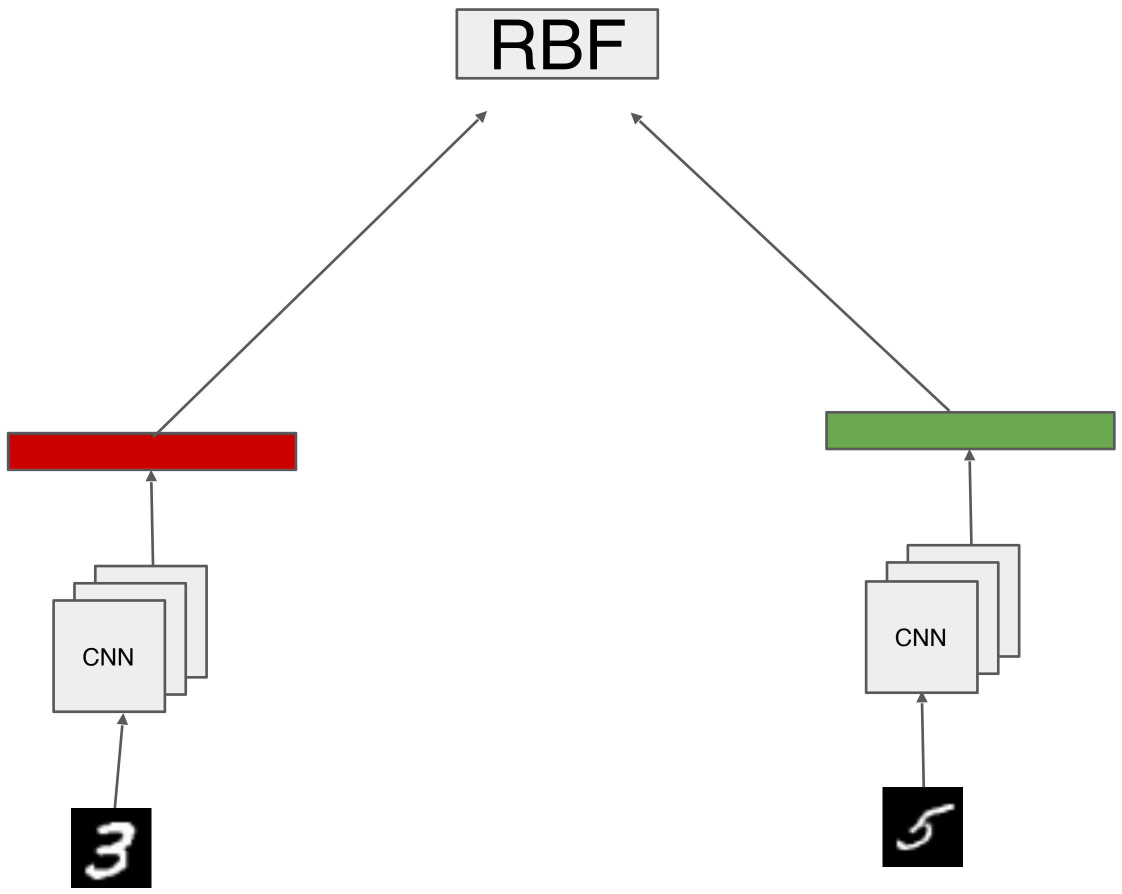
softunification(X,Y) returns 1 if X and Y unify

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

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

Embeddings in MNIST

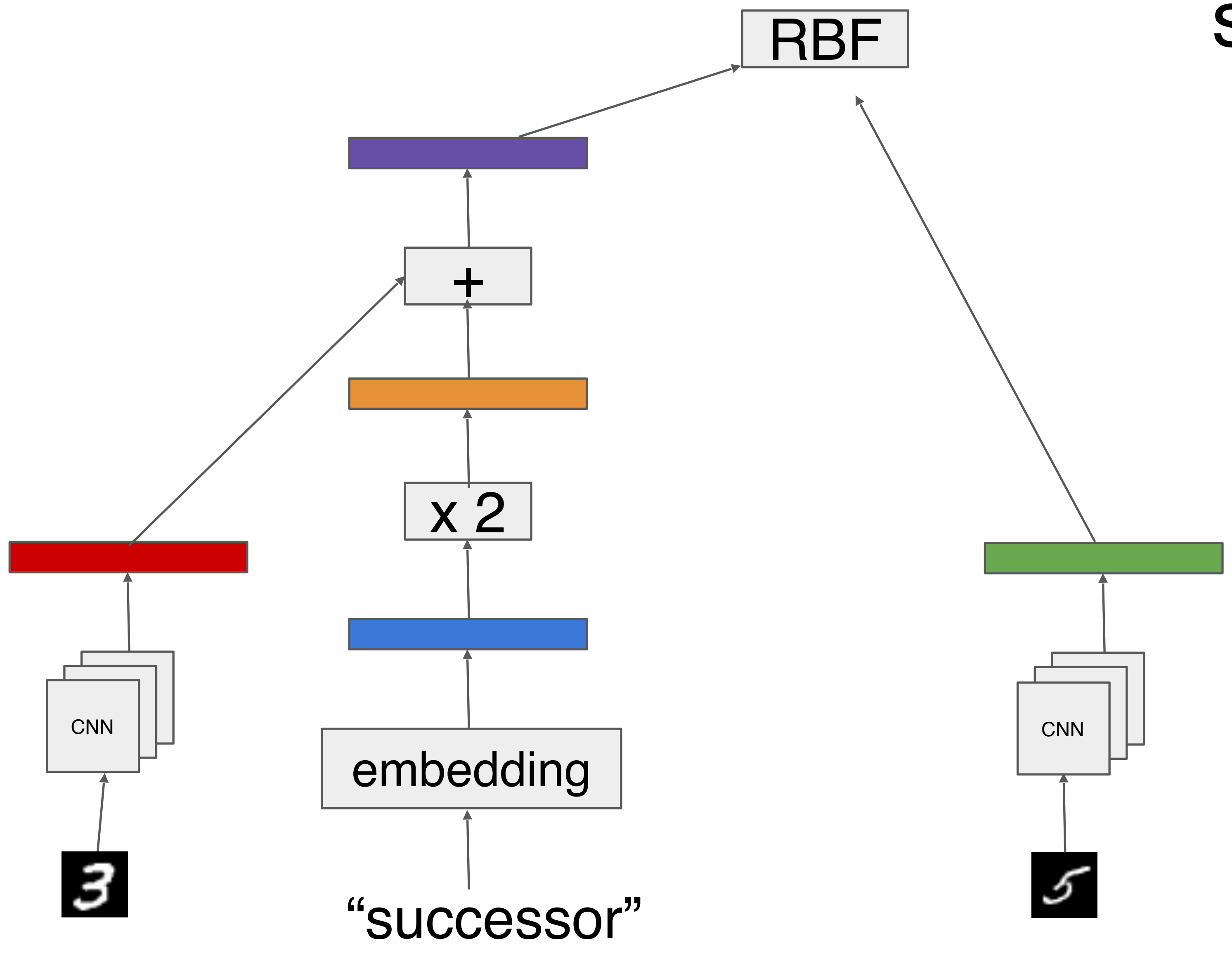
Computational Graph



soft(,) :-
cnn_embed(,e1),
cnn_embed(,e2),
rbf(e2,e3).

Embeddings in MNIST

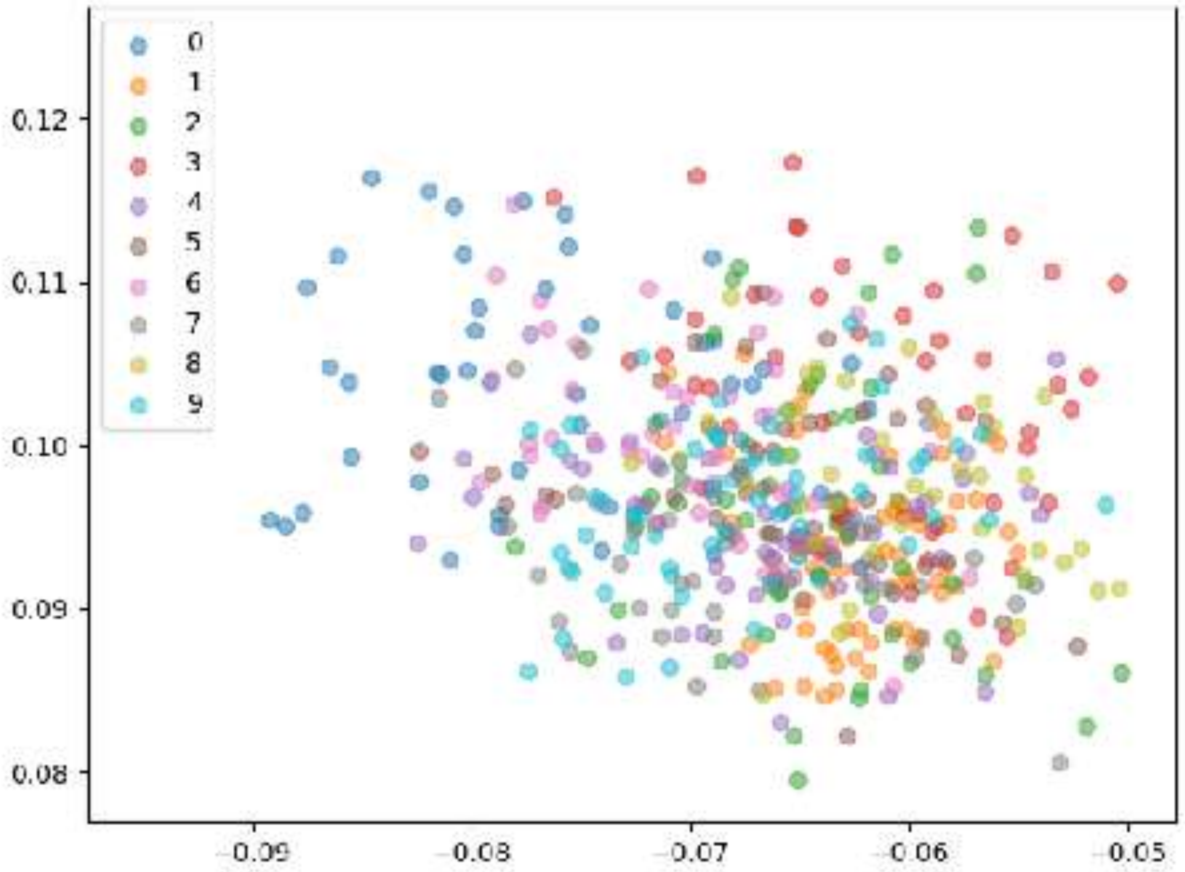
Computational Graph

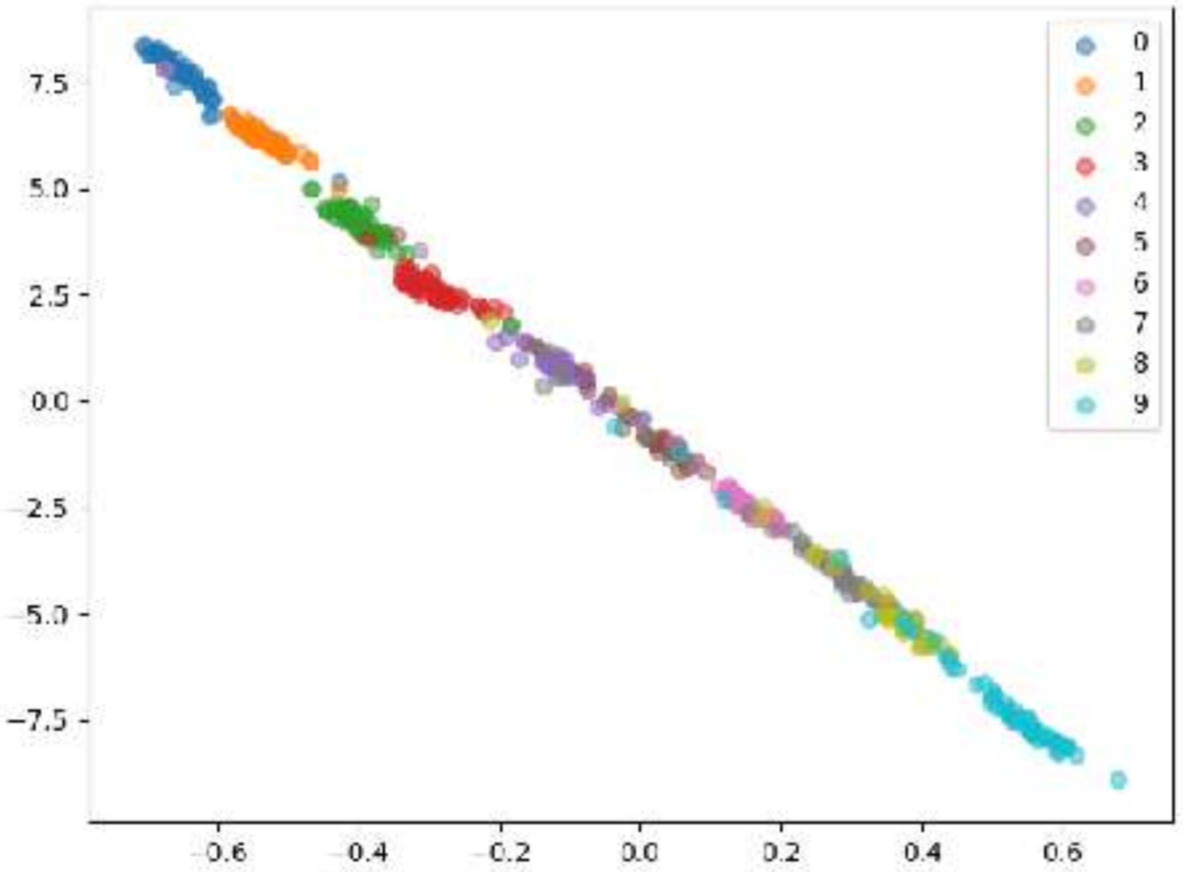


succesor_n(, ,2) :-
cnn_embed(,e1),
cnn_embed(,e2),
embed(“successor”,r),
mul(r,2,r2),
add(r2,e1,e3),
rbf(e2,e3).

Idea of TransE [Bordes et al]

2D MNIST image embeddings





The CLUTRR Dataset

Goal of the dataset [Sinha et al. EMNLP 19]:

Predict relations between named entities in the text that are not explicitly mentioned, but can be deduced using other mentioned relations.

E.g.

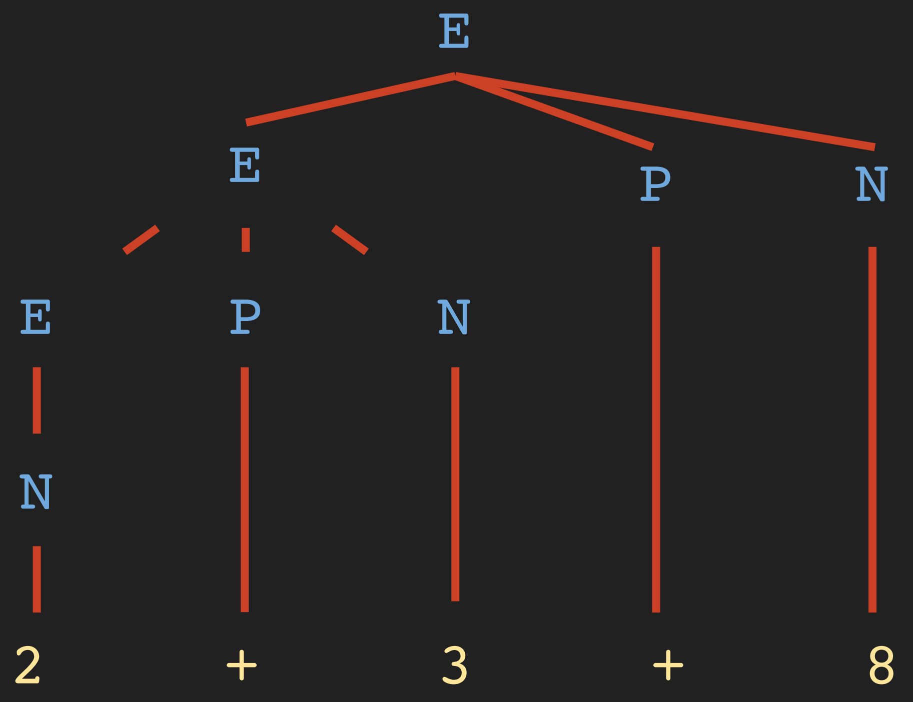
"Alice has a son called Bob. Bob has a brother called Charlie. Yesterday, Charlie and Bob went to visit Alice."

INFER son(alice,charlie)

FROM son(alice,bob) and brother(bob,charlie) .

CFG: Context-Free Grammar

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

P --> ["+"]

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

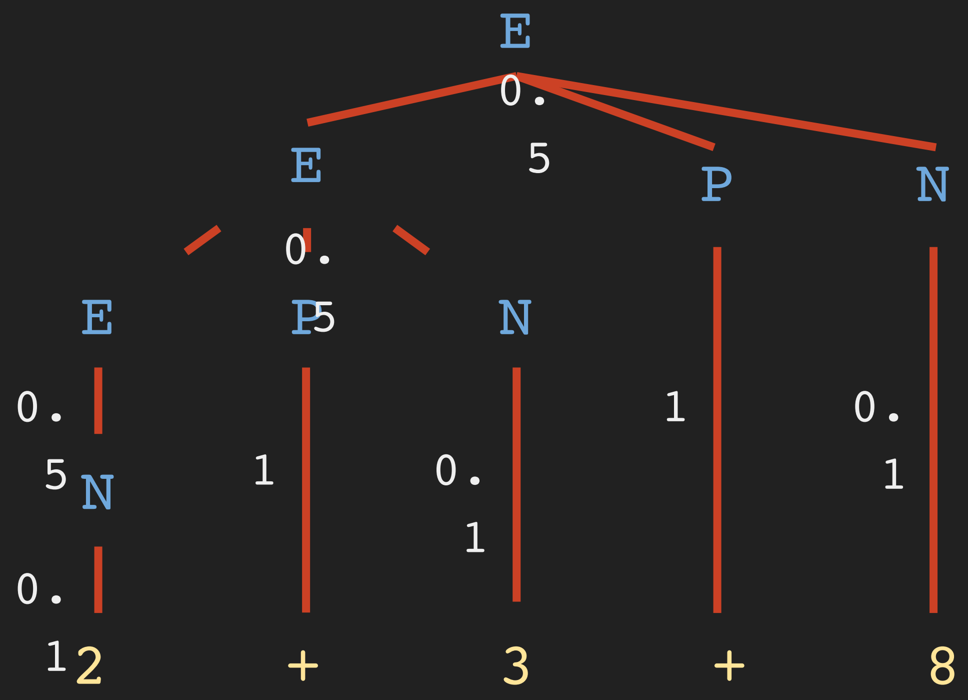


Useful for:
- Is sequence an element of the specified language?
- What is the *"part of speech"*-tag of a terminal
- Generate all elements of language

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*

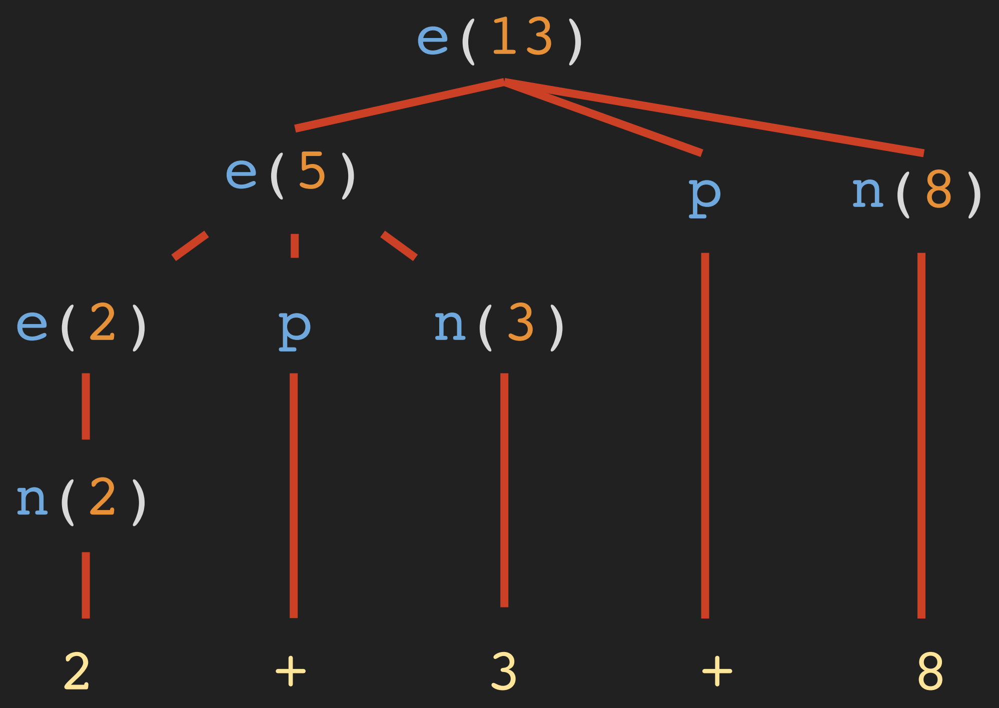
= 0.000125

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

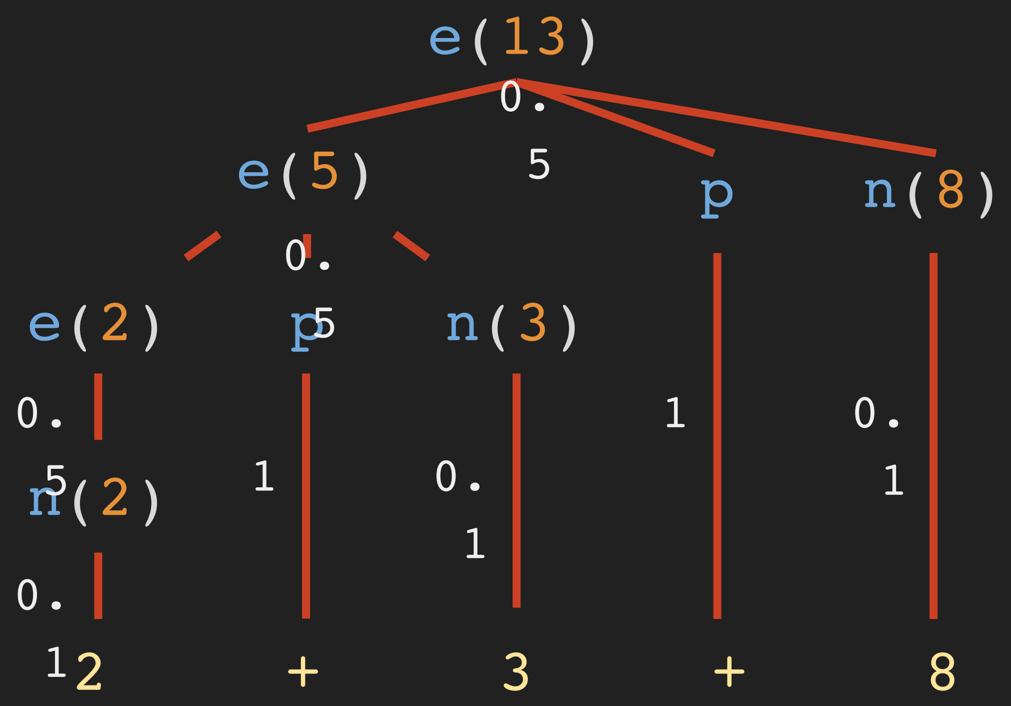
```
e(N) --> n(N).
e(N) --> e(N1), p, n(N2),
         {N is N1 + N2}.
p    --> [“+”].

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

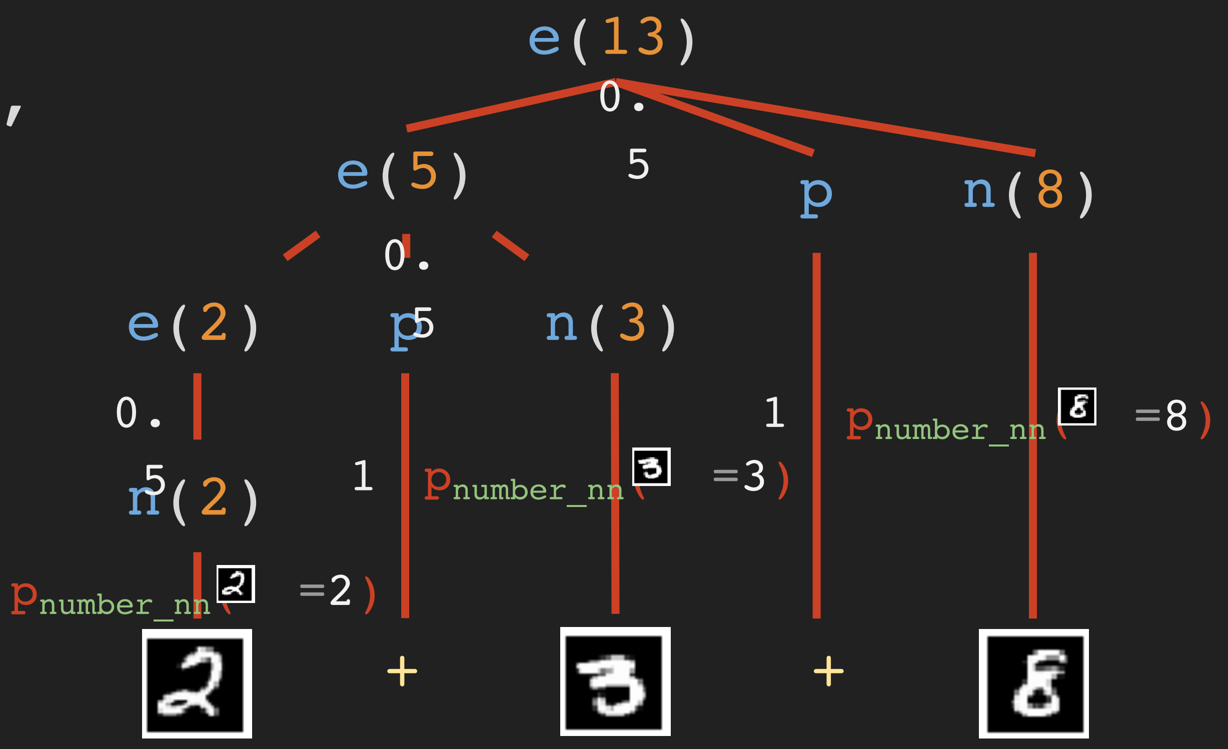
Useful for:

- 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_{num}
ber_nn(=8)

Useful for:

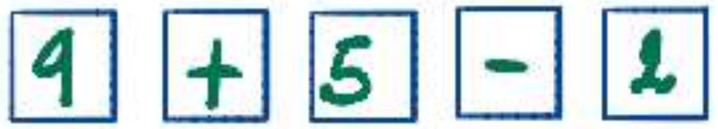
- Subsymbolic processing: e.g. tensors as terminals
- Learning rule probabilities using neural networks

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

DeepStochLog

```
Examples of the form
```



12

```
digit(Y) :- member(Y [0,1,2,3,4,5,6,7,8,9]).
op(Y) :- member(Y, [+,-]).
```

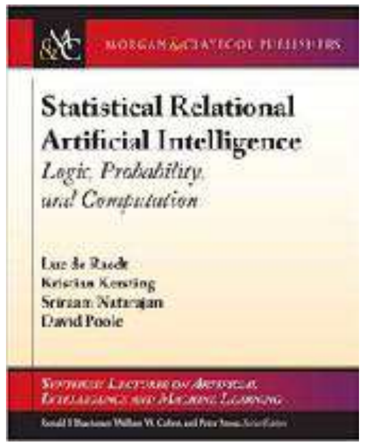
```
nn(mnist,[I],[N],[digit]) :: n(N) --> [I].
nn(operator,[I],[N],[op]) :: o(N) --> [I].
```

```
0.33::e(N) --> n(N).
0.33::e(S) --> e(E1), o(+), n(E2), {S is E1 + E2}.
0.33::e(S) --> e(E1), o(-), n(E2), {S is E1 - E2}.
```

Challenges

- For NeSy, DeepProbLog and others
  - scaling up (in DeepProbLog — now has both approximate and exact inference — an A* like algorithm to find the best proofs)
  - which models to use
  - real life applications
  - peculiarities of neural nets
    - need to have a signal (cf. addition of images only, and Poker …); aka curriculum learning + regularization
- This is an excellent area for starting researchers / PhDs

Key Message 1


MORGAN & CLAYPOOL PUBLISHERS

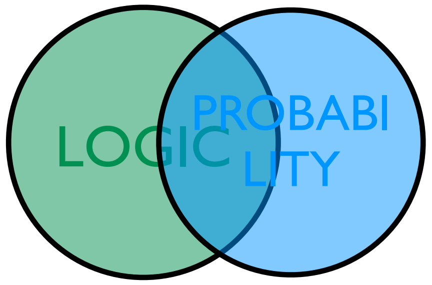
Statistical Relational Artificial Intelligence
Logic, Probability, and Computation

Luc de Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

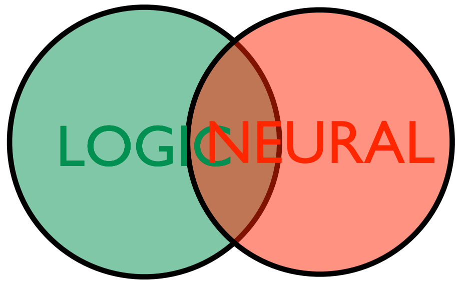
Synthesis Lectures on Artificial Intelligence and Machine Learning

Ronald J. Brachman, William W. Cohen, and Peter Stone, Series Editors

FROM



TO



StarAI and NeSy share similar problems and thus similar solutions apply

Part 1 of the talk

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

Key Message 2

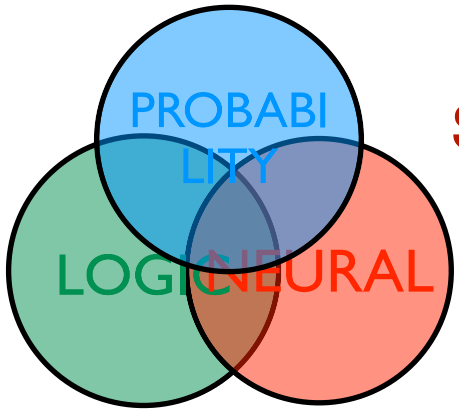
A different approach

A true integration T of X and Y should allow to reconstruct X and Y as special cases of T

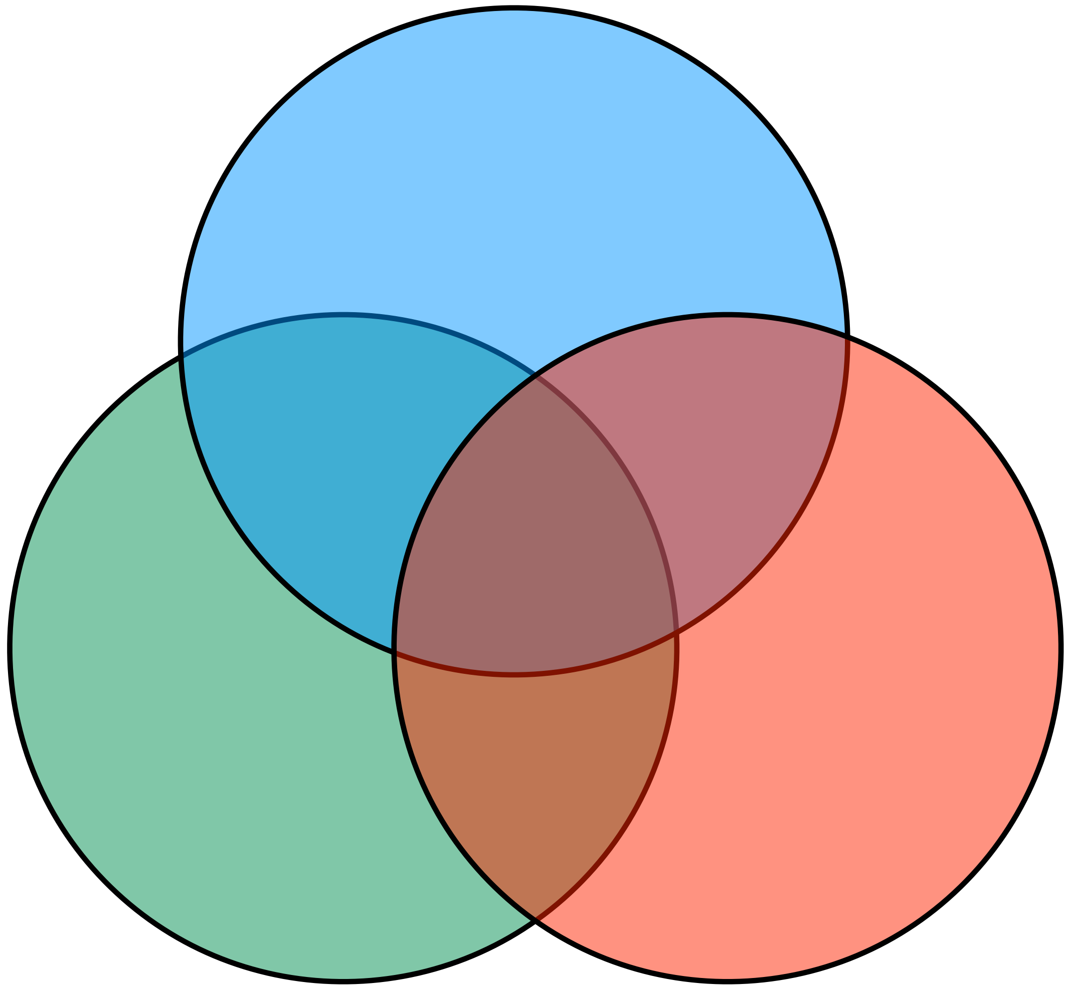
Thus, Neural Symbolic approaches should have both logic and neural networks as special cases

Our approach: “an interface layer (<> pipeline) between neural & symbolic components” will be illustrated with DeepProbLog

See also [Manhaeve et al., NeurIPS 18; arXiv: 1907.08194]



Part 2 of the talk — illustration with DeepProbLog [NeurIPS 2018]



THANKS