

# Language Model Programming: Lecture 4

Kyle Richardson<sup>α</sup>, Gijs Wijnholds<sup>β</sup>

Allen Institute for AI (AI2)<sup>α</sup>  
Leiden Institute of Advanced Computer Science (LIACS)<sup>β</sup>

July 2025

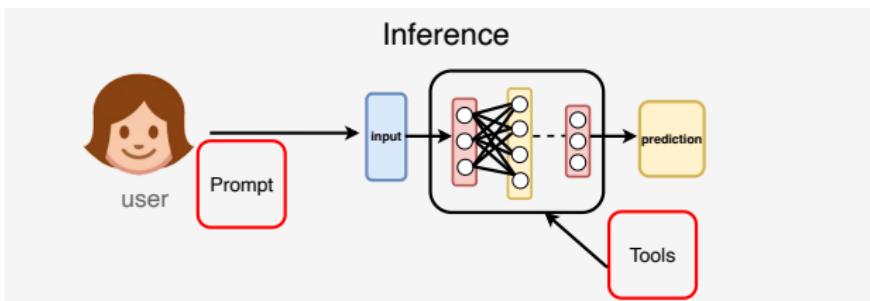
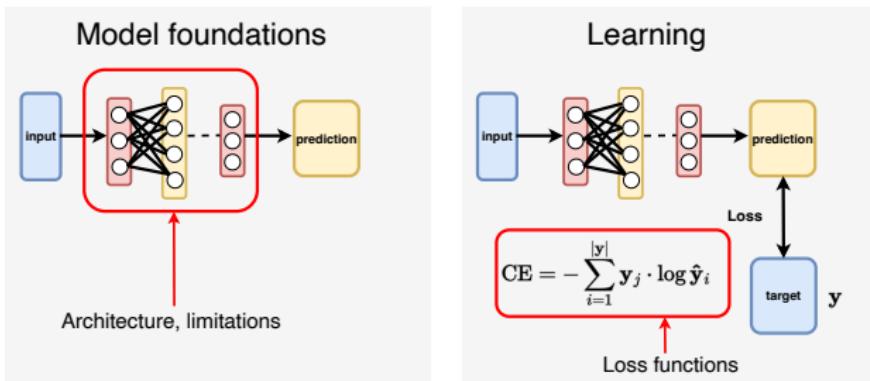


Leiden Institute of  
Advanced Computer  
Science

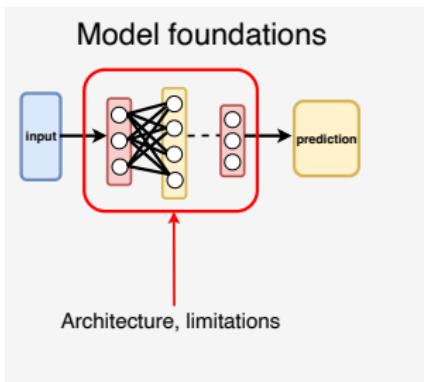


Universiteit  
Leiden

# Reminder of bigger picture



# Reminder of bigger picture



## RASPy

```
def flip():
    length = {key(1) == query{1}.value(z)
              flip = (key(length - indices - 1) == query{indices}).value(tokens)
              return flip
flip()
```

Input: h e l l o

1	1	1	1	1	1
1					
1					
1					
1					
5	5	5	5	5	5

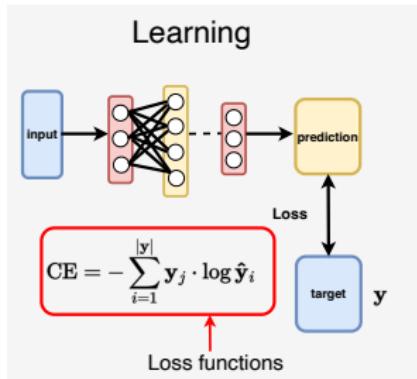
Layer 1

Layer 2

Final: o l l e h

0	1	2	3	4	h
3					
2					
1					
0					

# Reminder of bigger picture



```
1 // File path_planner.scl
2 type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)
3
4 const UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3
5 rel safe_cell(x, y) = range(0, 5, x), range(0, 5, y), not enemy(x, y)
6 rel edge(x, y, x, yp, UP) = safe_cell(x, y), safe_cell(x, yp), yp == y + 1
7 // Rules for DOWN, RIGHT, and LEFT edges are omitted...
8
9 rel next_pos(p, q, a) = actor(x, y), edge(x, y, p, q, a)
10 rel path(x, y, x, y) = next_pos(x, y, _)
11 rel path(x1, y1, x3, y3) = path(x1, y1, x2, y2), edge(x2, y2, x3, y3, _)
12 rel next_action(a) = next_pos(p, q, a), path(p, q, r, s), goal(r, s)
```

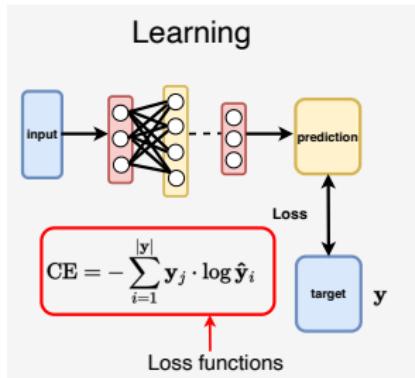
Fig. 3. The logic program of the PacMan-Maze application in Scallop.

# Reminder of bigger picture

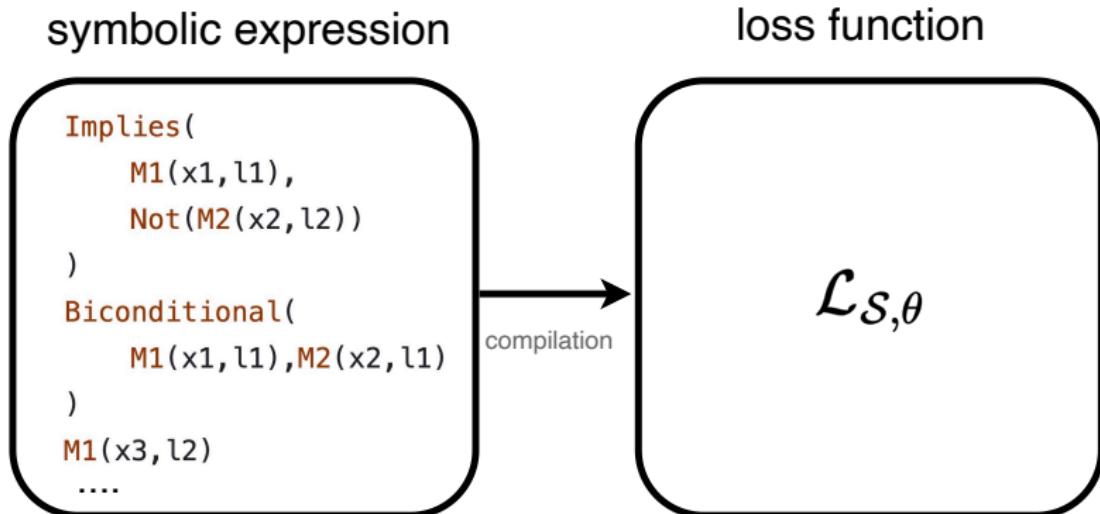
## Declarative Programming

```
1 // File path_planner.scl
2 type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)
3
4 const UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3
5 rel safe_cell(x, y) = range(0, 5, x), range(0, 5, y), not enemy(x, y)
6 rel edge(x, y, x, yp, UP) = safe_cell(x, y), safe_cell(x, yp), yp == y + 1
7 // Rules for DOWN, RIGHT, and LEFT edges are omitted...
8
9 rel next_pos(p, q, a) = actor(x, y), edge(x, y, p, q, a)
10 rel path(x, y, x, y) = next_pos(x, y, _)
11 rel path(x1, y1, x3, y3) = path(x1, y1, x2, y2), edge(x2, y2, x3, y3, _)
12 rel next_action(a) = next_pos(p, q, a), path(p, q, r, s), goal(r, s)
```

Fig. 3. The logic program of the PacMan-Maze application in Scallop.



## Lecture 2: Compiling high-level programs to loss



- ▶ **Declarative model programming**, saying what we want models to do, translating into a learning algorithm.

# Task-specific model fine-tuning

```
1 import torch
2 from transformers import \
3     AutoModelForSequenceClassification as hf_model
4
5 class NLIModel(torch.nn.Module):
6     def __init__(self, model_name):
7         super().__init__()
8         self.model = hf_model.from_pretrained(
9             model_name, ## base transformer
10            num_labels=3 ## 3 labels for NLI
11        )
12     def forward(self, features): ### forward pass
13         output = self.model(**features)
14         prob = output.logits.softmax(dim=-1)
15         return (output.loss, prob)
```

- ▶ Fine-tuning with task-specific parameters.

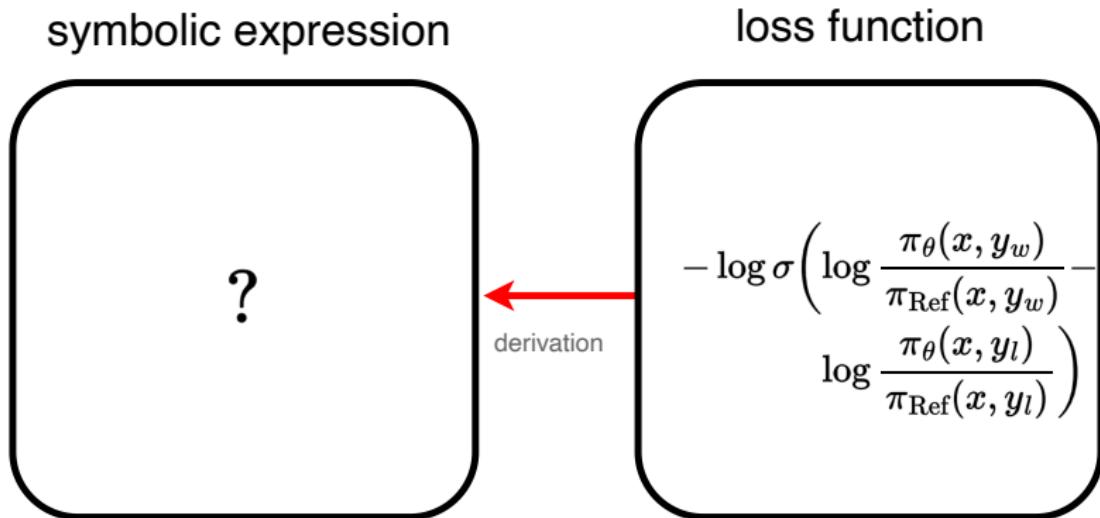
# Newer techniques for model fine-tuning

```
1 from transformers import T5ForConditionalGeneration
2 from peft import get_peft_model, LoraConfig, TaskType
3
4 model = T5ForConditionalGeneration.from_pretrained(
5     "t5-large"
6 )
7 peft_config = LoraConfig(
8     task_type=TaskType.SEQ_2_SEQ_LM,
9     r=8, lora_alpha=32,
10 )
11 model = get_peft_model(model, peft_config)
12 model.print_trainable_parameters()
13 # trainable params: 2359296 ||
14 # all params: 740027392 ||
15 # trainable%: 0.31881198257050464
```

see [peft library](#)

- ▶ **Parameter efficient fine-tuning:** Fine-tuning without tuning *all* of a model's parameters, rather a smaller set of *extra* parameters.

## Lecture 3: Going in the reverse direction



- ▶ **Decompilation**, going from known losses to symbolic programs, understanding direct preference alignment algorithms (*program analysis*).

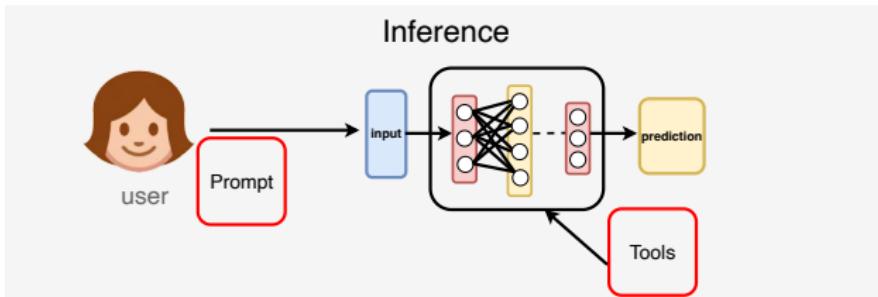
## Decoder-only models

```
1 | from transformers import AutoTokenizer as tokenizer,\n2 |                     AutoModelForCausalLM as model\n3 |\n4 | model_name = "HuggingFaceTB/SmollM2-135M"\n5 | tokenizer = tokenizer.from_pretrained(model_name)\n6 | model = model.from_pretrained(model_name)\n7 | ### example data\n8 | prompt = "Will drinking brake fluid kill you?"\n9 | loser = "No, it won't kill you"\n10 | winner = "No, but this could cause you serious harm"\n11 | loser_input = tokenizer.encode(\n12 |     f"{prompt} {loser}", return_tensors="pt"\n13 | )\n14 | winner_input = tokenizer.encode(\n15 |     f"{prompt} {winner}", return_tensors="pt"\n16 | )\n17 | model(winner_input)\n18 | model(loser_input)
```

# Today's lecture

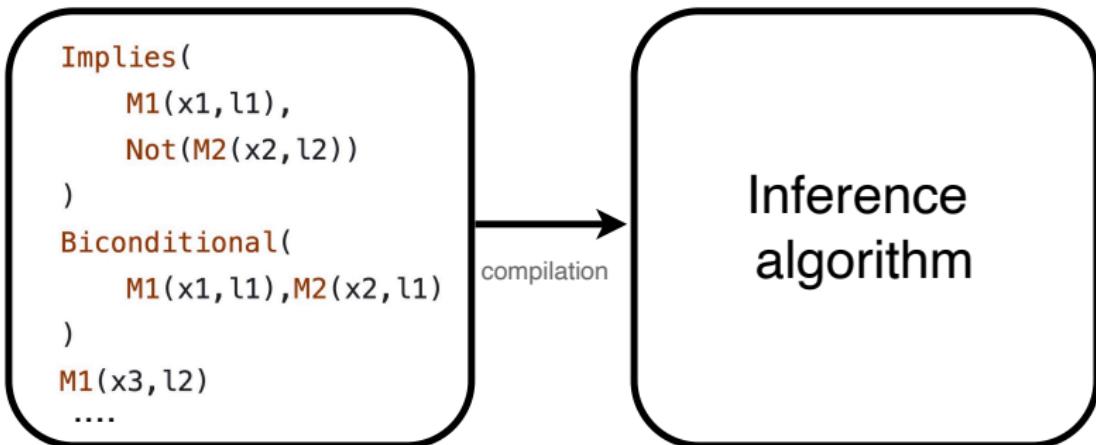
```
1 // File path_planner.scl
2 type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)
3
4 const UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3
5 rel safe_cell(x, y) = range(0, 5, x), range(0, 5, y), not enemy(x, y)
6 rel edge(x, y, x, yp, UP) = safe_cell(x, y), safe_cell(x, yp), yp == y + 1
7 // Rules for DOWN, RIGHT, and LEFT edges are omitted...
8
9 rel next_pos(p, q, a) = actor(x, y), edge(x, y, p, q, a)
10 rel path(x, y, x, y) = next_pos(x, y, _)
11 rel path(x1, y1, x3, y3) = path(x1, y1, x2, y2), edge(x2, y2, x3, y3, _)
12 rel next_action(a) = next_pos(p, q, a), path(p, q, r, s), goal(r, s)
```

Fig. 3. The logic program of the PacMan-Maze application in Scallop.



# Today's lecture

## symbolic expression

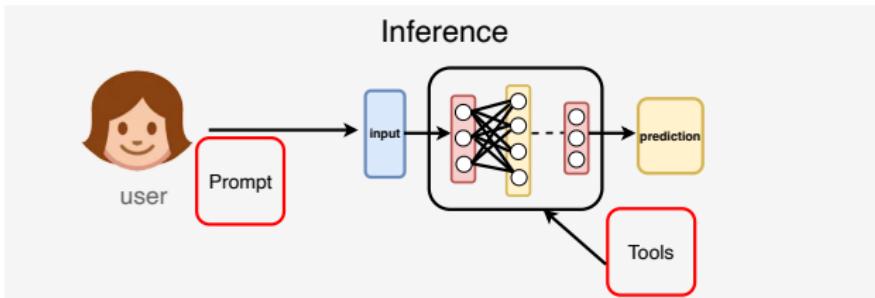


- ▶ Saying what we want the model to do (and what it can do), without saying how (applied to test-time inference).

# How does this work technically?

```
1 import problog as p          ## pip install problog
2 PROGRAM = p.program.PrologString("""
3 0.8::stress(ann).
4 0.4::stress(bob).
5 0.6::influences(ann,bob).
6 0.2::influences(bob,carl).
7
8 smokes(X) :- stress(X).
9 smokes(X) :- influences(Y,X), smokes(Y).
10
11 query(smokes(carl))."""
12 p.get_evaluatable().create_from(PROGRAM).evaluate()
13 ## => {smokes(carl): 0.1376000000000003}
```

# Goals for lecture today



- ▶ Extend our declarative approach from yesterday to **test-time inference** problems, model consistency.
- ▶ Introduce a SAT-based algorithm for MPE inference, circuits, logic programming.
- ▶ **next lecture:** Explore LLM prompting, imperative style programming.

Questions about last lecture?

## A motivating example

## Another mistake involving geography

**USER**

Which country is physically bigger, Germany or Poland? Please answer one word.

**ASSISTANT**

Poland

## Another mistake involving geography

**USER** Which country is physically bigger, Germany or Poland? Please answer one word.

**ASSISTANT** Poland

**USER** Is Poland physically bigger than Germany? Please answer only yes/no

**ASSISTANT** No.

# Querying ChatGPT

```
1 COUNTRIES = ["Germany", "France", "Poland"]
2 PROMPT_1 = f"Which country is physically bigger" +\
3 "{c1} or {c2}? Please provide a one word answer."
4 PROMPT_2 = f"Is {c2} physically bigger than" +\
5 "{c1}, answer only yes/no."
6
7 for c1 in COUNTRIES:
8     for c2 in [c for c in COUNTRIES if c != c1]:
9         prompt_1 = PROMPT_1.format(c1=c1,c2=c2)
10        answer1 = query_chatgpt(prompt_1)
11        prompt_1 = PROMPT_2.format(c1=c1,c2=c2)
12        answer2 = query_chatgpt(prompt_1)
```

# Querying ChatGPT

```
1  from openai import OpenAI
2  client = OpenAI() #<---- requires API key
3
4  def query_chatgpt(prompt):
5      response = client.chat.completions.create(
6          model="gpt-3.5-turbo",
7          messages = [{{
8              "role": "user",
9              "content" : [
10                 {
11                     "type" : "text",
12                     "text" : prompt
13                 }],}}],
14          temperature=0., logprobs=True
15      )
16      out_text = response.choices[0].message.content
17      prob = math.exp(
18          response.choices[0].logprobs.content[0].logprob
19      )
20      return (out_text,prob)
```

# Querying ChatGPT

```
1 COUNTRIES = ["Germany", "France", "Poland"]
2 PROMPT_1 = f"Which country is physically bigger" +\
3 "{c1} or {c2}? Please provide a one word answer."
4 PROMPT_2 = f"Is {c2} physically bigger than" +\
5 "{c1},answer only yes/no."
```

# Querying ChatGPT

```
1 | COUNTRIES = ["Germany", "France", "Poland"]
2 | PROMPT_1 = f"Which country is physically bigger" +\
3 | "{c1} or {c2}? Please provide a one word answer."
4 | PROMPT_2 = f"Is {c2} physically bigger than" +\
5 | "{c1},answer only yes/no."
```

## ► What it returns (gpt-3.5-turbo)

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	invariance?
(DE,FR)	FR (c2)	No	no
(DE,PL)	PL (c2)	No	no
(FR,DE)	DE (c2)	No	no
(FR,PL)	PL (c2)	No	no
(PL,DE)	DE (c2)	No	no
(PL,FR)	FR (c2)	No	no

# Querying ChatGPT

```
1 COUNTRIES = ["Germany", "France", "Poland"]
2 PROMPT_1 = f"Which country is physically bigger" +\
3 "{c1} or {c2}? Please provide a one word answer."
4 PROMPT_2 = f"Is {c2} physically bigger than" +\
5 "{c1}, answer only yes/no."
```

- ▶ What it returns (gpt-3.5-turbo) updated: Aug. 5 2024

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	invariance?
(DE,FR)	FR (c2)	No	no
(DE,PL)	DE (c1)	No	yes
(FR,DE)	FR (c1)	No	yes
(FR,PL)	FR (c1)	No	yes
(PL,DE)	DE (c2)	No	no
(PL,FR)	FR (c2)	No	no

# Querying ChatGPT

```
1 COUNTRIES = ["Germany", "France", "Poland"]
2 PROMPT_1 = f"Which country is physically bigger" +\
3 "{c1} or {c2}? Please provide a one word answer."
4 PROMPT_2 = f"Is {c2} physically bigger than" +\
5 "{c1}, answer only yes/no."
```

- ▶ What it returns (gpt-3.5-turbo) updated: July 29 2025

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	invariance?
(DE,FR)	FR (c2)	No	no
(DE,PL)	DE (c1)	No	yes
(FR,DE)	FR (c1)	No	yes
(FR,PL)	FR (c1)	No	yes
(PL,DE)	DE (c2)	No	no
(PL,FR)	FR (c2)	No	no

## More advanced prompting

- ▶ **Caveats:** very sensitive to prompt, order, not best model or prompting strategy, **be skeptical.**

## More advanced prompting

- ▶ **Caveats:** very sensitive to prompt, order, not best model or prompting strategy, **be skeptical**.

**USER** Which country is physically bigger, Germany or Poland? Please explain step by step.

**ASSISTANT** To determine which country is physically bigger, we can compare their respective land areas.

1. Germany: The total land area of Germany is approximately 357,022 square kilometers.

2. Poland: The total land area of Poland is approximately 312,696 square kilometers.

By comparing these numbers, we can see that Germany has a larger land area than Poland. Specifically, Germany is around 44,326 square kilometers bigger than Poland. Therefore, Germany is physically bigger than Poland.

# More advanced prompting

- ▶ **Caveats:** very sensitive to prompt, order, not best model or prompting strategy, **be skeptical.**

USER      Which country is physically bigger, Germany or Poland? Please explain step by step.

ASSISTANT    To determine which country is physically bigger, we can compare their respective land areas.

## Will discuss prompting tomorrow

1. Germany: The total land area of Germany is approximately 357,022 square kilometers.

2. Poland: The total land area of Poland is approximately 312,696 square kilometers.

By comparing these numbers, we can see that Germany has a larger land area than Poland. Specifically, Germany is around 44,326 square kilometers bigger than Poland. Therefore, Germany is physically bigger than Poland.

## How well do other general purpose models do?

```
1  from transformers import AutoTokenizer as tokenizer,\n2      AutoModelForCausallLM as model\n3  ## instruct tuned models, SFT + DPO\n4  mname = "HuggingFaceTB/SmollM2-135M-Instruct"\n5  tokenizer = tokenizer.from_pretrained(mname)\n6  smollm = model.from_pretrained(mname)\n7  messages = [{\n8      "role": "user",\n9      "content": "Which is bigger, Germany or France?"\n10  }]\n11  inputs = tokenizer.apply_chat_template(\n12      messages,\n13      add_generation_prompt=True,\n14      tokenize=True,\n15      return_dict=True,\n16      return_tensors="pt",\n17  ).to(model.device)\n18\n19  outputs = smollm.generate(**inputs)\n20  # how to get output? probabilities?
```

What we want a model to do (semantics)

## Thinking about semantics again

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	<i>invariance?</i>
(DE,FR)	FR (c2)	No	<b>no</b>
(DE,PL)	DE (c1)	No	<b>yes</b>
(FR,DE)	FR (c1)	No	<b>yes</b>
(FR,PL)	FR (c1)	No	<b>yes</b>
(PL,DE)	DE (c2)	No	<b>no</b>
(PL,FR)	FR (c2)	No	<b>no</b>

# Thinking about semantics again

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	invariance?
(DE,FR)	FR (c2)	No	no
(DE,PL)	DE (c1)	No	yes
(FR,DE)	FR (c1)	No	yes
(FR,PL)	FR (c1)	No	yes
(PL,DE)	DE (c2)	No	no
(PL,FR)	FR (c2)	No	no

- ▶ Invariance constraints.

**C1**(c1, c2), **C2**(c1, c2), **Y**(c2, c1), **N**(c2, c1)

first: *c1/c2 is bigger*; second: *c2 is bigger than c1, yes/no.*

# Thinking about semantics again

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	invariance?
(DE,FR)	FR (c2)	No	no
(DE,PL)	DE (c1)	No	yes
(FR,DE)	FR (c1)	No	yes
(FR,PL)	FR (c1)	No	yes
(PL,DE)	DE (c2)	No	no
(PL,FR)	FR (c2)	No	no

## ► Invariance constraints.

$$\mathbf{C1}(c1, c2), \mathbf{C2}(c1, c2), \mathbf{Y}(c2, c1), \mathbf{N}(c2, c1)$$

first:  $c1/c2$  is bigger; second:  $c2$  is bigger than  $c1$ , yes/no.

$$\forall(c1, c2).$$

$$(\mathbf{C2}(c1, c2) \leftrightarrow \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \leftrightarrow \mathbf{N}(c2, c1))$$

# Thinking about semantics again

## ► Invariance constraints.

$$\mathbf{C1}(c1, c2), \mathbf{C2}(c1, c2), \mathbf{Y}(c2, c1), \mathbf{N}(c2, c1)$$

first:  $c1/c2$  is bigger; second:  $c2$  is bigger than  $c1$ , yes/no.

$$\forall(c1, c2).$$

$$((\mathbf{C2}(c1, c2) \leftrightarrow \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \leftrightarrow \mathbf{N}(c2, c1)))$$

## ► Order constraints:

$$\forall(c1, c2).$$

$$((\mathbf{C2}(c1, c2) \rightarrow \neg \mathbf{C2}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \rightarrow \neg \mathbf{C1}(c2, c1)) \wedge \\ (\mathbf{Y}(c1, c2) \rightarrow \neg \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{N}(c1, c2) \rightarrow \neg \mathbf{N}(c2, c1)))$$

# Thinking about semantics again

## ► Invariance constraints.

$$\mathbf{C1}(c1, c2), \mathbf{C2}(c1, c2), \mathbf{Y}(c2, c1), \mathbf{N}(c2, c1)$$

first:  $c1/c2$  is bigger; second:  $c2$  is bigger than  $c1$ , yes/no.

$$\forall(c1, c2).$$

$$((\mathbf{C2}(c1, c2) \leftrightarrow \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \leftrightarrow \mathbf{N}(c2, c1)))$$

## ► Order constraints:

$$\forall(c1, c2).$$

$$((\mathbf{C2}(c1, c2) \rightarrow \neg \mathbf{C2}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \rightarrow \neg \mathbf{C1}(c2, c1)) \wedge \\ (\mathbf{Y}(c1, c2) \rightarrow \neg \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{N}(c1, c2) \rightarrow \neg \mathbf{N}(c2, c1)))$$

**Not exhaustive:** label uniqueness, transitivity constraints, etc.

# Thinking about semantics again

## ► Invariance constraints.

$$\mathbf{C1}(c1, c2), \mathbf{C2}(c1, c2), \mathbf{Y}(c2, c1), \mathbf{N}(c2, c1)$$

first:  $c1/c2$  is bigger; second:  $c2$  is bigger than  $c1$ , yes/no.

$$\forall(c1, c2).$$

$$((\mathbf{C2}(c1, c2) \leftrightarrow \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \leftrightarrow \mathbf{N}(c2, c1)))$$

## ► Order constraints:

$$\forall(c1, c2).$$

$$((\mathbf{C2}(c1, c2) \rightarrow \neg \mathbf{C2}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \rightarrow \neg \mathbf{C1}(c2, c1)) \wedge \\ (\mathbf{Y}(c1, c2) \rightarrow \neg \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{N}(c1, c2) \rightarrow \neg \mathbf{N}(c2, c1)))$$

**Observation:** even (seemingly) simple problems have a rich semantics.

# Thinking about semantics again

- ▶ Taking the simple case of just  $\text{C1}(c1, c2)$ ,  $\text{C2}(c1, c2)$ ,  $\text{Y}(c2, c1)$ ,  $\text{N}(c2, c1)$ , we can again think in terms of our *possible world* semantics:

worlds	
w <sub>1</sub>	$\text{C1}(c1, c2), \neg\text{C2}(c1, c2), \text{Y}(c2, c1), \neg\text{N}(c2, c1)$
w <sub>2</sub>	$\text{C1}(c1, c2), \neg\text{C2}(c1, c2), \neg\text{Y}(c2, c1), \text{N}(c2, c1)$
w <sub>3</sub>	$\neg\text{C1}(c1, c2), \text{C2}(c1, c2), \text{Y}(c2, c1), \neg\text{N}(c2, c1)$
w <sub>4</sub>	$\neg\text{C1}(c1, c2), \text{C2}(c1, c2), \neg\text{Y}(c2, c1), \text{N}(c2, c1)$

What kind of problems do we want to solve?

# Thinking about semantics again

- ▶ Taking the simple case of just  $\text{C1}(c1, c2)$ ,  $\text{C2}(c1, c2)$ ,  $\text{Y}(c2, c1)$ ,  $\text{N}(c2, c1)$ , we can again think in terms of our *possible world* semantics:

worlds	
w <sub>1</sub>	$\text{C1}(c1, c2), \neg\text{C2}(c1, c2), \text{Y}(c2, c1), \neg\text{N}(c2, c1)$
w <sub>2</sub>	$\text{C1}(c1, c2), \neg\text{C2}(c1, c2), \neg\text{Y}(c2, c1), \text{N}(c2, c1)$
w <sub>3</sub>	$\neg\text{C1}(c1, c2), \text{C2}(c1, c2), \text{Y}(c2, c1), \neg\text{N}(c2, c1)$
w <sub>4</sub>	$\neg\text{C1}(c1, c2), \text{C2}(c1, c2), \neg\text{Y}(c2, c1), \text{N}(c2, c1)$

What kind of problems do we want to solve?

- ▶ **Model testing:** Does my model satisfy a particular set of constraints? behavioral testing.

# Thinking about semantics again

- ▶ Taking the simple case of just  $\text{C1}(c1, c2)$ ,  $\text{C2}(c1, c2)$ ,  $\text{Y}(c2, c1)$ ,  $\text{N}(c2, c1)$ , we can again think in terms of our *possible world* semantics:

worlds	
w <sub>1</sub>	$\text{C1}(c1, c2), \neg\text{C2}(c1, c2), \text{Y}(c2, c1), \neg\text{N}(c2, c1)$
w <sub>2</sub>	$\text{C1}(c1, c2), \neg\text{C2}(c1, c2), \neg\text{Y}(c2, c1), \text{N}(c2, c1)$
w <sub>3</sub>	$\neg\text{C1}(c1, c2), \text{C2}(c1, c2), \text{Y}(c2, c1), \neg\text{N}(c2, c1)$
w <sub>4</sub>	$\neg\text{C1}(c1, c2), \text{C2}(c1, c2), \neg\text{Y}(c2, c1), \text{N}(c2, c1)$

What kind of problems do we want to solve?

- ▶ **Model testing:** Does my model satisfy a particular set of constraints? behavioral testing.
- ▶ **Model correction:** Can I use constraints to fix errors, arrive at new predictions?

# Toward probabilistic reasoning over predictions

- ▶ Extracting the token probabilities from chatGPT.

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	<i>invariance?</i>
(DE,FR)	FR (c2) 0.752	No 0.957	<b>no</b>
(DE,PL)	DE (c1) 0.986	No 0.990	<b>yes</b>
(FR,DE)	FR (c1) 0.575	No 0.898	<b>yes</b>
(FR,PL)	FR (c1) 0.956	No 0.991	<b>yes</b>
(PL,DE)	DE (c2) 0.997	No 0.654	<b>no</b>
(PL,FR)	FR (c2) 0.998	No 0.882	<b>no</b>

# Toward probabilistic reasoning over predictions

- ▶ Extracting the token probabilities from chatGPT.

(c1,c2)	bigger, c1 or c2?	c2 bigger c1?	invariance?
(DE,FR)	FR (c2) 0.752	No 0.957	no
(DE,PL)	DE (c1) 0.986	No 0.990	yes
(FR,DE)	FR (c1) 0.575	No 0.898	yes
(FR,PL)	FR (c1) 0.956	No 0.991	yes
(PL,DE)	DE (c2) 0.997	No 0.654	no
(PL,FR)	FR (c2) 0.998	No 0.882	no

**Main topic:** Can we improve consistency through inference time probabilistic reasoning over model outputs?

# Getting probabilities from models

- ▶ Task specific encoder models

```
1 ## run forward pass
2 loss,probabilities = nli_model({
3     "input_ids": batch_data.input_ids,
4     "labels"   : labels,
5 })
6 print(probabilities)
7 #####tensor([[0.3339, 0.3034, 0.3627]], ....)
```

- ▶ Decoder only models

```
1 outputs = smollm.generate(**inputs)
2 # compute over token probabilities
```

## Probabilistic reasoning over predictions

Interpreting predictions using a simpler  
probabilistic model

# What kind of inference do we want to do?

symbolic expression

```
Implies(  
    M1(x1, l1),  
    Not(M2(x2, l2))  
)  
Biconditional(  
    M1(x1, l1), M2(x2, l1)  
)  
M1(x3, l2)  
....
```

compilation

Inference  
algorithm

## Graphical models: defining what we want to do

- ▶ **Probabilistic graphical models (PGM)**: models of complex joint probability distributions expressed in terms of graphs. E.g.,

$$p(\mathbf{A}, \mathbf{B})$$

(over two random variables  $\mathbf{A}, \mathbf{B}$ ) we might have the *directed* graph:

## Graphical models: defining what we want to do

- ▶ **Probabilistic graphical models (PGM)**: models of complex joint probability distributions expressed in terms of graphs. E.g.,

$$p(\mathbf{A}, \mathbf{B})$$

(over two random variables  $\mathbf{A}, \mathbf{B}$ ) we might have the *directed* graph:



# Graphical models: defining what we want to do

- ▶ **Probabilistic graphical models (PGM)**: models of complex joint probability distributions expressed in terms of graphs. E.g.,

$$p(\mathbf{A}, \mathbf{B})$$

(over two random variables  $\mathbf{A}, \mathbf{B}$ ) we might have the *directed* graph:



**Bayesian network**: a classic type of PGM, takes the form:

$$p(\mathbf{X}_1, \dots, \mathbf{X}_n) = \prod_{i=1}^1 p(\mathbf{X}_i \mid \text{PA}(\mathbf{X}_i))$$

# Graphical models: defining what we want to do

For  $p(\mathbf{A}, \mathbf{B})$  we have the graph:



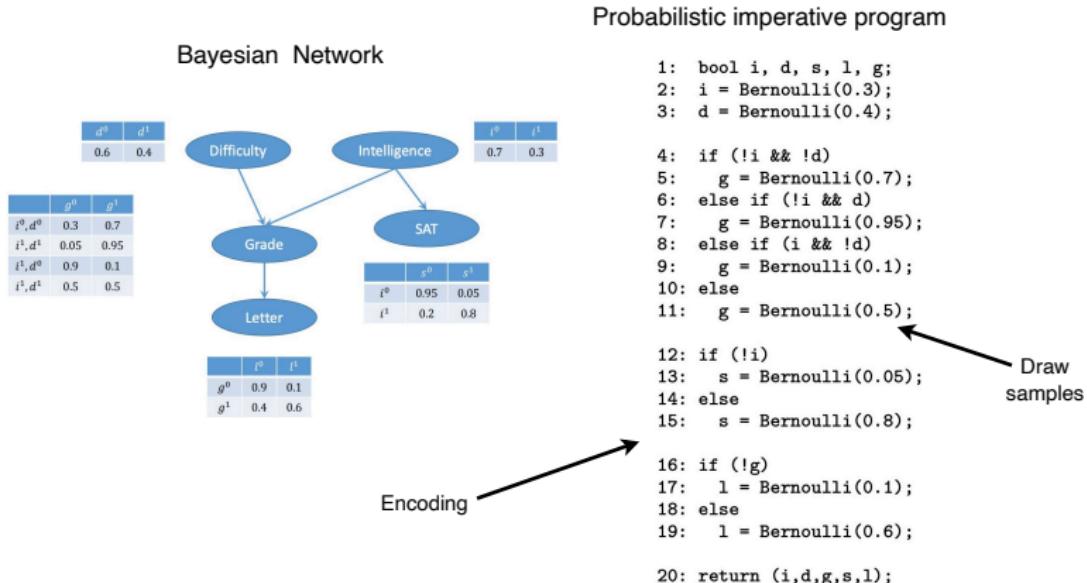
**Bayesian networks:** a classic type of PGM, take the form:

$$p(\mathbf{X}_1, \dots, \mathbf{X}_n) = \prod_{i=1}^n p(\mathbf{X}_i \mid \text{PA}(\mathbf{X}_i))$$

- ▶ **Directed graphical model**, involves having the following, *conditional probability tables (CPTs)* (example from [Park \(2002\)](#)):

A	$p(\mathbf{A})$	A	B	$p(\mathbf{B} \mid \mathbf{A})$
$a_1$	.3	$a_1$	$b_1$	.2
$a_2$	.5	$a_1$	$b_2$	.8
$a_3$	.2	$a_2$	$b_1$	.7
		$a_2$	$b_2$	.3
		$a_3$	$b_1$	.6
		$a_3$	$b_2$	.4

# Other encodings of Bayesian networks



from Gordon et al. (2014)

Some proposals about framing LLM prompting in terms of probabilistic (imperative) programming (Dohan et al., 2022).

# Other encodings of Bayesian networks



Probabilistic imperative program

```
1: bool i, d, s, l, g;
2: i = Bernoulli(0.3);
3: d = Bernoulli(0.4);

4: if (!i && !d)
5:   g = Bernoulli(0.7);
6: else if (!i && d)
7:   g = Bernoulli(0.95);
8: else if (i && !d)
9:   g = Bernoulli(0.1);
10: else
11:   g = Bernoulli(0.6); Draw samples

12: if (!g)
13:   s = Bernoulli(0.05);
14: else
15:   s = Bernoulli(0.95);

16: if (!s)
17:   l = Bernoulli(0.1);
18: else
19:   l = Bernoulli(0.6);

20: return (i,d,g,s,l);
```

Will discuss probabilistic programming tomorrow

Encoding

from Gordon et al. (2014)

Some proposals about framing LLM prompting in terms of probabilistic (imperative) programming (Dohan et al., 2022).

# Graphical models: defining what we want to do

**Bayesian networks:** a classic type of PGM, take the form:

$$p(\mathbf{X}_1, \dots, \mathbf{X}_n) = \prod_{i=1}^1 p(\mathbf{X}_i \mid \text{PA}(\mathbf{X}_i))$$

- ▶ **Directed graphical model**, involves having the following, *conditional probability tables (CPTs)*:

A	p(A)	A	B	$p(B \mid A)$
$a_1$	.3	$a_1$	$b_1$	.2
$a_2$	.5	$a_1$	$b_2$	.8
$a_3$	.2	$a_2$	$b_1$	.7
		$a_2$	$b_2$	.3
		$a_3$	$b_1$	.6
		$a_3$	$b_2$	.4

- ▶ The particular **inference problem** we will focus on:

**Most Probable Explanation (MPE):** *Finding the most probable complete variable assignment (given evidence).*

# MPE Inference and MAXSAT

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$p(\mathbf{A})$	$\mathbf{A}$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$a_1$	.3	$a_1$	$b_1$	.2
$a_2$	.5	$a_1$	$b_2$	.8
$a_3$	.2	$a_2$	$b_1$	.7
		$a_2$	$b_2$	.3
		$a_3$	$b_1$	.6
		$a_3$	$b_2$	.4

**Most Probable Explanation (MPE):** *Finding the highest probability variable assignment (given evidence), NP-complete*

# MPE Inference and MAXSAT

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$p(\mathbf{A})$	$\mathbf{A}$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$a_1$	.3	$a_1$	$b_1$	.2
$a_2$	.5	$a_1$	$b_2$	.8
$a_3$	.2	$a_2$	$b_1$	.7
		$a_2$	$b_2$	.3
		$a_3$	$b_1$	.6
		$a_3$	$b_2$	.4

**Most Probable Explanation (MPE):** *Finding the highest probability variable assignment (given evidence), NP-complete*

# MPE Inference and MAXSAT

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$p(\mathbf{A})$	$\mathbf{A}$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$a_1$	.3	$a_1$	$b_1$	.2
$a_2$	.5	$a_1$	$b_2$	.8
$a_3$	.2	$a_2$	$b_1$	.7
		$a_2$	$b_2$	.3
		$a_3$	$b_1$	.6
		$a_3$	$b_2$	.4

**Most Probable Explanation (MPE):** *Finding the highest probability variable assignment (given evidence), NP-complete*

- ▶ Often solved via SAT-based techniques ([Park, 2002](#)), variant:

# MPE Inference and MAXSAT

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$p(\mathbf{A})$	$\mathbf{A}$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$a_1$	.3	$a_1$	$b_1$	.2
$a_2$	.5	$a_1$	$b_2$	.8
$a_3$	.2	$a_2$	$b_1$	.7
		$a_2$	$b_2$	.3
		$a_3$	$b_1$	.6
		$a_3$	$b_2$	.4

**Most Probable Explanation (MPE):** *Finding the highest probability variable assignment (given evidence), NP-complete*

- ▶ Often solved via SAT-based techniques ([Park, 2002](#)), variant:

**MAXSAT:** Finding the maximum number of clauses of a formula that can be made true, common in some NLP work on LLMs.

SAT and related problems (slight detour)

## Boolean satisfiability

- ▶ Given a formula  $K$ , does there exist an interpretation that makes the formula true? E.g.,

$$K := (\textcolor{blue}{E} \wedge \neg \textcolor{red}{C} \wedge \neg \textcolor{blue}{N}) \vee (\neg \textcolor{blue}{E} \wedge \textcolor{red}{C} \wedge \neg \textcolor{blue}{N}) \vee (\neg \textcolor{blue}{E} \wedge \neg \textcolor{red}{C} \wedge \textcolor{blue}{N})$$

## Boolean satisfiability

- ▶ Given a formula  $K$ , does there exist an interpretation that makes the formula true? E.g.,

$$K := (\text{E} \wedge \neg \text{C} \wedge \neg \text{N}) \vee (\neg \text{E} \wedge \text{C} \wedge \neg \text{N}) \vee (\neg \text{E} \wedge \neg \text{C} \wedge \text{N})$$

**satisfying assignment:**  $\text{E}$  is true,  $\text{C}$  is false,  $\text{N}$  is false.

# Using propositional SAT solvers

```
1 ## to install: pip install z3-solver
2 from z3 import *
3
4 solver = Solver()
5 P,Q,R,S = Bools("P Q R S")
6
7 ### [((P^Q)→R)→S), formula, adds to solver
8 Gamma = [ Implies(Implies(And(P,Q),R),S) ]
9 solver.add(Gamma)
10
11 ## check if satisfiable
12 solver.check() ## => "sat",
13 ## finds a satisfying model
14 solver.model() ## => [S = True, R = False, ...]
```

# Boolean satisfiability and model counting

- Given a formula  $K$ , does there exist an interpretation that makes the formula true? E.g.,

$$K := (\textcolor{blue}{E} \wedge \neg \textcolor{blue}{C} \wedge \neg \textcolor{blue}{N}) \vee (\neg \textcolor{blue}{E} \wedge \textcolor{blue}{C} \wedge \neg \textcolor{blue}{N}) \vee (\neg \textcolor{blue}{E} \wedge \neg \textcolor{blue}{C} \wedge \textcolor{blue}{N})$$

**satisfying assignment:**  $\textcolor{blue}{E}$  is true,  $\textcolor{blue}{C}$  is false,  $\textcolor{blue}{N}$  is false.

Model counting is a variation of this, how many satisfying assignments does a formula have? weighted variant

$$\text{WMC}(K, \theta) := \sum_{\mathbf{w} \models K} \prod_{\mathbf{w} \models X_i} P_\theta(X_i) \cdot \prod_{\mathbf{w} \models \neg X_i} (1 - P_\theta(X_i))$$

# Boolean satisfiability and model counting

- Given a formula  $K$ , does there exist an interpretation that makes the formula true? E.g.,

$$K := (\textcolor{blue}{E} \wedge \neg \textcolor{blue}{C} \wedge \neg \textcolor{blue}{N}) \vee (\neg \textcolor{blue}{E} \wedge \textcolor{blue}{C} \wedge \neg \textcolor{blue}{N}) \vee (\neg \textcolor{blue}{E} \wedge \neg \textcolor{blue}{C} \wedge \textcolor{blue}{N})$$

**satisfying assignment:**  $\textcolor{blue}{E}$  is true,  $\textcolor{blue}{C}$  is false,  $\textcolor{blue}{N}$  is false.

Model counting is a variation of this, how many satisfying assignments does a formula have? weighted variant

$$\text{WMC}(K, \theta) := \sum_{\mathbf{w} \models K} \prod_{\mathbf{w} \models X_i} P_\theta(X_i) \cdot \prod_{\mathbf{w} \models \neg X_i} (1 - P_\theta(X_i))$$

Our probabilistic model from the last lecture can be expressed as a Bayesian network, but a simple one.

## Generalizing model counting

- ▶ Model counting is a variation of this, how many satisfying assignments does a formula have? Notation change,  $l$  (literal,  $X_i$  or  $\neg X_i$ )

$$\text{WMC}(K, \theta) := \sum_{w \models K} \prod_{w \models l} P_\theta(l).$$

## Generalizing model counting

- ▶ Model counting is a variation of this, how many satisfying assignments does a formula have? Notation change,  $l$  (literal,  $X_i$  or  $\neg X_i$ )

$$\text{WMC}(K, \theta) := \sum_{w \models K} \prod_{w \models l} P_\theta(l).$$

Generalize the symbols  $\sum$ ,  $\prod$  and  $P_\theta$  ([Kimmig et al., 2011, 2017](#)):

$$\text{AMC}(K) := \bigoplus_{w \models K} \bigotimes_{w \models l} a(l)$$

# Generalizing model counting

- Model counting is a variation of this, how many satisfying assignments does a formula have? Notation change,  $l$  (literal,  $X_i$  or  $\neg X_i$ )

$$\text{WMC}(K, \theta) := \sum_{w \models K} \prod_{w \models l} P_\theta(l).$$

Generalize the symbols  $\sum$ ,  $\prod$  and  $P_\theta$  (Kimmig et al., 2011, 2017):

$$\text{AMC}(K) := \bigoplus_{w \models K} \bigotimes_{w \models l} a(l)$$

Task	$A$	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$	$a(l)$	$a(\neg l)$
SAT	{true, false}	$\vee$	$\wedge$	false	true	true	true
#SAT (MC)	$\mathbb{N}$	$+$	$\times$	0	1	1	1
MAXSAT	$\mathbb{N}$	max	$\times$	0	1	1	1
WMC	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
WMAXSAT	$\mathbb{R}_{\geq 0}$	max	$\times$	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
PROB	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1	$\in [0, 1]$	$1 - a(l)$
MPE	$\mathbb{R}_{\geq 0}$	max	$\times$	0	1	$\in [0, 1]$	$1 - a(l)$
FUZZY	[0,1]	max	min	0	1	$\in [0, 1]$	$1 - a(l)$

Algebraic structure, semiring  $(A, \oplus, \otimes, \bar{0}, \bar{1})$ .

# Generalizing model counting

---

**Algorithm 2:** General search algorithm for commutative semirings

---

Generic  
algorithm

```

input: A CNF formula  $\Delta$ , variables  $V$ 
1 Function  $\text{SS}(\Delta, V)$ :
  2 if  $\{\} \in \Delta$  then
    3 return  $\bar{0}$ 
  4 else if  $\Delta = \{\}$  then
    5 if  $V$  is not empty then
      6 return  $\bigotimes_{v \in V} (a(v) \oplus a(\neg v))$ 
    7 return  $\bar{1}$ 
  8 select literal  $l$  from  $\Delta$  ;
  9 let  $V'$  be  $V$  with variable in  $l$  removed ;
  10 return  $(a(l) \otimes \text{SS}(\Delta|l, V')) \oplus (a(\neg l) \otimes \text{SS}(\Delta|\neg l, V'))$  ;

```

---

Task	$A$	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$	$a(l)$	$a(\neg l)$
SAT	{true, false}	$\vee$	$\wedge$	false	true	true	true
#SAT (MC)	$\mathbb{N}$	$+$	$\times$	0	1	1	1
MAXSAT	$\mathbb{N}$	max	$\times$	0	1	1	1
WMC	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
WMAXSAT	$\mathbb{R}_{\geq 0}$	max	$\times$	0	1	$\in \mathbb{R}_{\geq 0}$	$\in \mathbb{R}_{\geq 0}$
PROB	$\mathbb{R}_{\geq 0}$	$+$	$\times$	0	1	$\in [0, 1]$	$1 - a(l)$
MPE	$\mathbb{R}_{\geq 0}$	max	$\times$	0	1	$\in [0, 1]$	$1 - a(l)$
FUZZY	[0,1]	max	min	0	1	$\in [0, 1]$	$1 - a(l)$

discrete

uncertain  
reasoning

# Generalizing model counting: An aside

## An aside: gradient semirings and parameter estimation

In the last example, we showed how to use weighted model counting to compute the *success probability* of the weighted CNF formula  $S(F)$  in Eq 3 given Boolean variables  $P_1, P_2, P_3$  and their weights  $p = [0.8, 0.75, 0.4] = p_1, p_2, p_3$ . Supposing that our goal is to maximize the success probability of  $F$  (where we take  $F$  to be our *evidence*), one question we might ask is: *is it possible to learn variable weights that best realize this goal?* Such is the problem of parameter estimation that underlies virtually all machine learning problems.

Gradient-based optimization techniques are often used for solving such problems and will play a central role in the next chapter. Broadly, gradient methods rely on computing mathematical gradients, which (*loosely*) tell us about the direction we should shift our weights to get closer to our goal. Formally, gradients take the form of a vector

$$\vec{g} = \left\langle \frac{\partial S(F)}{\partial p_i}, \dots, \frac{\partial S(F)}{\partial p_n} \right\rangle \quad (6)$$

consisting of partial derivatives  $\frac{\partial S(F)}{\partial p_j}$  for each of our parameters  $p_j$ . With such quantities, we can then employ algorithms such as (stochastic) gradient ascent or its variants to make updates to our weights and ultimately train our model against data.

Semirings have also been investigated for computing gradients, including the *gradient semiring* of [Eisner, 2002] (see also Kimmig et al. [2017], Manhaeve et al. [2018]). Such a semiring is defined below:

Task	$A$	$\oplus$	$\otimes$	$\mathbf{0}$	$\mathbf{1}$	$a(l_i)$	$a(\neg l_i)$
gradient	$\mathbb{R}_{\geq 0} \times \mathbb{R}$	Eq.8	Eq.9	$(0, \mathbf{0})$	$(0, \mathbf{1})$	Eq.7	Eq.7

Here the elements of  $A$  are now tuples consisting of probabilities  $p_i$  and their gradients  $\vec{g}_i$

$$\left( p_i, \vec{g}_i \right).$$

As with our other problems, arriving at the final solution to our problem (i.e., finding the vector in Eq.6) will involve working bottom-up from our literals and our labeling function  $a$ . In the base case, the initial gradients of *positive* literals (i.e., literals without negation)  $l$  will be defined as one-shot vectors  $\vec{e}_j$  consisting of a 1 in location  $j$  and 0s elsewhere. Our labeling function can then be defined as follows (for any input literal  $x_i$ , either positive or negative):

$$a(l_i) = \begin{cases} (p_i, \vec{e}_j), & \text{if } l_i \text{ is a positive literal} \\ (1 - p_i, -\vec{e}_i), & \text{otherwise} \end{cases} \quad (7)$$

Our addition and multiplication functions will then be defined as follows (with the last rule corresponding to the *chain rule*):

$$(p_1, \vec{g}_1) \oplus (p_2, \vec{g}_2) = (p_1 + p_2, \vec{g}_1 \cdot \vec{g}_2) \quad (8)$$

$$(p_1, \vec{g}_1) \otimes (p_2, \vec{g}_2) = (p_1 \times p_2, p_2 \vec{g}_1 \cdot p_1 \vec{g}_2) \quad (9)$$

## MAXSAT example

- ▶ **Maximum Satisfiability Problem (MAXSAT):** Finding the maximum number of clauses of a formula that can be made true.

## MAXSAT example

- ▶ **Maximum Satisfiability Problem (MAXSAT):** Finding the maximum number of clauses of a formula that can be made true.

**Example:** the following formula

$$K := (P_0 \vee P_1)_1 \wedge (P_0 \vee \neg P_1)_2 \wedge (\neg P_0 \vee P_1)_3 \wedge (\neg P_0 \vee \neg P_1)_4$$

does not have a satisfying assignment.

## MAXSAT example

- ▶ **Maximum Satisfiability Problem (MAXSAT)**: Finding the maximum number of clauses of a formula that can be made true.

**Example:** the following formula

$$K := (P_0 \vee P_1)_1 \wedge (P_0 \vee \neg P_1)_2 \wedge (\neg P_0 \vee P_1)_3 \wedge (\neg P_0 \vee \neg P_1)_4$$

does not have a satisfying assignment.

**However** It can satisfy the first three clauses, i.e.,

$P_0$  is true,  $P_1$  is true

## MAXSAT example

- ▶ **Maximum Satisfiability Problem (MAXSAT):** Finding the maximum number of clauses of a formula that can be made true.

**Example:** the following formula

$$K := (P_0 \vee P_1)_1 \wedge (P_0 \vee \neg P_1)_2 \wedge (\neg P_0 \vee P_1)_3 \wedge (\neg P_0 \vee \neg P_1)_4$$

does not have a satisfying assignment.

**However** It can satisfy the first three clauses, i.e.,

$P_0$  is true,  $P_1$  is true

**Cost:** 1 (*one violated clause*)

## MAXSAT example

- ▶ **Weighted MAXSAT:** Given clauses with weights (via some  $w$ ), find truth values that maximize the combined weights of satisfied clauses:

$$\text{maximize} \sum_{c \in \Delta_{\text{sat}}} w(c)$$

## MAXSAT example

- ▶ **Weighted MAXSAT:** Given clauses with weights (via some  $w$ ), find truth values that maximize the combined weights of satisfied clauses:

$$\text{maximize} \sum_{c \in \Delta_{\text{sat}}} w(c)$$

**Example:** the following formula

$$K := (P_0 \vee P_1)^9 \wedge (P_0 \vee \neg P_1)^{12} \wedge (\neg P_0 \vee P_1)^3 \wedge (\neg P_0 \vee \neg P_1)^1$$

## MAXSAT example

- ▶ **Weighted MAXSAT:** Given clauses with weights (via some  $w$ ), find truth values that maximize the combined weights of satisfied clauses:

$$\text{maximize} \sum_{c \in \Delta_{\text{sat}}} w(c)$$

**Example:** the following formula

$$K := (P_0 \vee P_1)^9 \wedge (P_0 \vee \neg P_1)^{12} \wedge (\neg P_0 \vee P_1)^3 \wedge (\neg P_0 \vee \neg P_1)^1$$

Weight is 24 and **cost** is 1.

# MAXSAT example

```
1 ## pip install python-sat
2 from pysat.examples.rc2 import RC2
3 from pysat.formula import WCNF
4
5 wcnf = WCNF()
6 ### variables
7 P0,P1 = [1,2]
8 ### formula clauses
9 wcnf.extend(
10     [[P0,P1],[P0,-P1],[-P0,P1],[-P0,-P1]],
11     weights=[9,12,3,1],
12 )
13 with RC2(wcnf) as rc2:
14     for m in rc2.enumerate():
15         print(f"model {m} has cost {rc2.cost}")
16 # model [1, 2] has cost 1
17 # model [1, -2] has cost 3
18 # model [-1, -2] has cost 9
19 # model [-1, 2] has cost 12
```

# MPE to MaxSAT

# The MPE to MAXSAT reduction

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$\mathbf{A}$	$p(\mathbf{A})$	
$a_1$	$b_1$	.2
$a_1$	$b_2$	.8
$a_2$	$b_1$	.7
$a_2$	$b_2$	.3
$a_3$	$b_1$	.6
$a_3$	$b_2$	.4

1. **Indicator clauses:** uniqueness constraints for  $\mathbf{A}$  and  $\mathbf{B}$ :

# The MPE to MAXSAT reduction

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$p(\mathbf{A})$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$a_1$	.3	$b_1$	.2
$a_1$	.3	$b_2$	.8
$a_2$	.5	$b_1$	.7
$a_2$	.5	$b_2$	.3
$a_3$	.2	$b_1$	.6
$a_3$	.2	$b_2$	.4

1. **Indicator clauses:** uniqueness constraints for  $\mathbf{A}$  and  $\mathbf{B}$ :

for  $\mathbf{A}$ :  $(I_{a_1} \vee I_{a_2} \vee I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_2}) \wedge (\neg I_{a_2} \vee \neg I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_3})$

for  $\mathbf{B}$ :  $(I_{b_1} \vee I_{b_2}) \wedge (\neg I_{b_1} \vee \neg I_{b_2})$

# The MPE to MAXSAT reduction

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$\mathbf{A}$	$p(\mathbf{A})$	
$a_1$	$b_1$	.2
$a_1$	$b_2$	.8
$a_2$	$b_1$	.7
$a_2$	$b_2$	.3
$a_3$	$b_1$	.6
$a_3$	$b_2$	.4

1. **Indicator clauses:** uniqueness constraints for  $\mathbf{A}$  and  $\mathbf{B}$ :

for  $\mathbf{A}$ :  $(I_{a_1} \vee I_{a_2} \vee I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_2}) \wedge (\neg I_{a_2} \vee \neg I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_3})$

for  $\mathbf{B}$ :  $(I_{b_1} \vee I_{b_2}) \wedge (\neg I_{b_1} \vee \neg I_{b_2})$

2. **Parameter clauses (units):** Negate and make clause weight  $-\log p$ :

$$(\neg I_{a_1})^{-\log(.3)} \wedge (\neg I_{a_2})^{-\log(.5)} \wedge (\neg I_{a_3})^{-\log(.2)}$$

# The MPE to MAXSAT reduction

For  $p(\mathbf{A}, \mathbf{B})$  we have the CPTs:

$\mathbf{A}$	$p(\mathbf{A})$	$\mathbf{B}$	$p(\mathbf{B}   \mathbf{A})$
$a_1$	.3	$b_1$	.2
$a_1$	.3	$b_2$	.8
$a_2$	.5	$b_1$	.7
$a_2$	.5	$b_2$	.3
$a_3$	.2	$b_1$	.6
$a_3$	.2	$b_2$	.4

1. **Indicator clauses:** uniqueness constraints for  $\mathbf{A}$  and  $\mathbf{B}$ :

for  $\mathbf{A}$ :  $(I_{a_1} \vee I_{a_2} \vee I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_2}) \wedge (\neg I_{a_2} \vee \neg I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_3})$

for  $\mathbf{B}$ :  $(I_{b_1} \vee I_{b_2}) \wedge (\neg I_{b_1} \vee \neg I_{b_2})$

2. **Parameter clauses (units):** Negate and make clause weight  $-\log p$ :

$$(\neg I_{a_1})^{-\log(.3)} \wedge (\neg I_{a_2})^{-\log(.5)} \wedge (\neg I_{a_3})^{-\log(.2)}$$

3. **Parameter clauses (non-units):** Negate each part in clause:

$$(\neg I_{a_1} \vee \neg I_{b_1})^{-\log(.2)} \wedge (\neg I_{a_1} \vee \neg I_{b_2})^{-\log(.8)} \wedge (\neg I_{a_2} \vee \neg I_{b_1})^{-\log(.7)} \wedge \dots$$

# The MPE to MAXSAT reduction: why this makes sense

1. **Indicator clauses:** uniqueness constraints for **A** and **B**:

for **A**:  $(I_{a_1} \vee I_{a_2} \vee I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_2}) \wedge (\neg I_{a_2} \vee \neg I_{a_3}) \wedge (\neg I_{a_1} \vee \neg I_{a_3})$

for **B**:  $(I_{b_1} \vee I_{b_2}) \wedge (\neg I_{b_1} \vee \neg I_{b_2})$

2. **Parameter clauses (units):** Negate and make clause weight  $-\log p$ :

$(\neg I_{a_1})^{-\log(.3)} \wedge (\neg I_{a_2})^{-\log(.5)} \wedge (\neg I_{a_3})^{-\log(.2)}$

3. **Parameter clauses (non-units):** Negate each part in clause:

$(\neg I_{a_1} \vee \neg I_{b_1})^{-\log(.2)} \wedge (\neg I_{a_1} \vee \neg I_{b_2})^{-\log(.8)} \wedge (\neg I_{a_2} \vee \neg I_{b_1})^{-\log(.7)} \wedge \dots$

**Example:** if we make  $I_{a_2}$  true, which has probability 0.5, then we violate  $(\neg I_{a_2})^{-\log(.5)}$  with a (relatively low) cost of 0.693

# The MPE to MAXSAT reduction: why this makes sense

1. **Indicator clauses:** uniqueness constraints for **A** and **B**:

for **A**:  $(I_{a_1} \vee I_{a_2} \vee I_{a_3})^\infty \wedge (\neg I_{a_1} \vee \neg I_{a_2})^\infty \wedge (\neg I_{a_2} \vee \neg I_{a_3})^\infty \wedge (\neg I_{a_1} \vee \neg I_{a_3})^\infty$   
for **B**:  $(I_{b_1} \vee I_{b_2})^\infty \wedge (\neg I_{b_1} \vee \neg I_{b_2})^\infty$

2. **Parameter clauses (units):** Negate and make clause weight  $-\log p$ :

$$(\neg I_{a_1})^{-\log(.3)} \wedge (\neg I_{a_2})^{-\log(.5)} \wedge (\neg I_{a_3})^{-\log(.2)}$$

3. **Parameter clauses (non-units):** Negate each part in clause:

$$(\neg I_{a_1} \vee \neg I_{b_1})^{-\log(.2)} \wedge (\neg I_{a_1} \vee \neg I_{b_2})^{-\log(.8)} \wedge (\neg I_{a_2} \vee \neg I_{b_1})^{-\log(.7)} \wedge \dots$$

**Infinite weight:** To enforce clauses that should always be true, we can add a weight  $\infty$  (operationally, make the weight of all clauses  $+ 1$ )

# The MPE to MAXSAT reduction: why this makes sense

1. **Indicator clauses:** uniqueness constraints for **A** and **B**:

for **A**:  $(I_{a_1} \vee I_{a_2} \vee I_{a_3})^\infty \wedge (\neg I_{a_1} \vee \neg I_{a_2})^\infty \wedge (\neg I_{a_2} \vee \neg I_{a_3})^\infty \wedge (\neg I_{a_1} \vee \neg I_{a_3})^\infty$   
for **B**:  $(I_{b_1} \vee I_{b_2})^\infty \wedge (\neg I_{b_1} \vee \neg I_{b_2})^\infty$

2. **Parameter clauses (units):** Negate and make clause weight  $-\log p$ :

$$(I_{a_1})^\infty \wedge (\neg I_{a_1})^{-\log(.2)} \wedge (\neg I_{a_2})^{-\log(.5)} \wedge (\neg I_{a_3})^{-\log(.2)}$$

3. **Parameter clauses (non-units):** Negate each part in clause:

$$(\neg I_{a_1} \vee \neg I_{b_1})^{-\log(.2)} \wedge (\neg I_{a_1} \vee \neg I_{b_2})^{-\log(.8)} \wedge (\neg I_{a_2} \vee \neg I_{b_1})^{-\log(.7)} \wedge \dots$$

**Evidence:** Do the same thing: add the evidence to the formula and weight with infinity.

## A MAXSAT reduction for our initial problem

- We have some **predictions**:  $C1(c1, c2)$ ,  $C2(c1, c2)$ ,  $Y(c2, c1)$ ,  $N(c2, c1)$ ,  
**probabilities**:  $\mathbf{p} = [p_{c1}, p_{c2}, p_Y, p_N]$

## A MAXSAT reduction for our initial problem

- ▶ We have some **predictions**:  $\mathbf{C1}(c_1, c_2)$ ,  $\mathbf{C2}(c_1, c_2)$ ,  $\mathbf{Y}(c_2, c_1)$ ,  $\mathbf{N}(c_2, c_1)$ ,  
**probabilities**:  $\mathbf{p} = [p_{c1}, p_{c2}, p_Y, p_N]$
1. **Hard constraints**: Translate all constraints into infinite weighted CNF clauses, i.e., constraints of the kind (translated to CNF):

$\forall(c_1, c_2).$

$$((\mathbf{C2}(c_1, c_2) \leftrightarrow \mathbf{Y}(c_2, c_1)) \wedge \\ (\mathbf{C1}(c_1, c_2) \leftrightarrow \mathbf{N}(c_2, c_1)))$$

# A MAXSAT reduction for our initial problem

- We have some **predictions**:  $\mathbf{C1}(c1, c2)$ ,  $\mathbf{C2}(c1, c2)$ ,  $\mathbf{Y}(c2, c1)$ ,  $\mathbf{N}(c2, c1)$ ,  
**probabilities**:  $\mathbf{p} = [p_{c1}, p_{c2}, p_Y, p_N]$

1. **Hard constraints**: Translate all constraints into infinite weighted CNF clauses, i.e., constraints of the kind (translated to CNF):

$$\forall(c1, c2).$$

$$((\mathbf{C2}(c1, c2) \leftrightarrow \mathbf{Y}(c2, c1)) \wedge \\ (\mathbf{C1}(c1, c2) \leftrightarrow \mathbf{N}(c2, c1)))$$

2. **Prediction units**: negate predictions and add  $-\log p$  weight, for example:

$$\neg \mathbf{C1}(c1, c2)^{-\log(p_{c1})} \wedge \neg \mathbf{C2}(c1, c2)^{-\log(p_{c2})} \wedge \neg \mathbf{Y}(c2, c1)^{-\log(p_Y)} \wedge \neg \mathbf{N}(c2, c1)^{\cdot\cdot}$$

## Hard vs. Soft clauses and constraints

```
1  from pysat.formula import WCNF
2
3  wcnf = WCNF()
4  ### variables
5  P0,P1 = [1,2]
6  ### formula clauses
7  wcnf.extend(
8      [[P0,P1],[P0,-P1],[-P0,P1],[-P0,-P1]],
9      weights=[9,12,3,1],
10 )
11
12 ### hard clause, evidence
13 wcnf.append([P1])
14 print(wcnf.hard)
```

# Related work in NLP

## Language Models with Rationality

Nora Kassner<sup>1,2</sup> Oyvind Tafjord<sup>1</sup> Ashish Sabharwal<sup>1</sup> Kyle Richardson<sup>1</sup>  
Hinrich Schütze<sup>2</sup> Peter Clark<sup>1</sup>

### Enhancing Self-Consistency and Performance of Pre-Trained Language Models through Natural Language Inference

Eric Mitchell, Joseph J. Noh, Siyan Li, William S. Armstrong,  
Ananth Agarwal, Patrick Liu, Chelsea Finn, Christopher D. Manning  
Stanford University  
[eric.mitchell@cs.stanford.edu](mailto:eric.mitchell@cs.stanford.edu)

### Ἁ Maieutic Prompting: Logically Consistent Reasoning with Recursive Explanations

Jaehun Jung<sup>†</sup> Lianhui Qin<sup>†</sup> Sean Welleck<sup>†‡</sup>  
Faeze Brahman<sup>†</sup> Chandra Bhagavatula<sup>†</sup> Ronan Le Bras<sup>‡</sup> Yejin Choi<sup>†‡</sup>

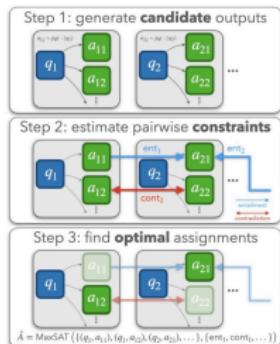
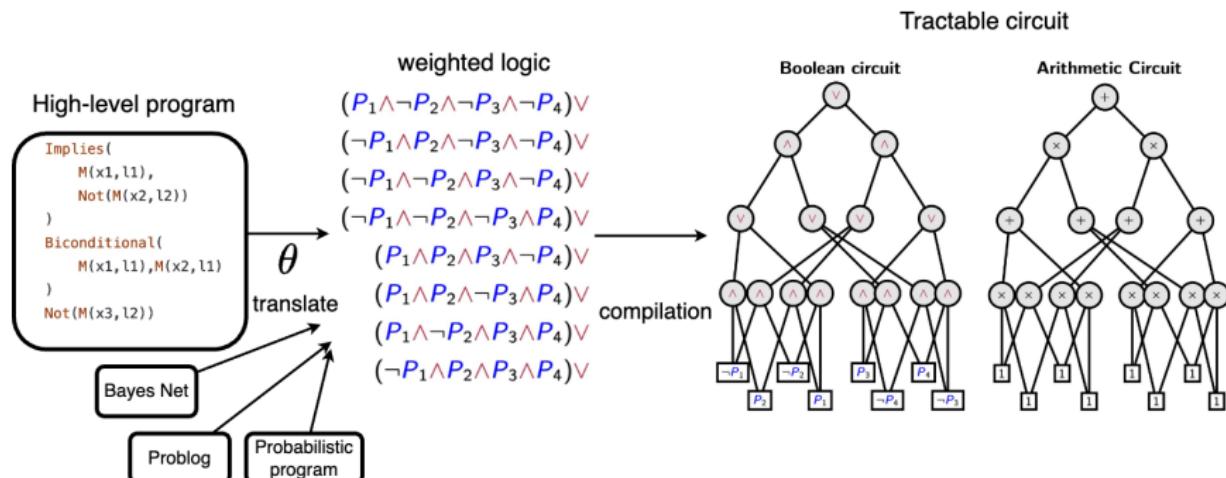


Figure 1: ConCoRD first generates candidate outputs from the base pre-trained model, then estimates soft pairwise constraints between output choices, and finally finds the most satisfactory choices of answers accounting for both the base model and NLI model's beliefs.

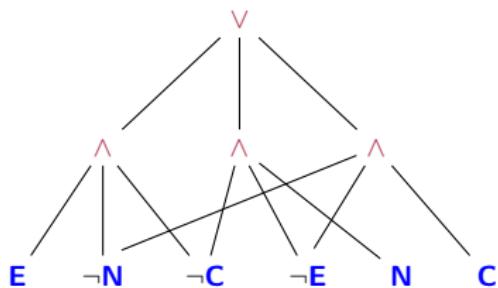
# Zooming out again: building high-level modeling languages



# Creating tractable representations: extending to MAXSAT

Uniqueness constraint again

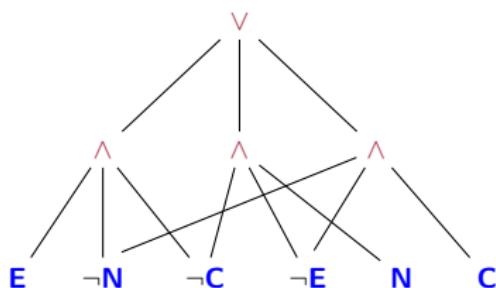
$$(\text{E} \wedge \neg \text{N} \wedge \neg \text{C}) \vee (\neg \text{E} \wedge \text{C} \wedge \neg \text{N}) \vee (\neg \text{E} \wedge \neg \text{C} \wedge \text{N})$$



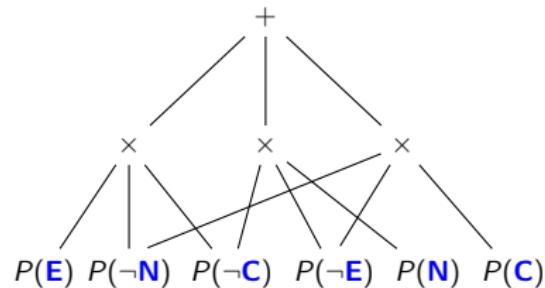
# Creating tractable representations: extending to MAXSAT

Uniqueness constraint again

$$(\text{E} \wedge \neg \text{N} \wedge \neg \text{C}) \vee (\neg \text{E} \wedge \text{C} \wedge \neg \text{N}) \vee (\neg \text{E} \wedge \neg \text{C} \wedge \text{N})$$



Boolean circuit

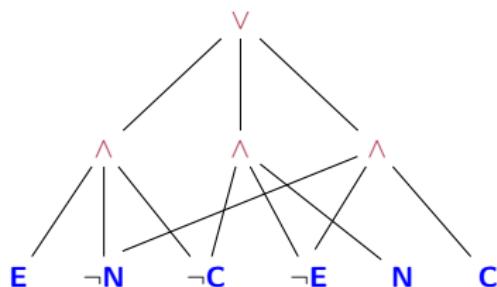


Arithmetic circuit

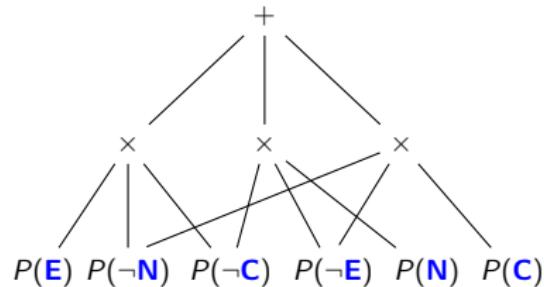
# Creating tractable representations: extending to MAXSAT

Uniqueness constraint again

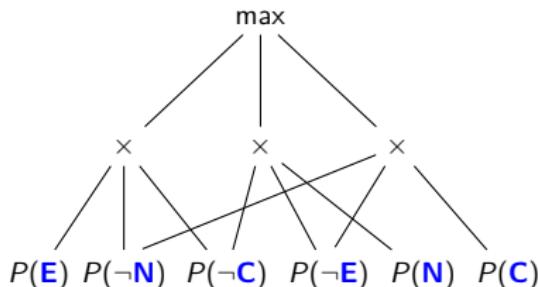
$$(\text{E} \wedge \neg \text{N} \wedge \neg \text{C}) \vee (\neg \text{E} \wedge \text{C} \wedge \neg \text{N}) \vee (\neg \text{E} \wedge \neg \text{C} \wedge \text{N})$$



**Boolean circuit**



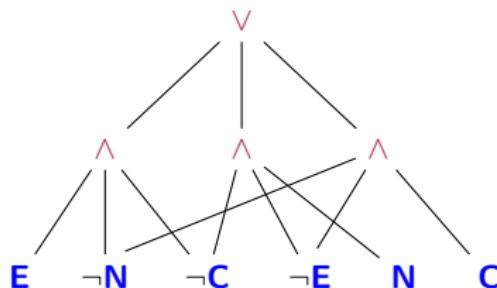
**Arithmetic circuit**



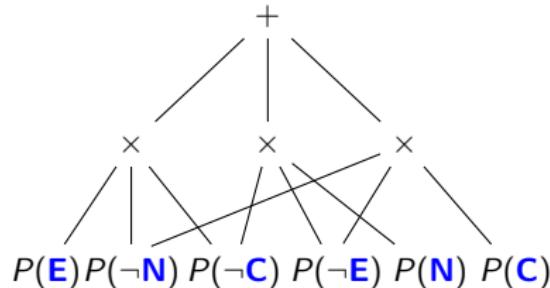
**Maximizer circuit**

Task	$A$	$\oplus$	$\otimes$
SAT	{true, false}	$\vee$	$\wedge$
#SAT (MC)	N	+	$\times$
MAXSAT	N	max	$\times$
WMC	$\mathbb{R}_{\geq 0}$	+	$\times$
WMAXSAT	$\mathbb{R}_{\geq 0}$	max	$\times$
PROB	$\mathbb{R}_{\geq 0}$	+	$\times$
MPE	$\mathbb{R}_{\geq 0}$	max	$\times$
FUZZY	[0,1]	max	min

## Creating tractable representations (reminder)



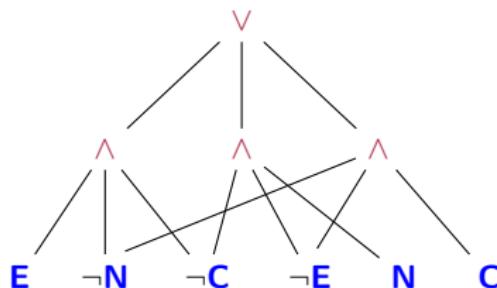
Boolean circuit



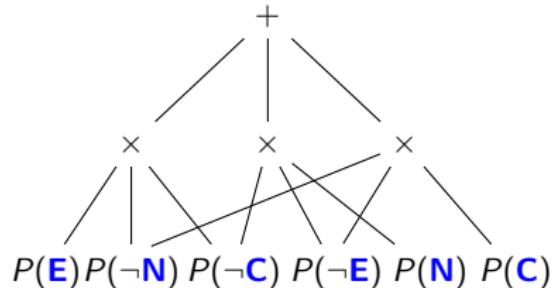
Arithmetic circuit

- ▶ Facilitates solving these algorithmic problems (SAT, WMC, MAXSAT) efficiently. Two key properties:

## Creating tractable representations (reminder)



Boolean circuit



Arithmetic circuit

- ▶ Facilitates solving these algorithmic problems (SAT, WMC, MAXSAT) efficiently. Two key properties:

**decomposability:** For conjunction nodes  $\wedge$ , children never share the same variables.

**determinism:** For each disjunction node  $\vee$ , the children are logically inconsistent.

## Compilation Software: example

```
1 from pysdd.sdd import SddManager
2
3 sdd = SddManager(var_count=3)
4 p1, p2, p3 = sdd.vars
5 uniqueness = (p1 & -p2 & -p3) | (-p1 & p2 & -p3) \
6             | (-p1 & -p2 & p3)
7
8 count = uniqueness.wmc(log_mode=False)
9 print(f"model count: {count.propagate()}")
```

<https://github.com/wannesm/PySDD>, see Darwiche (2011)

- ▶ Can extract underlying circuit, mix with Pytorch.

## Interim summary

- ▶ Focus on the classic **MPE** from bayes nets/directed PGMs, finding the most probable complete variable instantiation.

## Interim summary

- ▶ Focus on the classic **MPE** from bayes nets/directed PGMs, finding the most probable complete variable instantiation.

The problem we want to solve in our case, for model correction  
(most probable world)

## Interim summary

- ▶ Focus on the classic **MPE** from bayes nets/directed PGMs, finding the most probable complete variable instantiation.
  - The problem we want to solve in our case, for model correction  
(most probable world)
- ▶ Can be reduced to MAXSAT, requires some reduction into weighted logic  
(see Park (2002), example follows [here](#)).

## Interim summary

- ▶ Focus on the classic **MPE** from bayes nets/directed PGMs, finding the most probable complete variable instantiation.
  - The problem we want to solve in our case, for model correction (most probable world)
- ▶ Can be reduced to MAXSAT, requires some reduction into weighted logic (see Park (2002), example follows [here](#)).

Model correction via compilation to MAXSAT has received recent attention in NLP ([Kassner et al., 2021, 2023](#); [Gu et al., 2022](#); [Mitchell et al., 2022](#)).

# Can't we train models on such constraints?

## Logically Consistent Language Models via Neuro-Symbolic Integration

Diego Calanzzone\*

DISI, University of Trento

diego.calanzzone@studenti.unitn.it

Stefano Teso

CIMeC & DISI, University of Trento

stefano.teso@unitn.it

Antonio Vergari

School of Informatics, University of Edinburgh

avergari@ed.ac.uk

### Abstract

Large language models (LLMs) are a promising venue for natural language understanding and generation. However, current LLMs are far from reliable: they are prone to generating non-factual information and, more crucially, to contradicting themselves when prompted to reason about relations between entities of the world. These problems are currently addressed with large scale fine-tuning or by delegating reasoning to external tools. In this work, we strive for a middle ground and introduce a loss based on neuro-symbolic reasoning that teaches an LLM to be logically consistent with an external set of facts and rules and improves self-consistency even when the LLM is fine-tuned on a limited set of facts. Our approach also allows to easily combine multiple logical constraints at once in a principled way, delivering LLMs that are more consistent w.r.t. *all* constraints and improve over several baselines w.r.t. a given constraint. Moreover, our method allows LLMs to extrapolate to unseen but semantically similar factual knowledge, represented in unseen datasets, more systematically.

# Logic programming

## Alternative approach from yesterday

- ▶ **Idea:** Take a directed model (e.g., bayes net), use your neural model to parameterize the weights of that model.

## Alternative approach from yesterday

- ▶ **Idea:** Take a directed model (e.g., bayes net), use your neural model to parameterize the weights of that model.

*Best of both worlds?* keep properties of your original model (e.g., logical or probabilistic inference) improved estimation via neural methods. ([De Raedt et al., 2020](#))

# Directed Relational Models: DeepProblog

- An approach based on (probabilistic) logic programming ([Manhaeve et al., 2018](#)).

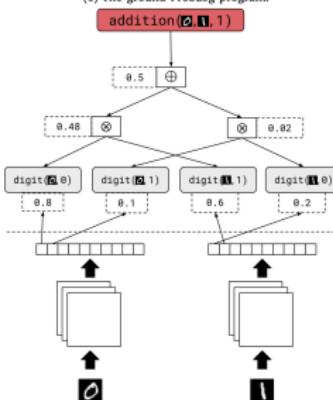
(a) The DeepProLog program.

```
nn(m_digit,[0],0)::digit(0,0);nn(m_digit,[0],1)::digit(0,1).  
nn(m_digit,[1],0)::digit(1,0);nn(m_digit,[1],1)::digit(1,1).  
addition([0][1],1) :- digit(0,0), digit(1,1).  
addition([0][1],1) :- digit(0,1), digit(1,0).
```

(b) The ground DeepProLog program.

```
0.8 :: digit(0,0); 0.1 :: digit(0,1).  
0.2 :: digit(1,0); 0.6 :: digit(1,1).  
addition([0][1],1) :- digit(0,0), digit(1,1).  
addition([0][1],1) :- digit(0,1), digit(1,0).
```

(c) The ground ProLog program.



(d) The AC for query  $\text{addition}([0][1], 1)$ .

## Semantic differences: logic programming and full logic

- Logic programming is based on clausal logic, of the form:

$$\underbrace{P_1}_{\text{head/conclusion}} \leftarrow \underbrace{P_2, P_3, \dots, P_n}_{\text{body}}$$

# Semantic differences: logic programming and full logic

- Logic programming is based on clausal logic, of the form:

$$\underbrace{P_1}_{\text{head/conclusion}} \leftarrow \underbrace{P_2, P_3, \dots, P_n}_{\text{body}}$$

Corresponding (in our terms) to a formula  $P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow P_1$ , or a clause

$$(\neg P_2 \vee \neg P_3 \vee \dots \vee \neg P_n \vee P_1)$$

## Semantic differences: logic programming and full logic

- Logic programming is based on clausal logic, of the form:

$$\underbrace{P_1}_{\text{head/conclusion}} \leftarrow \underbrace{P_2, P_3, \dots, P_n}_{\text{body}}$$

Corresponding (in our terms) to a formula  $P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow P_1$ , or a clause

$$(\neg P_2 \vee \neg P_3 \vee \dots \vee \neg P_n \vee P_1)$$

- **Definite clauses:** limit to a single head, one positive literal; like in Bayes nets, arrows suggest *causality, what can be inferred from what*

## Semantic differences: logic programming and full logic

- ▶ **Definite clauses:** limit to a single head, one positive literal; like in Bayes nets, arrows suggest *causality, what can be inferred from what*

**Example:** from [Raedt et al. \(2016\)](#)

```
flies ← bird,normal  
bird ← tweety  
bird ← oliver  
normal ← tweety  
tweety
```

## Semantic differences: logic programming and full logic

- ▶ **Definite clauses:** limit to a single head, one positive literal; like in Bayes nets, arrows suggest *causality*, *what can be inferred from what*

**Example:** from Raedt et al. (2016)

```
flies ← bird, normal  
bird ← tweety  
bird ← oliver  
normal ← tweety  
tweety
```

First case, can only make the conclusion (`flies`) if we know that the body is true; ordinary case: just an unordered clause:

$$(\text{flies} \vee \neg \text{bird} \vee \neg \text{normal})$$

# Semantic differences: logic programming and full logic

- ▶ Semantic assumes a kind of **closed world assumption**, *least Herbrand model*.

```
flies ← bird, normal  
bird ← tweety  
bird ← oliver  
normal ← tweety  
tweety
```

*Operationally:* we don't work with the above logic program, but a rewritten version via *Clark completion rules* ([Clark, 1977](#)):

```
flies↔bird∧normal  
bird↔tweety∨oliver  
normal↔tweety  
tweety  
oliver↔false
```

# Conclusion

- ▶ Covered methods for improving models at **inference time** (e.g., frozen LLMs) via logical-probabilistic reasoning.
- ▶ **MPE** inference, finding the most probably complete variable assignment, common from Bayes nets, can be used via reduction to **MAXSAT**
- ▶ **Next lecture:** prompting and probabilistic (imperative) programming.

Thank you.

## References I

- Clark, K. L. (1977). Negation as failure. In *Logic and data bases*, pages 293–322. Springer.
- Darwiche, A. (2011). Sdd: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*.
- De Raedt, L., Dumančić, S., Manhaeve, R., and Marra, G. (2020). From statistical relational to neuro-symbolic artificial intelligence. *arXiv preprint arXiv:2003.08316*.
- Dohan, D., Xu, W., Lewkowycz, A., Austin, J., Bieber, D., Lopes, R. G., Wu, Y., Michalewski, H., Saurous, R. A., Sohl-Dickstein, J., et al. (2022). Language model cascades. *arXiv preprint arXiv:2207.10342*.
- Gordon, A. D., Henzinger, T. A., Nori, A. V., and Rajamani, S. K. (2014). Probabilistic programming. In *Future of software engineering proceedings*, pages 167–181.
- Gu, Y., Mishra, B. D., and Clark, P. (2022). Do language models have coherent mental models of everyday things? *arXiv preprint arXiv:2212.10029*.
- Kassner, N., Tafjord, O., Sabharwal, A., Richardson, K., Schuetze, H., and Clark, P. (2023). Language models with rationality. *arXiv preprint arXiv:2305.14250*.
- Kassner, N., Tafjord, O., Schütze, H., and Clark, P. (2021). BeliefBank: Adding memory to a pre-trained language model for a systematic notion of belief. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8849–8861, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

## References II

- Kimmig, A., Van den Broeck, G., and De Raedt, L. (2011). An algebraic prolog for reasoning about possible worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 209–214.
- Kimmig, A., Van den Broeck, G., and De Raedt, L. (2017). Algebraic model counting. *Journal of Applied Logic*, 22:46–62.
- Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). Deepprobabilog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31.
- Mitchell, E., Noh, J. J., Li, S., Armstrong, W. S., Agarwal, A., Liu, P., Finn, C., and Manning, C. D. (2022). Enhancing self-consistency and performance of pre-trained language models through natural language inference. *arXiv preprint arXiv:2211.11875*.
- Park, J. D. (2002). Using weighted max-sat engines to solve mpe. In *AAAI/IAAI*, pages 682–687.
- Raedt, L. D., Kersting, K., Natarajan, S., and Poole, D. (2016). Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189.