

DeepProbLog

Programação Lógica Neuro-Probabilística

Parte 1 de 2

CPS840 – Tópicos Especiais em Inteligência Artificial
Professor: Gerson Zaverucha

Cleiton Moya de Almeida

Rio de Janeiro, 26 de agosto de 2020

Agenda

Parte 1:

- **Introdução**
 - ✓ Abordagens em IA
 - ✓ Proposta do DeepProbLog
 - ✓ Exemplo
- **Programação em Lógica**
 - ✓ Conceitos Básicos
 - ✓ Prolog
- **ProbLog**
 - ✓ Definição
 - ✓ Inferência
 - ✓ Aprendizado

Parte 2:

- **Deep Learning**
- **DeepProbLog**
 - ✓ Definição
 - ✓ Inferência
 - ✓ Aprendizado
- **Experimentos**
 - ✓ Raciocínio Lógico e *Deep Learning*
 - ✓ Programação Indutiva
 - ✓ Programação Probabilística e *Deep Learning*
- **Trabalhos correlatos**
- **Conclusões**

Referências

- **Artigo principal:**

- ✓ **Título:** *Neural Probabilistic Logic Programming in DeepProbLog*
- ✓ **Autores:** Robin Manhaeve^a, Sebastijan Dumančić^a, Angelika Kimmig^b, Thomas Demeester^c, Luc De Raedt^a
^aKatholieke Universiteit Lueven (Bélgica), ^bCardiff University (País de Gales), ^cGhent Univesity – imec (Bélgica)
- ✓ Versão estendida de: *DeepProbLog: Neural Probabilistic Logic Programming*, publicado anteriormente no NeurIPS 2018
- ✓ Disponível em: <https://arxiv.org/abs/1907.08194>

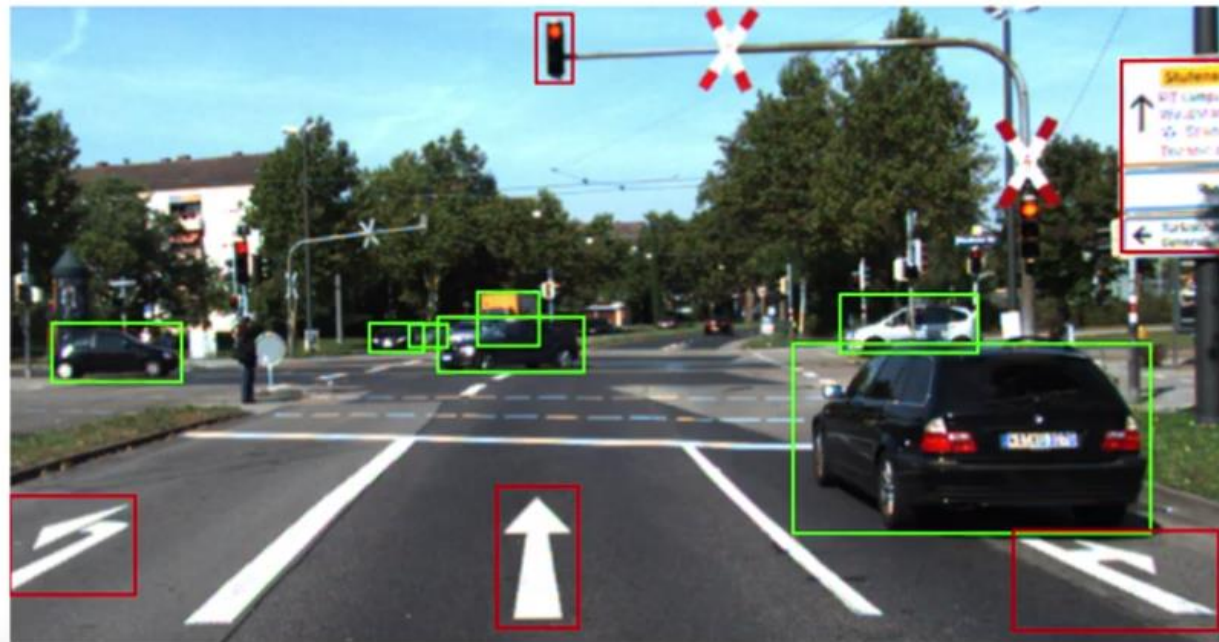
- **Outras de ProbLog:**

- ✓ Arenstein, Lucas Silva. ProbLog – Programação Lógica Probabilística: Um Estudo e Aplicação. Trabalho de Conclusão de Curso. Universidade de São Paulo, 2018.
- ✓ Tutoriais de ProbLog: <https://dtai.cs.kuleuven.be/problog/tutorial.html>
- ✓ Editor *online* de ProbLog: <https://dtai.cs.kuleuven.be/problog/editor.html>
- ✓ Publicações em ProbLog: <https://dtai.cs.kuleuven.be/problog/publications.html>

Introdução

- Abordagens em IA

How should I drive?

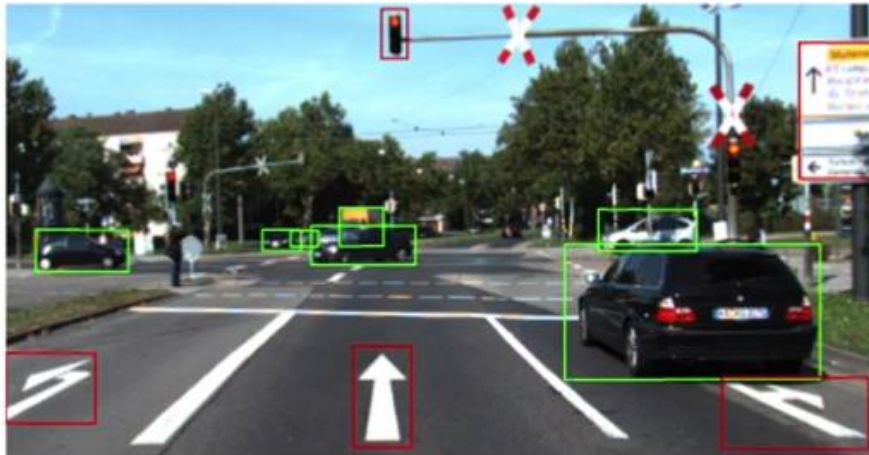


Fonte: Robin Manhaeve - DeepProbLog: Neural Probabilistic Logic Programming
<https://www.youtube.com/watch?v=b-AC428qhdQ&t=56s>

Introdução

- Abordagens em IA

Sub-symbolic perception



Fonte: Robin Manhaeve - DeepProbLog: Neural Probabilistic Logic Programming
<https://www.youtube.com/watch?v=b-AC428qhdQ&t=56s>

Deep Learning

Reasoning with knowledge

How should I drive, given:

Stop in front of a red light
Obey the speed limit
Be in the correct lane

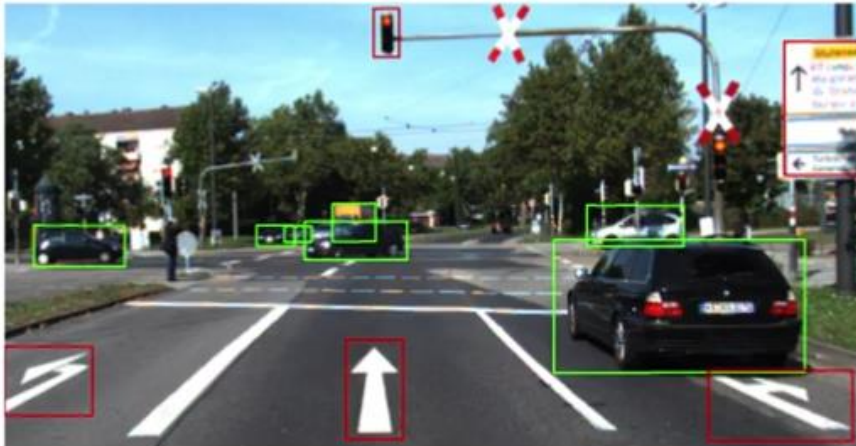
...

Lógica

Introdução

■ Abordagens em IA

Sub-symbolic perception



Fonte: Robin Manhaeve - DeepProbLog: Neural Probabilistic Logic Programming
<https://www.youtube.com/watch?v=b-AC428qhdQ&t=56s>

Deep Learning

Reasoning with knowledge
under uncertainty

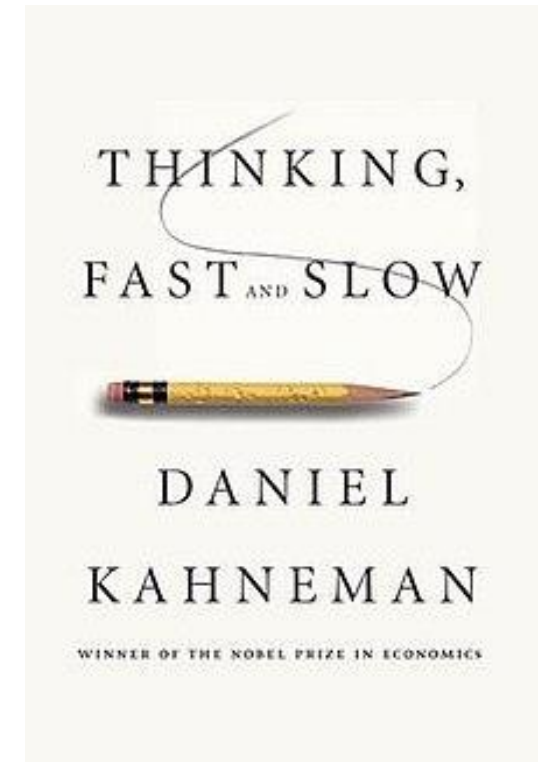
How should I drive, given:

Stop in front of a red light	$P(\text{light} = \text{red}) = 0.9$
Obey the speed limit	$P(\text{obj1} = \text{car}) = 0.8$
Be in the correct lane	$P(\text{obj1 turn right}) = 0.7$
...	

Lógica Probabilística

Introdução

- Abordagens em IA
 - **Integração neuro-simbólica-probabilística:**
 - ✓ Objetivo:
 - Flexibilidade do raciocínio lógico e probabilístico +
 - Poder de representação da redes neurais;
 - **Propriedade desejada:**
 - ✓ Os modelos puros neurais, lógicos e probabilísticos devem ser casos especiais do modelo neuro-simbólico-probabilístico.



Introdução

■ Proposta do DeepProbLog

• DeepProbLog:

- ✓ Estende a linguagem **ProbLog** [1] através de neuro-predicados;
- ✓ Ao invés de integrar capacidade de raciocínio nas redes neurais, faz o caminho inverso;
- ✓ Integra as redes neurais ao modelo de Programação Lógica-Probabilística;
- ✓ **Lógica probabilística**: Expressão atômica $q(t_1, \dots, t_N)$ (também conhecida como *tupla* em uma banco de dados relacional) possui uma **probabilidade** p .
- ✓ **Predicado neural**: expressão é rotulada com uma rede neural cujas saídas possam ser consideradas distribuições de probabilidade.



• Vantagens da inclusão de probabilidade:

- ✓ Simplifica a integração de redes neurais com lógica:
 - Provê um critério claro de otimização (probabilidade dos exemplares);
 - Valores de probabilidade são adequados para se trabalhar com gradiente descendente.

Introdução

■ Exemplo

- **Operação com dígitos da base MNIST:**

- ✓ Ex.: `addition(, 8)`
- ✓ Predicado: `addition(X, Y, Z)`
- ✓ X, Y : Imagens de dígitos
- ✓ Z : número natural correspondente à soma de X e Y ;

- **Solução somente com Redes Neurais:**

- ✓ Permite a aprendizagem do predicado;
- ✓ Porém não incorpora o conhecimento prévio.

- **Solução com DeepProbLog:**

- ✓ Conhecimento prévio de adição facilmente modelado através de regra:
 - `addition(I_X, I_Y, N_Z) :- digit(I_X, N_X), digit(I_Y, N_Y), N_Z is $N_X + N_Y$`
- ✓ Rede precisa aprender apenas o predicado `digit`;
- ✓ Rede pode ser re-utilizada com outras tarefas;
- ✓ Pode ser estendida a múltiplos dígitos sem necessidade de mais treinamento.

Programação em Lógica

■ Conceitos Básicos

- **Átomo:**

- ✓ Expressão na forma $q(t_1, \dots, t_n)$, onde q é o predicado (de aridade n , ou q/n em notação simplificada) e t_i são os termos;

- **Literal:**

- ✓ Átomo ou negação $\neg q(t_1, \dots, q_n)$ de um átomo;

- **Termo:**

- ✓ Uma constante c , uma variável V ou um termo estruturado na forma $f(u_1, \dots, u_k)$, onde f é um functor de termos u_i ;

- **Regra:**

- ✓ Expressão na forma $h : - b_1, \dots, b_n$
 - $-$ representa implicação lógica (\leftarrow);
 - Vírgula (,) representa conjunção (\wedge);
 - **Fato:** regra com corpo vazio ($n = 0$).

- **Programa lógico:**

- ✓ Conjunto finito de regras.

Programação em Lógica

■ Conceitos Básicos

- **Substituição:**

- ✓ Uma substituição $\theta = \{V_1 = t_1, \dots, V_n = t_n\}$ é uma atribuição de termos t_i às variáveis V_i ;
- ✓ Quando aplicamos θ a uma expressão e , substituímos todas as ocorrências de V_i por t_i e denotamos a expressão resultante como $e\theta$.

- **Ground facts:**

- ✓ Expressões que não contém nenhuma variável são chamadas de *ground*.

- **Inferência:**

- ✓ Pergunta: *query* q é verdadeira no modelo canônico do programa lógico P ? Caso sim:
 - $P \models q$
 - q é uma consequência lógica de P

Progamação em Lógica

■ Prolog

• Termos

- ✓ Constantes: a, b, c (minúscula);
- ✓ Variáveis: X, Y, Z (maiúscula);
- ✓ $f(u_1, \dots, u_k)$;

• Regra:

- ✓ Ex.: $\text{avô}(X,Y) \text{ :- } \text{pai}(X,Z), \text{pai}(Z,Y)$

• Fato:

- ✓ $\text{pai}(\text{josé}, \text{joão})$.

Exemplo de programa ProLog:

```
pai(josé, joão).  
pai(manoel, josé).  
avô(X,Y) :- pai(X,Z), pai(Z,Y).
```

Pergunta:

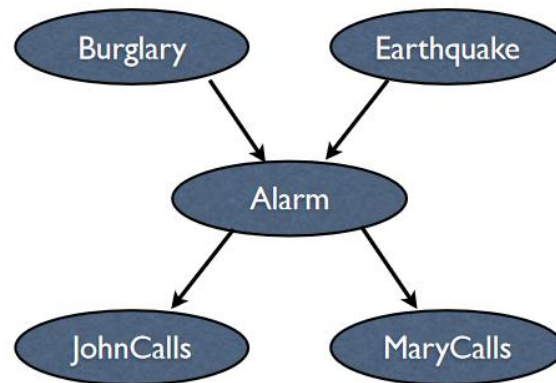
?-avô(X,joão)

- ✓ Editor *online* de Prolog:
 - <https://swish.swi-prolog.org/>

ProbLog

■ Definição

- ✓ **Programa ProbLog:** Conjunto \mathcal{F} de *fatos probabilísticos* + conjunto \mathcal{R} de *regras*;
- ✓ $p :: f$ são fatos probabilísticos, onde p é a probabilidade e f um *átomo probabilístico*.



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

Exemplo de programa ProbLog

ProbLog

■ Definição

- ✓ Cada fato probabilístico $p :: f$ corresponde a uma *variável aleatória independente*;
- ✓ Cada fato probabilístico possui uma *decisão atômica*, ou seja, podemos incluí-lo com probabilidade p ou descartá-lo com probabilidade $(1 - p)$;
- ✓ Definimos como *escolha total (ET)* uma decisão atômica para cada um dos fatos;

$ET_i \rightarrow$

E	B	H A(M)	H A(J)	P(ET _i)
F	F	F	F	0,216
F	F	F	V	0,216
F	F	V	F	0,144
F	F	V	V	0,144
F	V	F	F	0,054
F	V	F	V	0,054
F	V	V	F	0,036
F	V	V	V	0,036
V	F	F	F	0,024
V	F	F	V	0,024
V	F	V	F	0,016
V	F	V	V	0,016
V	V	F	F	0,006
V	V	F	V	0,006
V	V	V	F	0,004
V	V	V	V	0,004

```

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

Exemplo de programa ProbLog

ProbLog

■ Definição

- ✓ A união de uma escolha total $F \subseteq \mathcal{F}$ conjunto de regras \mathcal{R} define um **mundo possível** w_F ;
- ✓ Um **mundo possível** w_F é um modelo de **programa lógico**.

$$w_{\{\text{burglary, hears_alarm(mary)}\}} = \{\text{burglary, hears_alarm(mary)}\} \cup \{\text{alarm, calls(mary)}\}$$

- ✓ A probabilidade $P(w_F)$ deste mundo possível é então o produto das probabilidades dos valores verdades dos fatos probabilísticos:

$$P(w_F) = \prod_{f_i \in F} p_i \prod_{f_i \in \mathcal{F} \setminus F} (1 - p_i)$$

$$P(w_{\{\text{burglary, hears_alarm(mary)}\}}) = 0.1 \times 0.5 \times (1 - 0.2) \times (1 - 0.4) = 0.024$$

- ✓ A probabilidade de um fato *ground* (query q) é então definida como a soma da probabilidade de todos os “mundos” contendo q :

$$P(q) = \sum_{F \subseteq \mathcal{F}: q \in w_F} P(w_F)$$

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

B	E	H A(J)	H A(M)	P(Eti)
F	F	F	F	0,216
F	F	F	V	0,216
F	F	V	F	0,144
F	F	V	V	0,144
F	V	F	F	0,054
F	V	F	V	0,054
F	V	V	F	0,036
F	V	V	V	0,036
V	F	F	F	0,024
V	F	F	V	0,024
V	F	V	F	0,016
V	F	V	V	0,016
V	V	F	F	0,006
V	V	F	V	0,006
V	V	V	F	0,004
V	V	V	V	0,004

ProbLog

■ Inferência

- ✓ Pergunta (*query*): `calls(mary)`
- ✓ **Passo1:** Programa lógico é instanciado

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

Programa ProbBlog



```
0.2::earthquake.  
0.1::burglary.  
0.5::hears_alarm(mary).  
  
alarm :- earthquake.  
alarm :- burglary.  
calls(mary):-alarm,hears_alarm(mary).
```

Programa lógico instanciado

ProbLog

■ Inferência

- ✓ **Passo2:** Determinação da fórmula proposicional:
 $\text{calls}(\text{mary}) \leftrightarrow \text{hears_alarm}(\text{mary}) \wedge (\text{burglary} \vee \text{earthquake})$
- ✓ Podemos aplicar a *contagem ponderada dos modelos* (WMC) diretamente na fórmula, porém não é eficiente.

Models of $\text{calls}(\text{mary}) \leftrightarrow \text{hears_alarm}(\text{mary}) \wedge (\text{burglary} \vee \text{earthquake})$	w
{}	0.36
{hears_alarm(mary)}	0.36
{earthquake}	0.09
{earthquake, hears_alarm(mary), calls(mary)}	0.09
{burglary}	0.04
{burglary, hears_alarm(mary), calls(mary)}	0.04
{burglary, earthquake}	0.01
{burglary, earthquake, hears_alarm(mary), calls(mary)}	0.01
$\sum_{\text{calls}(\text{mary}) \in \text{model}}$	0.14

(c) The weighted count of the models where $\text{calls}(\text{mary})$ is true.

B	E	H A(M)	P(Eti)
F	F	F	0,36
F	F	V	0,36
F	V	F	0,09
F	V	V	0,09
V	F	F	0,04
V	F	V	0,04
V	V	F	0,01
V	V	V	0,01

ProbLog

■ Inferência

- ✓ **Passo 3:** Compilação do conhecimento [2];
- ✓ Fórmula lógica é transformada em uma representação que permite a contagem ponderada dos modelos (WFC) de uma forma mais eficiente;
- ✓ ProbLog atualmente utiliza modelo SDD – *Sentential Decision Diagram* [3];

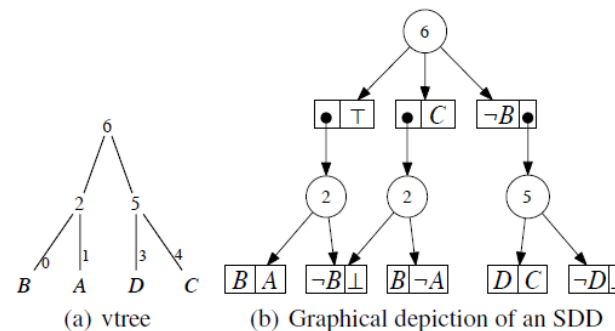


Figure 1: Function $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$.

Exemplo de SDD - Fonte: [2]

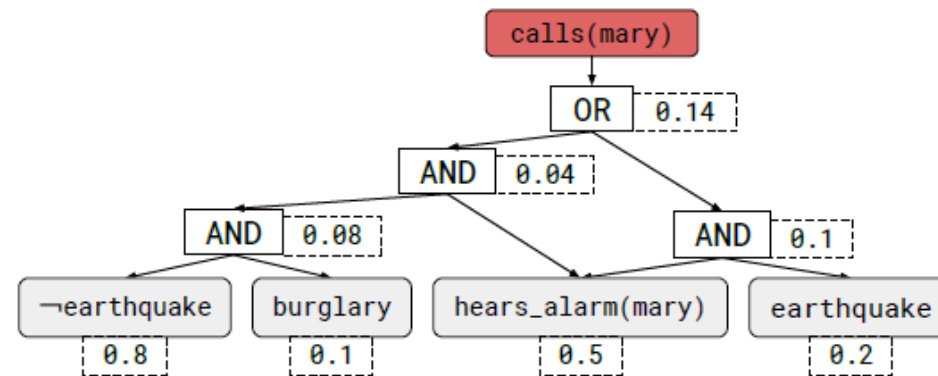
[2] A. Darwiche, P. Marquis, A knowledge compilation map, Journal of Artificial Intelligence Research 17 (2002) 229–264

[3] A. Darwiche, SDD: A new canonical representation of propositional knowledge bases, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI-11, 2011, pp. 819–826

ProbLog

■ Inferência

- ✓ **Passo 4:** Circuito aritmético;
- ✓ SDD transformado em um circuito aritmético (AC).
 - Folhas: fatos probabilísticos
 - Nodos “OR”: adição;
 - Nodos “AND”: multiplicação;
- ✓ Contagem ponderada dos modelos (WMC) é calculada através da avaliação do AC.



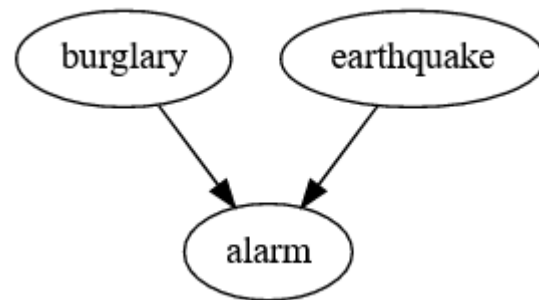
Circuito Aritmético

ProbLog

■ Aprendizado

- ✓ Aprendizagem das probabilidades dos fatos à partir de exemplares;
- ✓ Versão atual: ProbLog utiliza **aprendizado por interpretações** parciais (algoritmo LFI-ProbLog)[4].

• Exemplo



```

1  %%% The program:
2  t(0.5)::burglary.
3  0.2::earthquake.
4  t(_)::p_alarm1.
5  t(_)::p_alarm2.
6  t(_)::p_alarm3.
7
8  alarm :- burglary, earthquake, p_alarm1.
9  alarm :- burglary, \+earthquake, p_alarm2.
10 alarm :- \+burglary, earthquake, p_alarm3.
  
```

Examples (specified as evidence, separated by ---):

```

1  %%% The data:-
2  evidence(burglary,false).
3  evidence(alarm,false).
4  -----
5  evidence(earthquake,false).
6  evidence(alarm,true).
7  evidence(burglary,true).
8  -----
9  evidence(burglary,false).
  
```

Fact▼	Location	Probability
t(0.5)::burglary	2:9	0.33333333
t(_)::p_alarm1	4:7	0.83433737
t(_)::p_alarm2	5:7	1
t(_)::p_alarm3	6:7	0

ProbLog

■ Aprendizado

• Ideias principais – LFI-ProbLog

✓ Estimativa da Máxima Verossimilhança:

Definition 1 (Max-Likelihood Parameter Estimation). *Given a ProbLog program $\mathcal{T}(\mathbf{p})$ containing the probabilistic facts \mathcal{F} with unknown parameters $\mathbf{p} = \langle p_1, \dots, p_N \rangle$ and background knowledge \mathcal{BK} , and a set of (possibly partial) interpretations $\mathcal{D} = \{I_1, \dots, I_M\}$ (the training examples). Find maximum likelihood probabilities $\hat{\mathbf{p}} = \langle \hat{p}_1, \dots, \hat{p}_N \rangle$ such that*

$$\hat{\mathbf{p}} = \arg \max_{\mathbf{p}} P(\mathcal{D} | \mathcal{T}(\mathbf{p})) = \arg \max_{\mathbf{p}} \prod_{m=1}^M P_w(I_m | \mathcal{T}(\mathbf{p}))$$

✓ Dois casos:

- Interpretações completas (*full observability*): contagem das *true ground instances* para cada interpretação;
- Interpretações parciais (*partial observability*): esperança do número de *true ground instances* para cada interpretação.

ProbLog

■ Aprendizado

• Outra abordagem: Gradiente descendente com Algebraic Problog – aProbLog [5]

- ✓ Paper [4] mostrou que o último passo da inferência (avaliação do circuito lógico) é válida não somente com probabilidades, mas com qualquer estrutura algébrica do tipo semi-anel cumulativo (*cumulative semiring*);
- ✓ *Cumulative semiring*:
 - ✓ Adição é associativa, comutativa e possui um elemento neutro;
 - ✓ Multiplicação é associativa, comutativa e possui um elemento neutro;
- ✓ aProbLog:
 - ✓ Define adição;
 - ✓ Define multiplicação;
 - ✓ Define função de rotulagem para literais

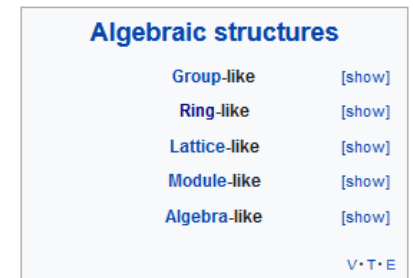
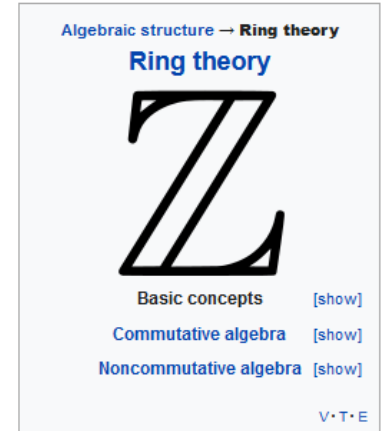
$$a \oplus b = a + b$$

$$a \otimes b = ab$$

$$e^{\oplus} = 0$$

$$e^{\otimes} = 1$$

$$\begin{aligned} L(f) &= p && \text{for } p :: f \\ L(\neg f) &= 1 - p && \text{with } L(f) = p \end{aligned}$$



Fonte: <https://en.wikipedia.org/wiki/Semiring>

ProbLog

■ Aprendizado

• Outra abordagem: Gradiente descendente com Algebraic Problog - aProbLog

- ✓ aProbLog utiliza uma *função de rotulagem* que associa explicitamente valores dos *semirings* com os fatos e suas negações

$$\begin{array}{ll} L(f) = p & \text{for } p :: f \\ L(\neg f) = 1 - p & \text{with } L(f) = p \end{array}$$

- ✓ *semiring* gradiente:

- Elementos são tuplas $\left(p, \frac{\partial p}{\partial \theta}\right)$

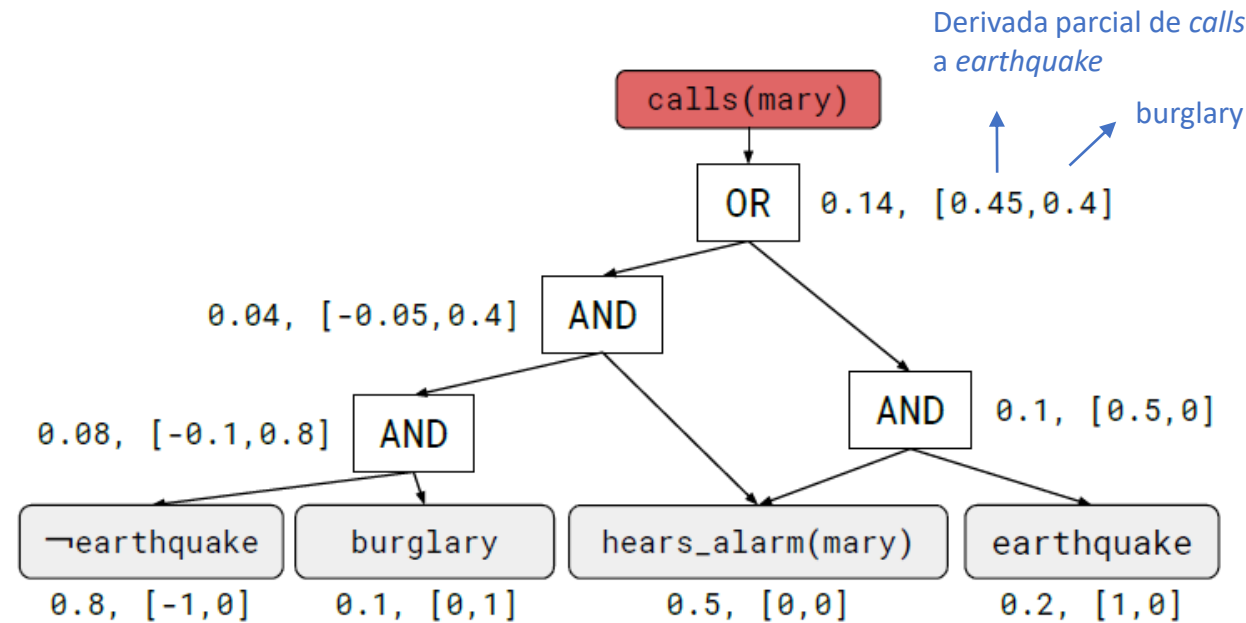
$$\begin{array}{lll} (a_1, \vec{a}_2) \oplus (b_1, \vec{b}_2) = (a_1 + b_1, \vec{a}_2 + \vec{b}_2) & L(f) = (p, \vec{0}) & \text{for } p :: f \text{ with fixed } p \\ (a_1, \vec{a}_2) \otimes (b_1, \vec{b}_2) = (a_1 b_1, b_1 \vec{a}_2 + a_1 \vec{b}_2) & L(f_i) = (p_i, \mathbf{e}_i) & \text{for } t(p_i) :: f_i \text{ with learnable } p_i \\ e^\oplus = (0, \vec{0}) & L(\neg f) = (1 - p, -\nabla p) & \text{with } L(f) = (p, \nabla p) \\ e^\otimes = (1, \vec{0}) & & \end{array}$$

ProbLog

■ Aprendizado

• Outra abordagem: Gradiente descendente com Algebraic Problog - aProbLog

- ✓ Exemplo: aprendizado da probabilidade de *earthquake* e *burglary* no exemplo anterior (*call(mary)*) mantendo a probabilidade dos outros fatos fixas;



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

Circuito AC avaliado com o *gradient semiring*

Obrigado
Dúvidas?