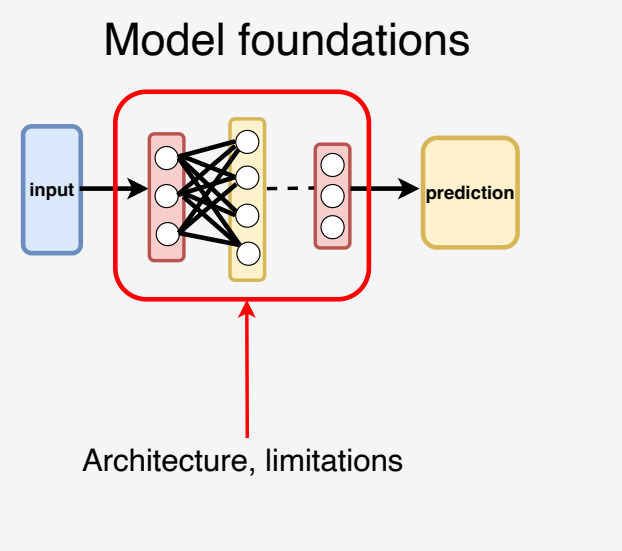
Language Model Programming Lecture 5: Grammar-Constrained Decoding

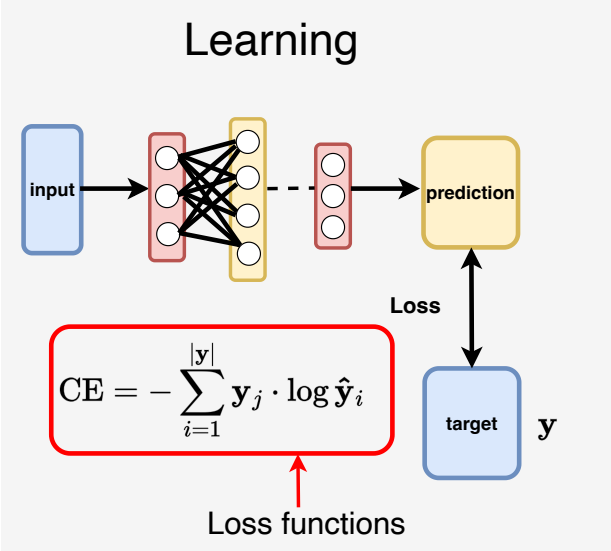
Kyle Richardson, **Gijs Wijnholds**

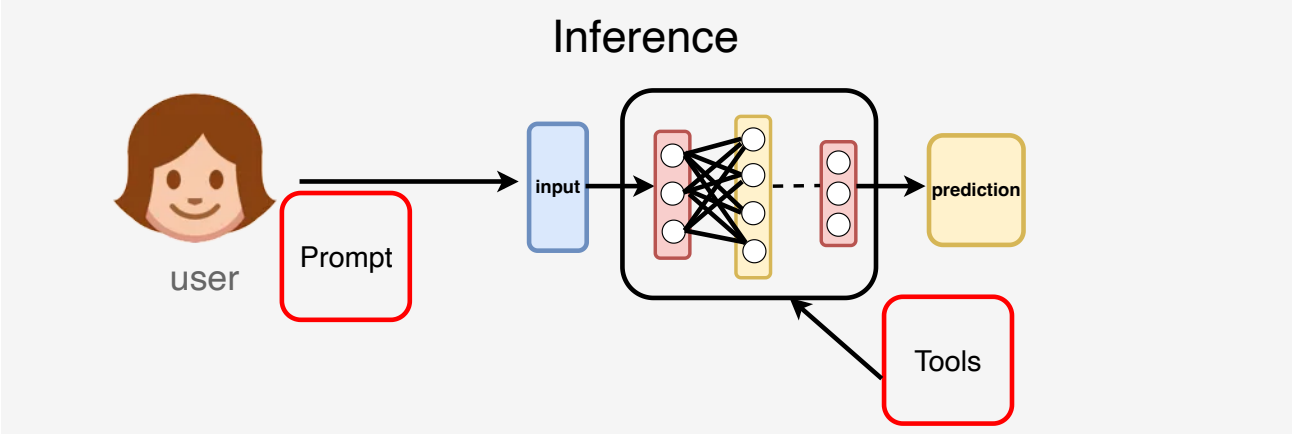
Allen Institute for AI (AI2)
Leiden Institute of Advanced Computer Science

August 2024

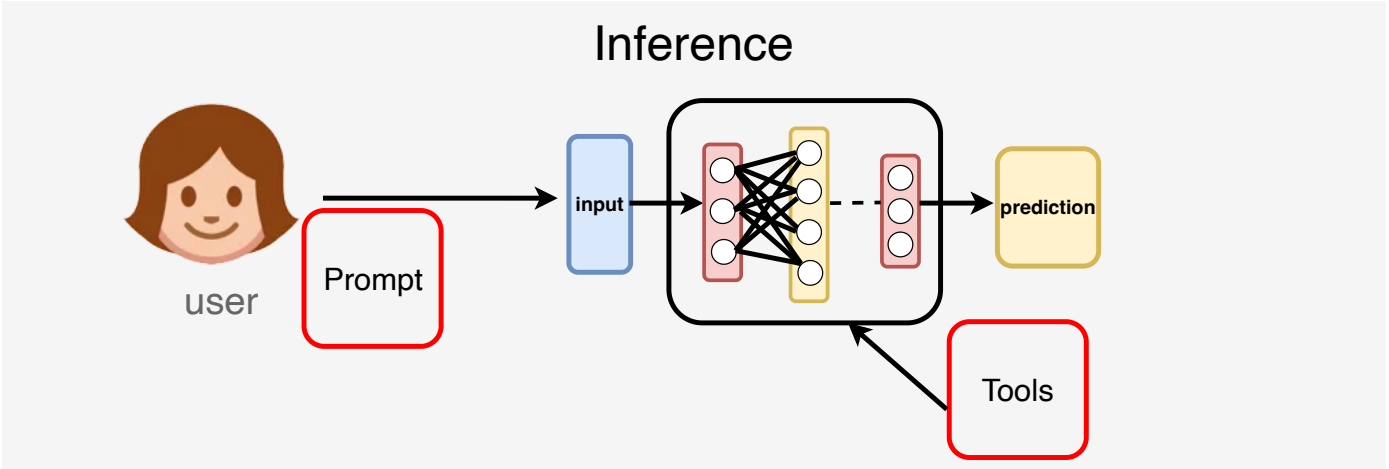
Course Overview







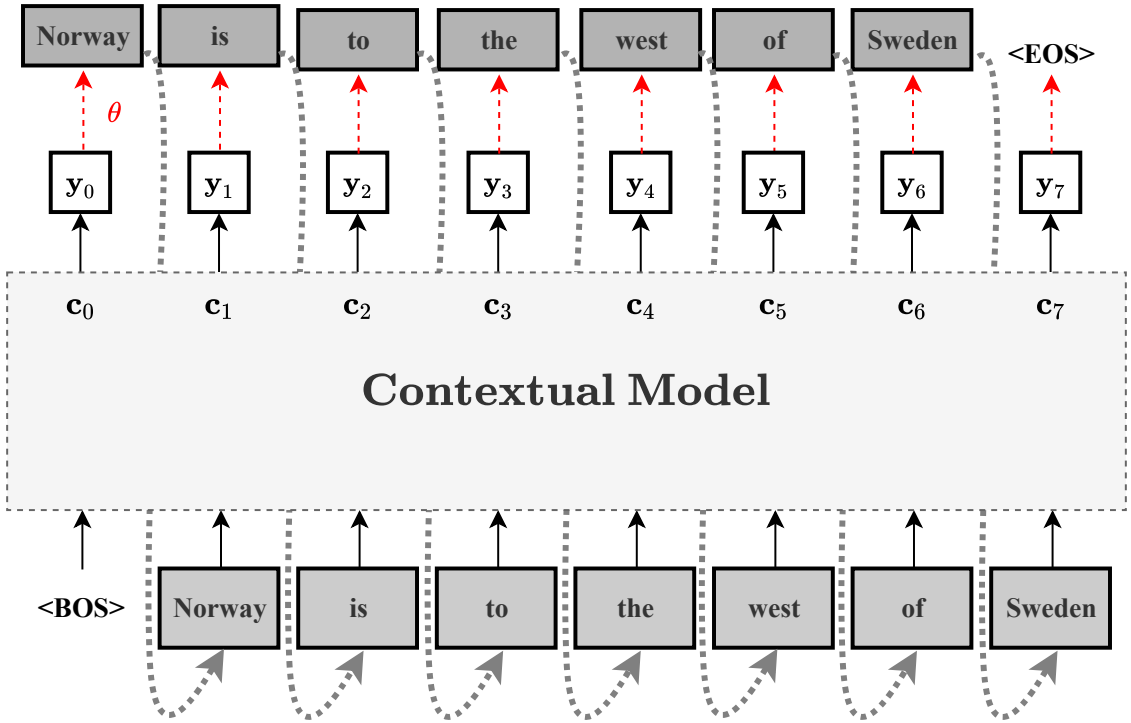
Today's Focus



- Grammar-constrained decoding
- Break
- Applications: NLI and logical formulas
- Wrap up

Quick recap of yesterday

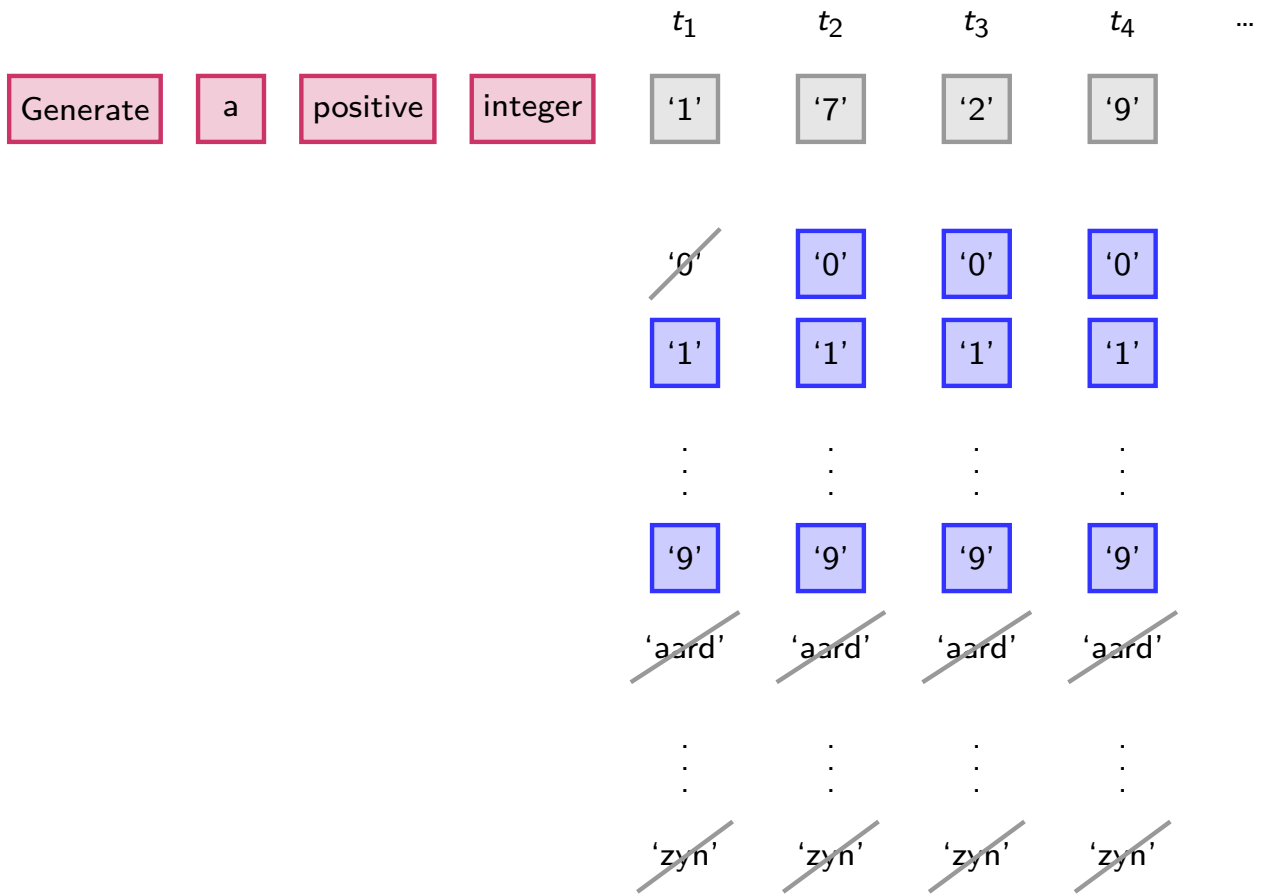
Prompting and decoding



- Learnt LLM probabilities are conditioned on the **prompt**
- The generated output is the combination of:
  - The prompt-conditioned probabilities
  - A **decoding strategy**

Masked Decoding

Core idea block out part of the vocabulary during decoding



Masked Decoding

Algorithm 2: Decoding

Input: trace u, scope σ, LM f
Output: decoded sequence v

$v \leftarrow \epsilon$
while ***True*** **do**
  $\boldsymbol{m} \leftarrow \text{compute\_mask}(u, \sigma, v)$
  if $\bigwedge_i (m_i = 0)$ **then break**
  $\boldsymbol{z} \leftarrow {}^{1}/_{Z} \cdot \boldsymbol{m} \odot \text{softmax}(\boldsymbol{f}(uv))$
  $t \leftarrow \text{pick}(z)$
  if $t = \textit{EOS}$ **then break**
  $v \leftarrow vt$
end

How to do this efficiently?

Yesterday's cliffhanger

But what if we want to generate
structured output 😍?

- JSON 😍?
- Python 😍😍?
- First-order logic 😍😍😍?

RegEx-constrained decoding

Regular expressions

Summary formal expressions designed to match against patterns in a string

Cheat sheet

Regex	Description	First match
`the`	exact match	o<u>the</u>r
`[tT]he`	choice	<u>The</u>re
`[0-9]`	single digit	<u>9</u> to 5
`[^A-Z]`	no uppercase	T<u>h</u>e
`colou?r`	optional	<u>color</u> <u>colour</u>
`o*h`	0 or more	<u>h</u>, <u>oh</u>, <u>ooh</u>
`o+h`	1 or more	<u>oh</u>, <u>ooh</u>, <u>oooh</u>
`[^A-Z]`	no uppercase	“T<u>h</u>e”
`beg.in`	any char	<u>begin</u>, <u>began</u>, <u>beg7n</u>

Finite state automata

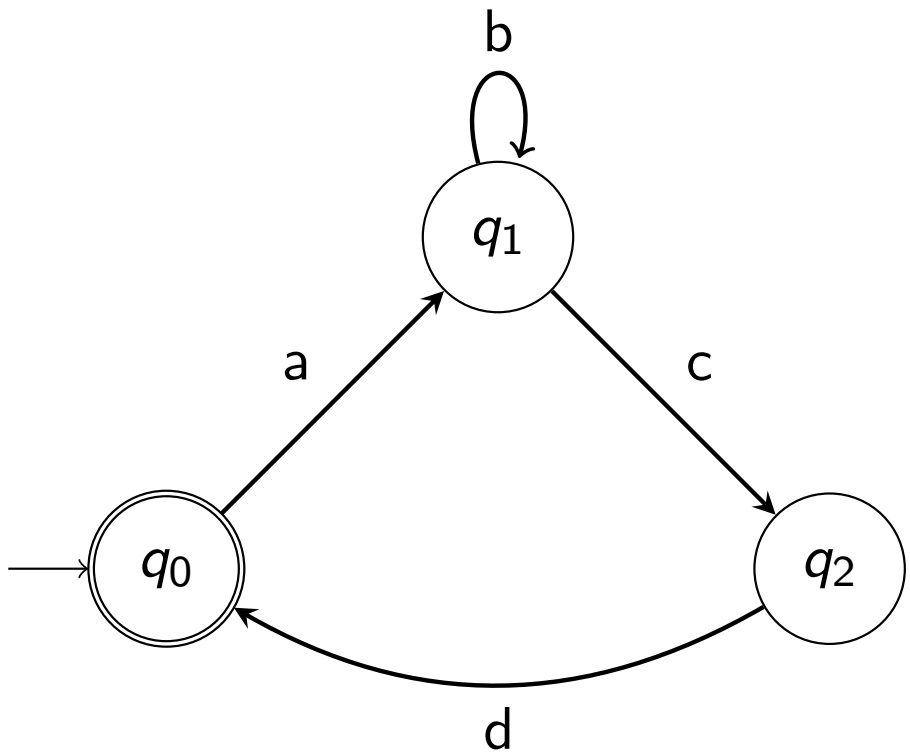
Core idea a transition machine that recognizes regular languages (i.e. generated by some regex)

Formally a tuple $(\Sigma, Q, \delta, s, F)$ where

- Σ is an alphabet,
- Q is a finite set of states,
- $\delta : \Sigma \times Q \to Q$ is a transition function,
- $s \in Q$ is the unique starting state,
- $F \subseteq Q$ is a set of accepting (or *final*) states.

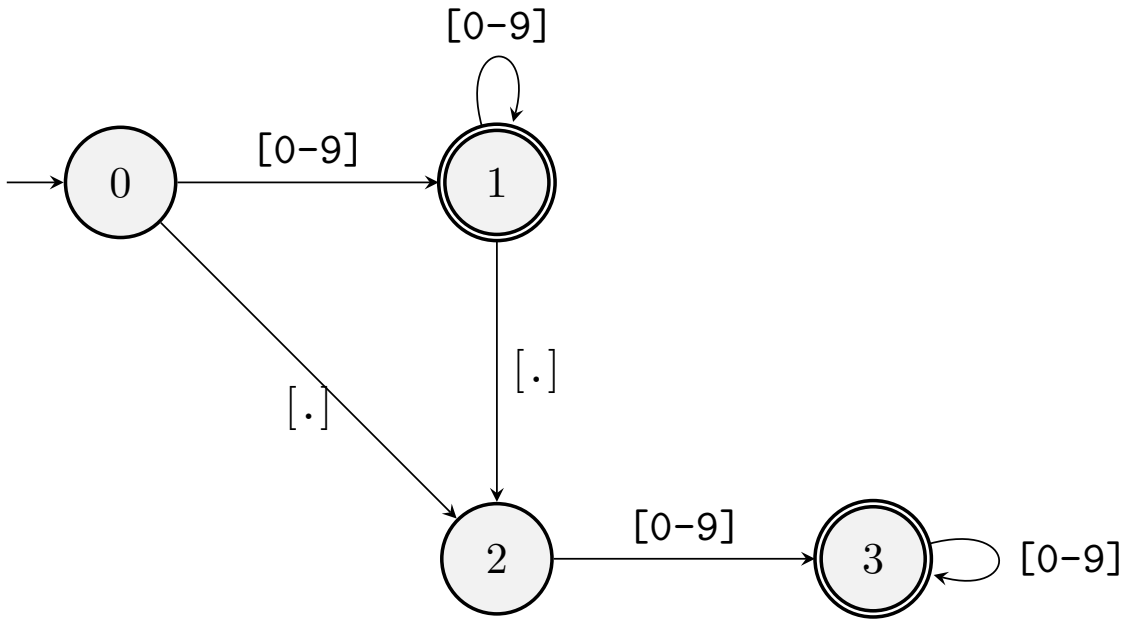
Recap: finite state automata

Drawing FSAs states as circles, transitions as labelled edges, accepting states are emphasized:



Recognition the language $(ab^*cd)^*$ is recognized by the above automaton

Another example: floating point numbers



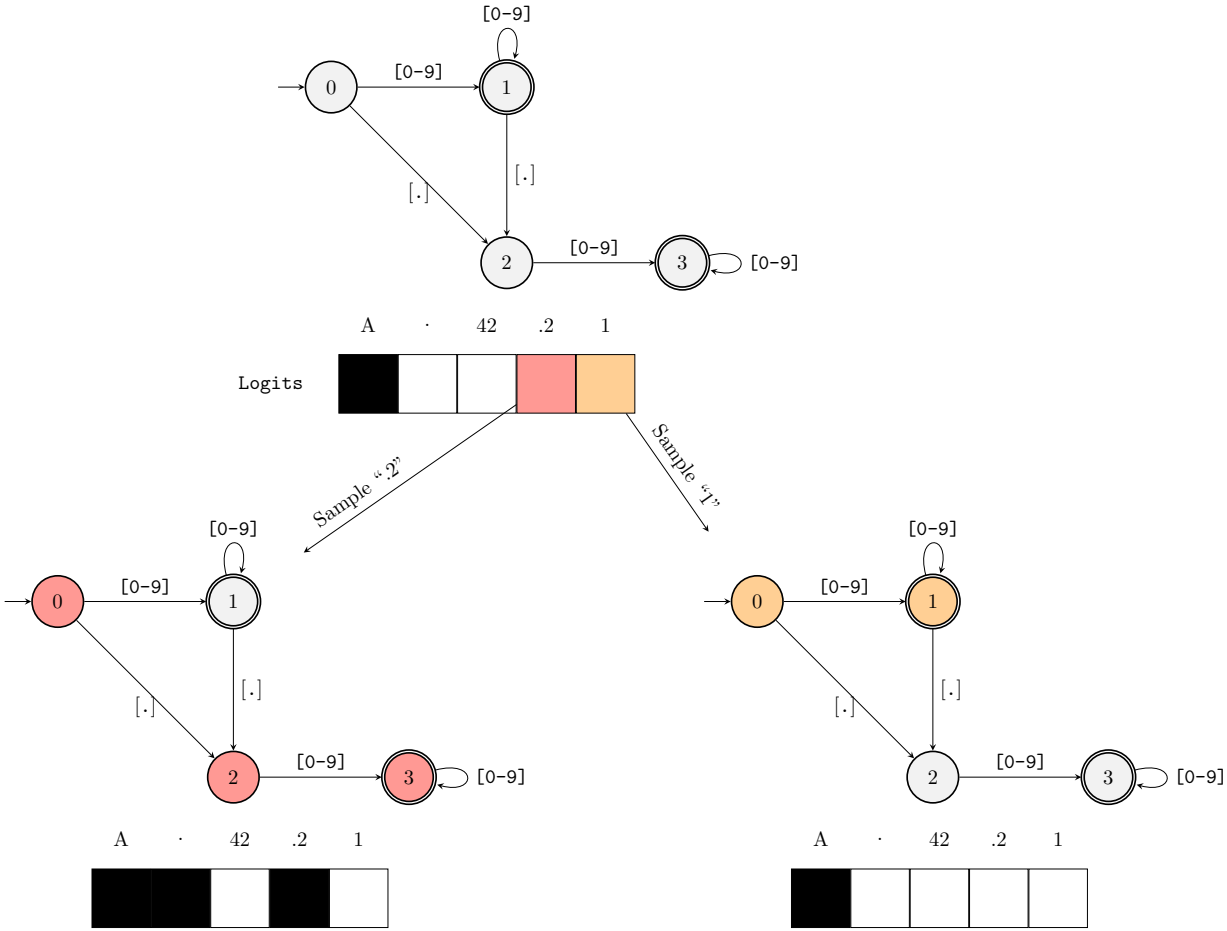
Example

Accepted	Rejected
`.57`	`8932...81`
`352.49`	`5.42.95`
`523`	

Willard and Louf [2023]

FSA-constrained decoding

Intuition keep track of the current state, and advance accordingly:



Willard and Louf [2023]

FSM Masking

Not ideal running over the whole vocabulary during decoding is expensive

FSM Masking

Not ideal running over the whole vocabulary during decoding is expensive

Why we ask the model to decode, then decide if this is a valid transition, then recompute if invalid

FSM Masking

Not ideal running over the whole vocabulary during decoding is expensive

Why we ask the model to decode, then decide if this is a valid transition, then recompute if invalid

Desideratum we wish to construct a map $\sigma : \mathcal{Q} \to \mathcal{P}(\mathcal{V})$, that maps states of the FSA to valid token sequences in the LM vocabulary

FSM Masking

Not ideal running over the whole vocabulary during decoding is expensive

Why we ask the model to decode, then decide if this is a valid transition, then recompute if invalid

Desideratum we wish to construct a map $\sigma : \mathcal{Q} \to \mathcal{P}(\mathcal{V})$, that maps states of the FSA to valid token sequences in the LM vocabulary

New algorithm σ is an index of valid transitions (the token mask), during decoding we just keep track of current FSA state

Efficient guidance for regular expressions

Algorithm 3 Find sub-sequences of the FSM M that accept the string $\boldsymbol{v}$

```
1:  function FIND_SUB_SEQUENCES(M, v)
2:      M = (Q, Σ, δ, q_0, F)
3:      res ← ()
4:      for r ∈ δ^{-1}(·, v_0) do          ▷ Loop through states that read v_0
5:          p ← (r)
6:          for i ← 1, |v| − 1 do                          ▷ Walk the FSM
7:              if δ(r, v_i) = ∅ then          ▷ The FSM does not read v_i
8:                  p ← ()
9:                  break          ▷ Stop walking and try the next start state
10:             end if
11:             r ← δ(r, v_i)
12:             p ← append(p, r)
13:         end for
14:         res ← append(res, p)
15:     end for
16:     return res
17: end function
```

Willard and Louf [2023]

Efficient guidance for regular expressions

Algorithm 4 Construct a map from FSM states to subsets of $\mathcal{V}$

function MAP_STATES_TO_VOCAB(M, $\mathcal{V}$)
    $M = (Q, \Sigma, \delta, q_0, F)$
    Initialize the map σ with empty sets for each element in Q
    for $v \in \mathcal{V}$ **do** ▷ Loop through the vocabulary
        $Z \leftarrow$ find_sub_sequences(M, v)
        for $z \in Z$ **do** ▷ Loop through state sequences accepting v
            $\sigma(z_0) \leftarrow \sigma(z_0) \cup v$
        end for
    end for
    return σ
end function

Guided decoding during decoding, we assign each token in the vocabulary and the current state to the corresponding resulting state and continue

Willard and Louf [2023]

The limits

Expressivity FSAs/RegEx have limited expressivity as there is no memory involved

The limits

Expressivity FSAs/RegEx have limited expressivity as there is no memory involved

Brackets If we want to generate structured output like JSON or Python, or logic formulas, we need better machinery

Grammar-constrained decoding

Context Free Grammars

Definition A context free grammar G is a tuple (N, Σ, P, S) where

- N is a set of help symbols (non-terminals),
- Σ is the alphabet of words (terminal symbols),
- $P \subseteq N \times (N \cup \Sigma)^*$ is a set of rewrite rules,
- $S \in N$ is the designated start symbol of the grammar

Context Free Grammars

Definition A context free grammar G is a tuple (N, Σ, P, S) where

- N is a set of help symbols (non-terminals),
- Σ is the alphabet of words (terminal symbols),
- $P \subseteq N \times (N \cup \Sigma)^*$ is a set of rewrite rules,
- $S \in N$ is the designated start symbol of the grammar

(a) Example Grammar

```
E = E | E + E | (E) | int
int = ([1-9][0-9]*)|(0+)
```

Efficient parsing

Core idea break up CFG recognition into *parsing* and *scanning/lexing*:

> **Lemma 3.1** *Let L_G be the language described by a CFG G. Further, let $r_1, \ldots, r_n$ be the regular expressions of the terminals of G and the $r_{\text{EOS}} = \$$. Then, it holds that:*
>
> - *The union of these regular expressions $r = r_1 | \ldots | r_n | r_{\text{EOS}}$ matches any terminal in G.*
> - *The regular expression $R = r+$ matches all non-empty sequences of terminals in the language.*
> - *The language L_R described by R contains all legal programs in G, i.e., $L_G \subseteq L_R$, but also*
> - *L_R contains some illegal programs, i.e., $L_R \not\subseteq L_G$.*

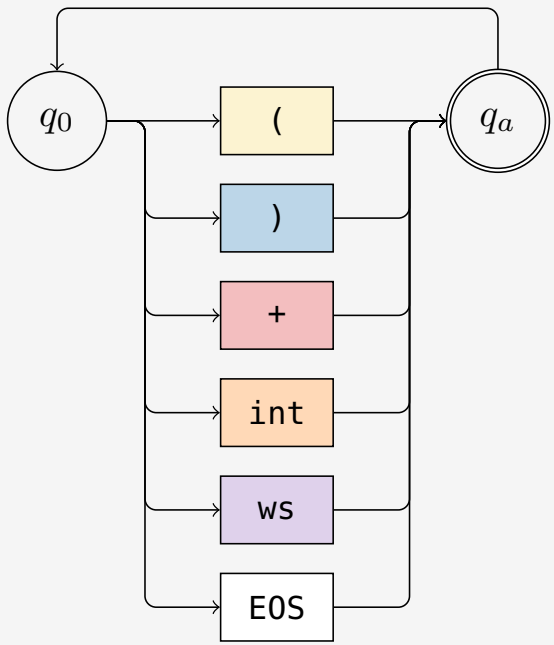
Intuition terminal symbols are analyzed with an FSA, high-level structure by the parser

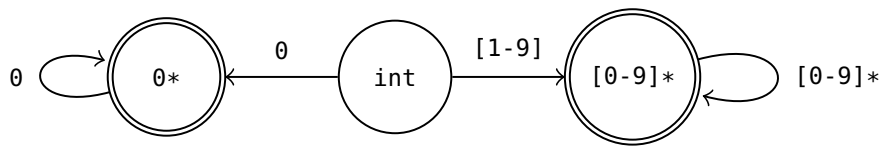
Scanner

(a) Example Grammar

```
E = E | E + E | (E) | int
int = ([1-9][0-9]*)|(0+)
```

(b) Character Scanner





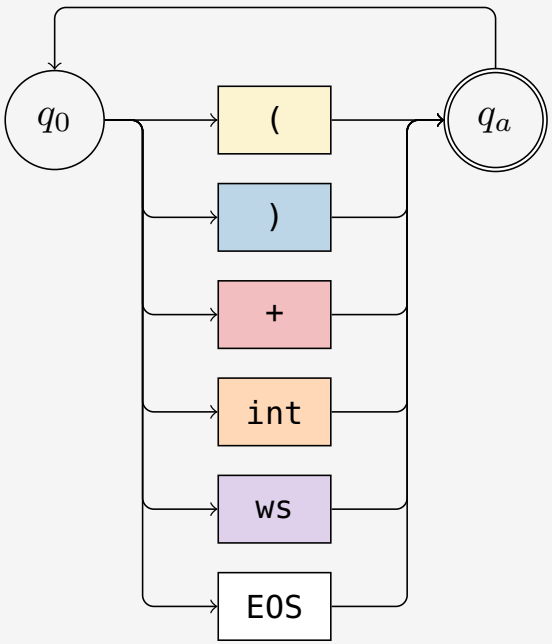
Expressivity enough to ensure that generated strings are in L_R, but not to guarantee that they are also in L_G

Vocabulary-Aligned Subterminal Tree

(a) Example Grammar

```
E = E | E + E | (E) | int
int = ([1-9][0-9]*)|(0+)
```

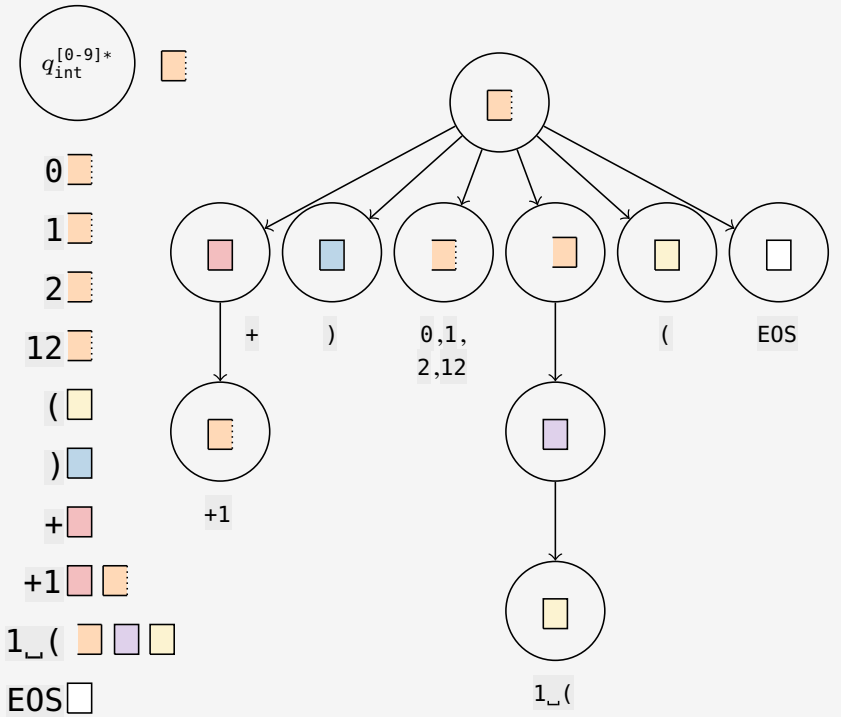
(b) Character Scanner



(c) Vocabulary

$$\mathcal{V} = \{0, 1, 2, 12,), (, +, +1\ 1\text{␣}(, \text{EOS}\}$$

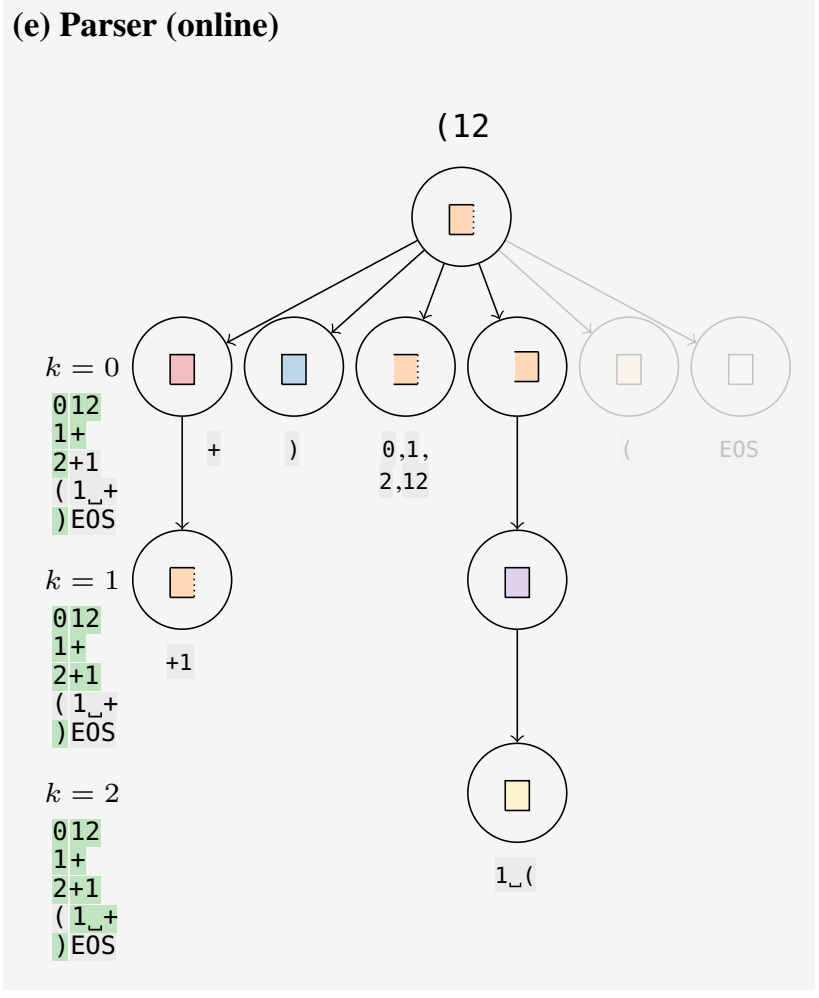
(d) Vocabulary-aligned Subterminal Tree (offline, per node)



Computation the scanner and subterminal tree are computed offline

Online parsing

Core idea Combine the subterminal tree with parser to cut off illegal continuations



Application: Grade School Math (GSM8k)

Dataset

Problem: Beth bakes 4, 2 dozen batches of cookies in a week. If these cookies are shared amongst 16 people equally, how many cookies does each person consume?
Solution: Beth bakes 4 2 dozen batches of cookies for a total of 4*2 = <<4*2=8>>8 dozen cookies
There are 12 cookies in a dozen and she makes 8 dozen cookies for a total of 12*8 = <<12*8=96>>96 cookies
She splits the 96 cookies equally amongst 16 people so they each eat 96/16 = <<96/16=6>>6 cookies
Final Answer: 6

Constrained output

```
{
    "thoughts": [
        {
            "step": "Find the distance
                     between the first and
                     second stops",
            "calculation": "60 - 20 - 15",
            "result": 25
        },
        {
            "step": "Find the distance
                     between the first and
                     second stops",
            "calculation": "25 + 15",
            "result": 40
        }
    ],
    "answer": 40
}
```

Accuracy

Configuration	*Mistral 7B*	*Llama-2 13B*
Unconstrained	0.415	0.155
DOMINO ($k = 0$)	0.308	0.0
DOMINO ($k = 1$)	0.1	0.036
DOMINO ($k = \infty$)	**0.418**	**0.157**

Beurer-Kellner et al. [2024]

Grammar-constrained output: some libraries

Table 1. Overview of different constrained decoding methods

	Regex	CFG	Pre-Computed	Minimally Invasive
LMQL	✓	✗	✗	✗
GUIDANCE	✓	(✓)	✗	(✓) *
OUTLINES	✓	✓	(✓) †	✗
PICARD	✓	✓	✗	✗
SYNCHROMESH	✓	✓	✗	✓
LLAMA.CPP	✓	✓	✗	✓ +
GCD	✓	✓	✗	✓
DOMINO (ours)	✓	✓	✓	✓

Beurer-Kellner et al. [2024]

Natural Language Inference with (constrained) LLMs

Recap of Natural Language Inference

Definition Given a premise p and hypothesis h, will h be **E**ntailed, **N**eutral, or **C**ontradicted by p?

Examples from the SICK dataset Marelli et al. [2014]

Premise	**Hypothesis**	**Label**
A deer is jumping over a fence	A deer isn't jumping over a fence	C
A player is running with the ball	Two teams are competing in a football match	N
An old man is sitting in a field	A man is sitting in a field	E

More datasets

- Larger, more fuzzy: SNLI Bowman et al. [2015]
- Multigenre: MNLI Williams et al. [2018]
- Multilingual: XNLI Conneau et al. [2018]

Monotonicity Reasoning as NLI

(a) ***Every*** [*man* ↓] [*sung and danced* ↑].
(b) *Every bald man sung and danced.* ✓
(c) *Every man danced.* ✓
(d) *Every human sung and danced.* ✗

Figure 1: Example cases of monotonicity reasoning as natural language inference.

Yanaka et al. [2019]

Monotonicity Reasoning and transfer

	SICK-NL$_d$	**SICK-NL$_t$**	**MED-NL**
BERTje	86.89	87.40	47.56
RobBERT	86.43	85.79	46.07
mBERT	71.20	71.38	49.74

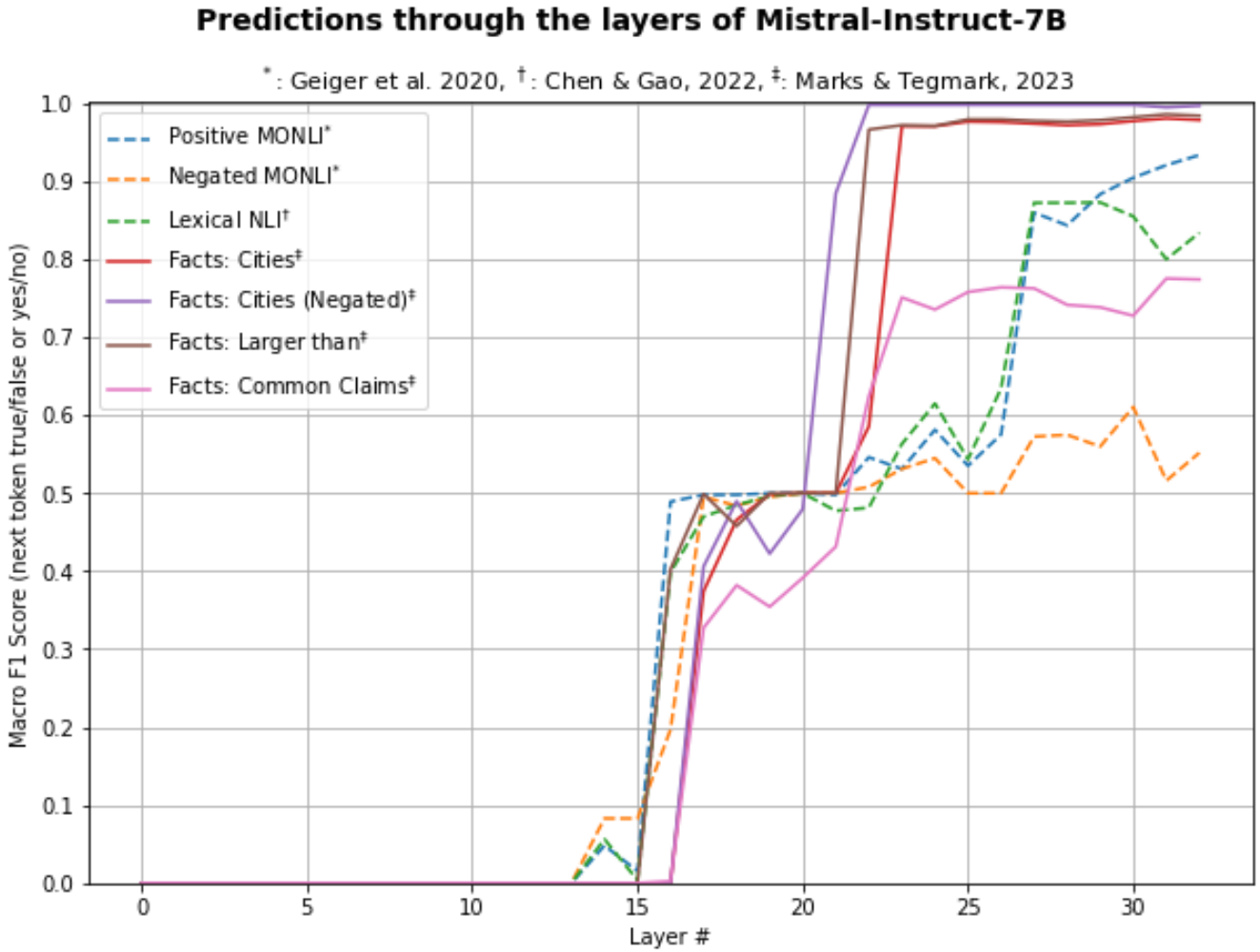
Table 2: Seed-averaged (over 3 runs) accuracy results for two Dutch BERT models and multilingual BERT, trained on SICK-NL, evaluated on both SICK-NL and the new MED-NL dataset.

% Negation	**SICK-NL**	**MED-NL**
Entailment	1.26	69.80
Non-entailment	31.94	47.81

Table 7: Distribution of negation in cases of entailment and non-entailment in SICK-NL and MED-NL.

Wijnholds [2023]

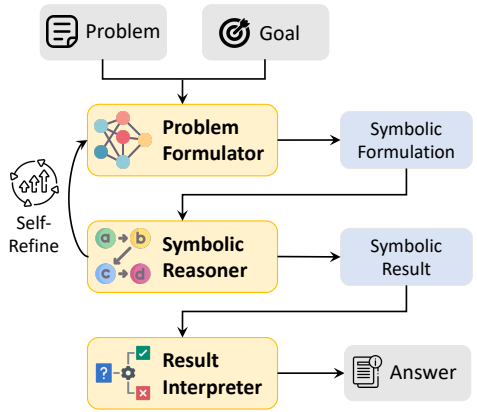
LLM prompting on negation



By: Anna Langedijk @ LIACS

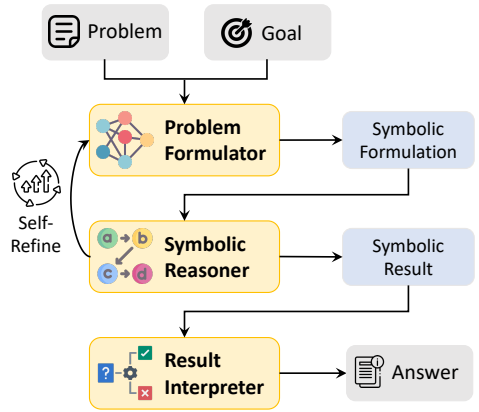
LLMs and logic

Setup



LLMs and logic

Setup



Prompt

```
Problem: "Stranger Things" is a popular Netflix show. If a Netflix show is popular, Karen will binge-watch it.
         If and only if Karen binge-watches a Netflix show, she will download it. Karen does not download "Black Mirror".
         "Black Mirror" is a Netflix show. If Karen binge-watches a Netflix show, she will share it to Lisa.
Question: Is the following statement true, false, or uncertain? "Black Mirror" is popular.
          (A) True              (B) False           (C) Uncertain
Generated symbolic Formulations:
    Predicates:
        NetflixShow(x) # x is a Netflix show.
        Popular(x) # x is popular.
        BingeWatch(x, y) # x binge-watches y.
        Download(x, y) # x downloads y.
        Share(x, y, z) # x shares y to z.
    Premises:
        NetflixShow(strangerThings) ∧ Popular(strangerThings) # "Stranger Things" is a popular Netflix show.
        ∀x (NetflixShow(x) ∧ Popular(x) → BingeWatch(karen, x)) # If a Netflix show is popular, Karen will binge-watch it.
        ∀x (NetflixShow(x) ∧ BingeWatch(karen, x) ↔ Download(karen, x)) # If and only if Karen … , she will download it.
        NetflixShow(blackMirror) ∧ ¬Download(karen, blackMirror) # Karen does not … "Black Mirror" is a Netflix show.
        ∀x (NetflixShow(x) ∧ BingeWatch(karen, x) → Share(karen, x, lisa)) # If Karen … , she will share it to Lisa.
    Conclusion:
        Popular(blackMirror) # "Black Mirror" is popular.
Predicted answer: B
```

Pan et al. [2023]

Monotonicity reasoning with Logic & LLMs

The plan investigate whether logical representations can help with difficult NLI tasks

Approach a mixture of constrained decoding and LLM prompting

The prompt

```
Given a pair of sentences, the task is to parse each sentence into first-order logic formulas.
The grammar of first-order logic formulas is defined as follows:
1) logical conjunction of expr1 and expr2: expr1 ∧ expr2
2) logical disjunction of expr1 and expr2: expr1 ∨ expr2
3) logical negation of expr1: ¬expr1
5) expr1 implies expr2: expr1 → expr2
6) expr1 if and only if expr2: expr1 ↔ expr2
7) logical universal quantification: ∀x
8) logical existential quantification: ∃x
```

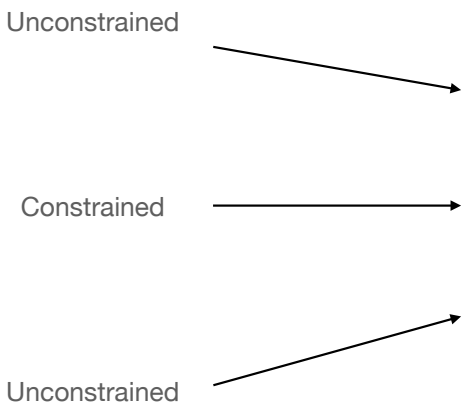
Constraining to formulas #1

```
@guidance(stateless=True)
def constant(lm):
    return lm + gen(regex='[a-u]', max_tokens=5)

@guidance(stateless=True)
def variable(lm):
    return lm + gen(regex='[v-z][0-9]*', max_tokens=5)

@guidance(stateless=True)
def term(lm):
    return lm + select([variable(), constant()])

@guidance(stateless=True)
def quantifier(lm):
    return lm + select(["∀", "∃"])

@guidance(stateless=True)
def unop(lm):
    return lm + "¬"

@guidance(stateless=True)
def binop(lm):
    return lm + select(["→", "↔", "∧", "∨"])
```

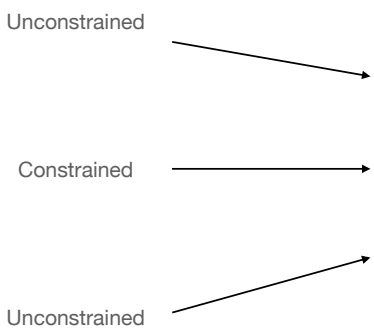
Constraining to formulas #2

```
@guidance(stateless=True)
def form(lm, words, name=None):
    return lm + select([
        aform(words),
        unop() + aform(words),
        unop() + '(' + form() + ')',
        aform(words) + ws() + binop() + ws() + aform(words),
        aform(words) + ws() + binop() + ws() + '(' + form(words) + ')',
        quantifier() + variable() + aform(words),
        quantifier() + variable() + '(' + form(words) + ')'
    ], name=name)
```

Aliens



Item 4909
Prm: No alien bought any shirts.
Hyp: No alien bought clothes.
preds1: Alien(x, y), Bought(x, y), Bought(x, y)
P1: ∀x ∀y (Alien(x, y) → ¬Bought(x, y))
H1: ∀x ∀y (Alien(x, y) → ¬Bought(x, y))
preds2: Alien(x, y), Shirts(x, y), Clothes(x, y)
P2: ¬(Alien(x, y) ∧ Shirts(x, y))
H2: ¬(Alien(x, y) ∧ Clothes(x, y))
preds3: Alien(x, y), Shirts(x, y), Clothes(x, y)
P3: ¬(Alien(x, y) ∧ Shirts(x, y))
H3: ¬(Alien(x, y) ∧ Clothes(x, y))



Item 4431
Prm: Only 12 aliens ate at some restaurants.
Hyp: Only 12 aliens ate at Mexican restaurants.
preds1: Alien(x, y), Mexican(x, y)
P1: ∀x ∀y (Alien(x, y) → ∃z (Mexican(z, y) → ∀w (Alien(x, w) → ¬Mexican(z, w))))
H1: ∀x ∀y (Alien(x, y) → ∃z (Mexican(z, y) → ∀w (Alien(x, w) → ¬Mexican(z, w))))
preds2: Restaurants(x, y), Mexican(x, y)
P2: ∀x ∀y (Restaurants(x, y) ∧ Restaurants(y, x))
H2: ∀x ∀y (Restaurants(x, y) ∧ Mexican(y, x))
preds3: Restaurants(x, y), Mexican(x, y)
P3: ∀x ∀y (Restaurants(x, y) ∧ 12 ≤ ∃z (Restaurants(z, y)))
H3: ∀x ∀y (Restaurants(x, y) ∧ 12 ≤ ∃z (Mexican(z, y)))

Changing the representation

The model we used Zephyr, an open source model optimized for chat/feedback

Sensitivity to representation The model works much better with a different representation format:

Before

```
Given a pair of sentences, the task is to parse each sentence into first-order logic formulas.
The grammar of first-order logic formulas is defined as follows:
1) logical conjunction of expr1 and expr2: expr1 ∧ expr2
2) logical disjunction of expr1 and expr2: expr1 ∨ expr2
3) logical negation of expr1: ¬expr1
5) expr1 implies expr2: expr1 → expr2
6) expr1 if and only if expr2: expr1 ↔ expr2
7) logical universal quantification: ∀x
8) logical existential quantification: ∃x
```

After

```
Given a pair of sentences, the task is to parse each sentence into first-order logic formulas.
The grammar of first-order logic formulas is defined as follows:
1) logical conjunction of expr1 and expr2: expr1 & expr2
2) logical disjunction of expr1 and expr2: expr1 | expr2
3) logical negation of expr1: -expr1
5) expr1 implies expr2: expr1 -> expr2
6) expr1 if and only if expr2: expr1 <-> expr2
7) logical universal quantification over expr1: forall x. expr1
8) logical existential quantification over expr1: exists x. expr1
```

Results

Three modes

- **Direct Answer** asking the model to directly decide on an NLI label
- **With formulas** asking the model generate a formula, then decide on an NLI label
- **Prover9** Passing the generated formulas through a theorem prover

Results

Three modes

- **Direct Answer** asking the model to directly decide on an NLI label
- **With formulas** asking the model generate a formula, then decide on an NLI label
- **Prover9** Passing the generated formulas through a theorem prover

F1 scores

	Entailment	Neutral	Macro Avg.
Direct Answer	45.39	48.93	47.16
With formulas	69.58	44.97	57.27
Prover9	27.42	55.23	41.33

Preliminary Conclusion

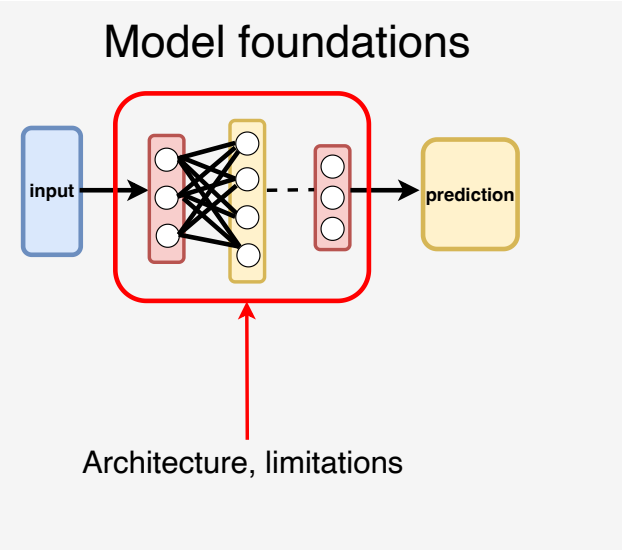
Grammar-constrained decoding we can use constraints to force a model to generate correct first-order logic formulas

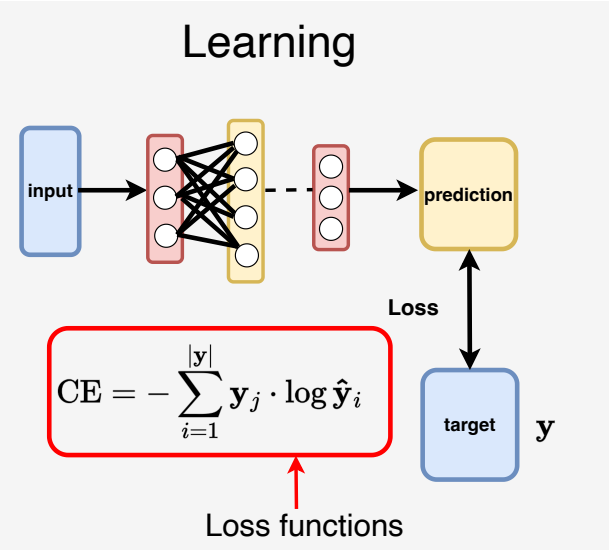
Performance and explanation generating logic can help the model in reasoning about complex NLI cases

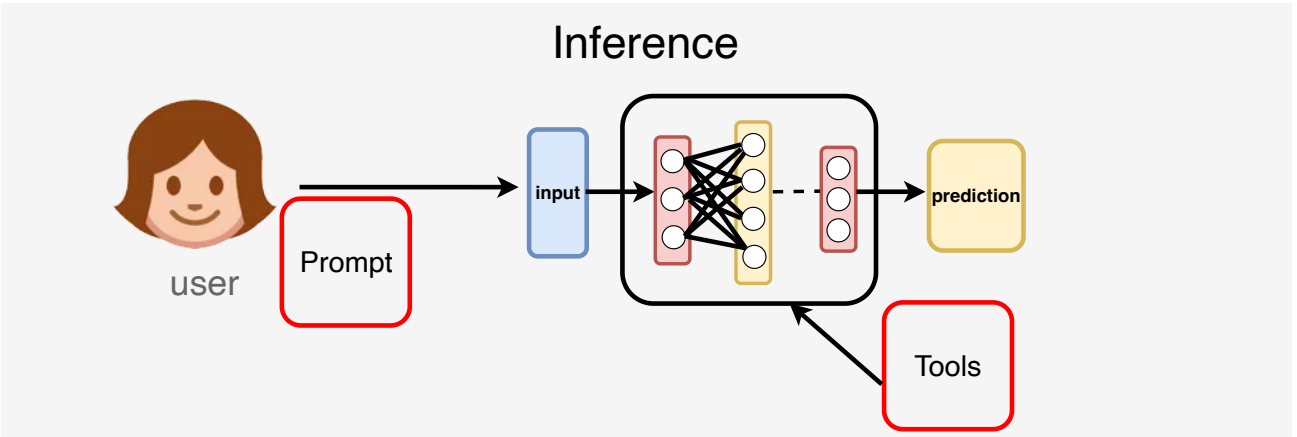
Theorem proving we can use a theorem prover to verify semantic correctness of the generated formulas

Limitations work in progress, feel free to come with suggestions!

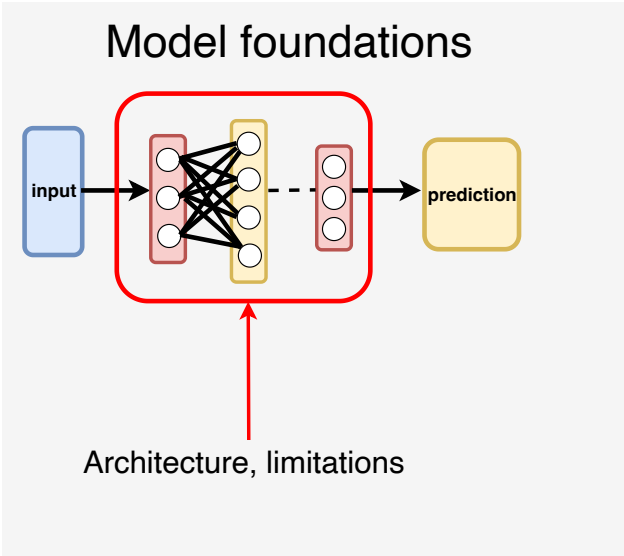
Course Overview





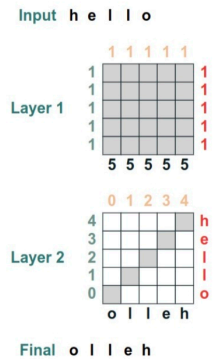


Lecture 1



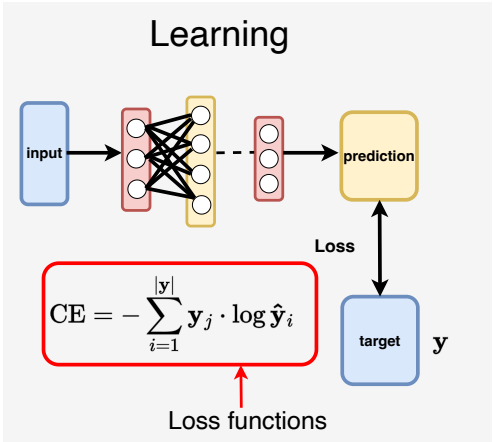
RASPy

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



Question what are the limits of Transformer expressivity?

Lecture 2



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)
```

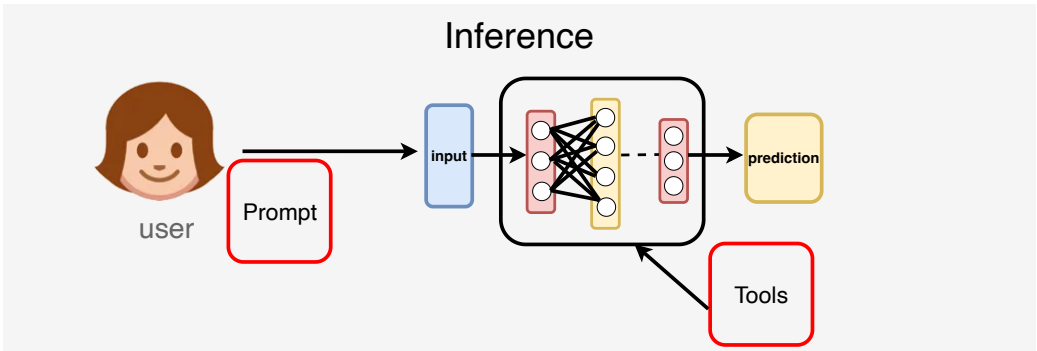
Fig. 3. The logic program of the PacMan-Maze application in Scallop.

Question how can semantic loss help, in training more efficiently?

Lecture 3

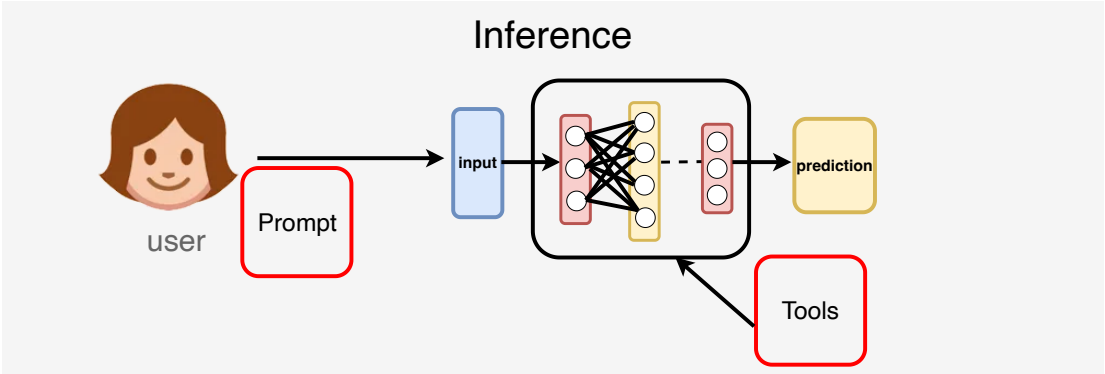
```
1  // File path_planner.scl
2  type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)
3
4  const UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3
5  rel safe_cell(x, y) = range(0, 5, x), range(0, 5, y), not enemy(x, y)
6  rel edge(x, y, x, yp, UP) = safe_cell(x, y), safe_cell(x, yp), yp == 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.



Question what kind of consistency checking can we define declaratively?

Lecture 4



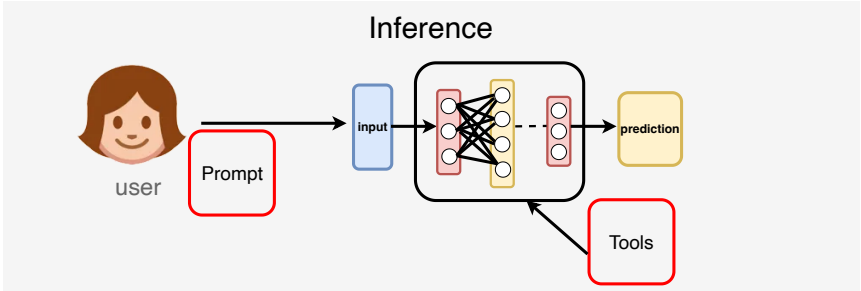

```
argmax
  "A list of things not to forget when "
  "travelling:\n"
  things = []
  for i in range(2):
    "- [THING]\n"
    things.append(THING)
  "The most important of these is [ITEM]."
from "EleutherAI/gpt-j-6B"
where
   THING in ["passport",
             "phone",
             "keys", ...] // a longer list
   and len(words(ITEM)) <= 2
```

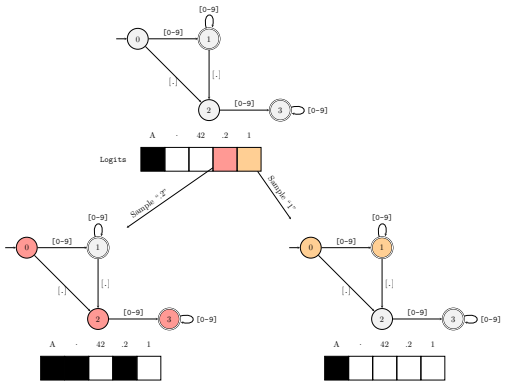
expression	$\textsc{Follow}[\cdot](u,t)$
$\langle$const$\rangle$	$[\![\langle\text{const}\rangle]\!]_\sigma$
python variable $\langle$pyvar$\rangle$	$[\![\texttt{pyvar}]\!]_{\sigma[v\leftarrow vt]}$
previous hole $\langle$var$\rangle$	$[\![\langle\text{var}\rangle]\!]_\sigma$
current var v	$\begin{cases}\textsc{fin}(v) & \text{if } t = \textsc{eos}\\ \textsc{inc}(vt) & \text{else}\end{cases}$
future hole $\langle$var$\rangle$	None
words(v)	$\begin{cases}\textsc{fin}(w_1,\ldots,w_k) & \text{if } t = \textsc{eos}\\ \textsc{inc}(w_1,\ldots,w_k) & \text{if } t = \text{␣}\\ \textsc{inc}(w_1,\ldots,w_k t) & \text{else}\end{cases}$ where $w_1,\ldots,w_k \leftarrow [\![\texttt{words}(v)]\!]_\sigma$
sentences(v)	$\begin{cases}\textsc{fin}(s_1,\ldots,s_k) & \text{if } t = \textsc{eos}\\ \textsc{inc}(s_1,\ldots,s_k,t) & \text{if } s_k\texttt{.endswith(".")}\\ \textsc{inc}(s_1,\ldots,s_k t) & \text{else}\end{cases}$ where $s_1,\ldots,s_k \leftarrow [\![\texttt{sentences}(v)]\!]_\sigma$
len(v)	$\begin{cases}\texttt{len}(v) & \text{if } t = \textsc{eos}\\ \texttt{len}(v)+1 & \text{else}\end{cases}$
len(l) over list l	$len([\![l]\!]_{\sigma[v\leftarrow vt]})$

expression	$\textsc{Follow}[\cdot](u,t)$
fn$(\tau_1,\ldots,\tau_k)$	$\texttt{fn}([\![\tau_1]\!]_{\sigma[v\leftarrow vt]},\ldots,[\![\tau_k]\!]_{\sigma[v\leftarrow vt]})$
stop_at$(var,$ s$)$	$\begin{cases}\textsc{fin}(b) & \text{if } b \wedge \textsc{Final}[var] = \textsc{inc}\\ \textsc{var}(l) & \text{else}\end{cases}$ where $b = [\![var]\!]_\sigma\texttt{.endswith}(s)$
x in s for string s and constant x	$\begin{cases}\top & \text{if x in s} \vee \text{x in } t\\ \bot & \text{else}\end{cases}$
x in l for constant list/set l	$\begin{cases}\textsc{fin}(\top) & \text{if t in l}\\ \textsc{var}(\bot) & \text{if } \exists e \in l\bullet\ \texttt{e.startswith}(vt)\\ \bot & \text{else}\end{cases}$
x < y	$[\![x]\!]_{\sigma[v\leftarrow vt]} < [\![y]\!]_{\sigma[v\leftarrow vt]}$
string comp. a == v	$\begin{cases}\textsc{fin}(\top) & \text{if } vt = \text{a}\\ \textsc{var}(\bot) & \text{if a.startswith}(vt)\\ \bot & \text{else}\end{cases}$
number comp. x == y	$[\![x]\!]_{\sigma[v\leftarrow vt]} = [\![y]\!]_{\sigma[v\leftarrow vt]}$
a and b	$[\![x]\!]_{\sigma[v\leftarrow vt]}$ and $[\![y]\!]_{\sigma[v\leftarrow vt]}$
a or b	$[\![x]\!]_{\sigma[v\leftarrow vt]}$ or $[\![y]\!]_{\sigma[v\leftarrow vt]}$
not a	not $[\![x]\!]_{\sigma[v\leftarrow vt]}$

Question can we define (simpler) and *complete* semantics?

Lecture 5






```python
@guidance(stateless=True)
def constant(lm):
    return lm + gen(regex='[a-u]', max_tokens=5)

@guidance(stateless=True)
def variable(lm):
    return lm + gen(regex='[v-z][0-9]*', max_tokens=5)

@guidance(stateless=True)
def term(lm):
    return lm + select([variable(), constant()])

@guidance(stateless=True)
def quantifier(lm):
    return lm + select(["∀", "∃"])

@guidance(stateless=True)
def unop(lm):
    return lm + "¬"

@guidance(stateless=True)
def binop(lm):
    return lm + select(["→", "↔", "∧", "∨"])
```

Unconstrained

Constrained

Unconstrained

```
Item 4431
Prm: Only 12 aliens ate at some restaurants.
Hyp: Only 12 aliens ate at Mexican restaurants.
preds1: Alien(x, y), Mexican(x, y)
P1: ∀x ∀y (Alien(x, y) → ∃z (Mexican(z, y) → ∀w (Alien(x, w) → ¬Mexican(z, w))))
H1: ∀x ∀y (Alien(x, y) → ∃z (Mexican(z, y) → ∀w (Alien(x, w) → ¬Mexican(z, w))))
preds2: Restaurants(x, y), Mexican(x, y)
P2: ∀x ∀y (Restaurants(x, y) ∧ Restaurants(y, x))
H2: ∀x ∀y (Restaurants(x, y) ∧ Mexican(y, x))
preds3: Restaurants(x, y), Mexican(x, y)
P3: ∀x ∀y (Restaurants(x, y) ∧ 12 ≤ ∃z (Restaurants(z, y)))
H3: ∀x ∀y (Restaurants(x, y) ∧ 12 ≤ ∃z (Mexican(z, y)))
```

Question Where does grammar-constrained decoding help NLP?

Course webpage

Lecturers

Kyle Richardson (Allen Institute for AI)

Gijs Wijnholds (Leiden Institute of Advanced Computer Science)

Slides

lecture 1: course overview, **language modeling basics**, RASP. extended notes on transformers

lecture 2: declarative approaches to **model training and fine-tuning**, the semantic loss and weighted model counting, other approaches. background logic notes

lecture 3: declarative and probabilistic approaches to **structured inference**, LLM self-correction.

lecture 4 LLM decoding, advanced prompting techniques, prompting is programming and LMQL.

Helpful Resources

Below are some pointers to code resources:

- **languages** [scallop], [problog], [pyDatalog], [lmql],[rasp], [NumPy Rasp], [deepproblog]
- **automated reasoning tools** [Z3 solver], [python-sat], [pysdd]
- **NLP and general ML** [transformers], [PyTorch], [pylon-lib], [hf datasets]
- **other useful utilities** [sympy]

`github.com/yakazimir/esslli_2024_llm_programming`

Thank you.

kyler@allenai.org
g.j.wijnholds@liacs.leidenuniv.nl

References I

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs the right way: Fast, non-invasive constrained generation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=pXaEYzrFae.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL https://aclanthology.org/D15-1075.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. XNLI: Evaluating cross-lingual sentence representations. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2475–2485, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1269. URL https://aclanthology.org/D18-1269.

References II

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/363_Paper.pdf.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.248. URL https://aclanthology.org/2023.findings-emnlp.248.

Gijs Wijnholds. Assessing monotonicity reasoning in Dutch through natural language inference. In Andreas Vlachos and Isabelle Augenstein, editors, *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1494–1500, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-eacl.110. URL https://aclanthology.org/2023.findings-eacl.110.

References III

Brandon T Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv e-prints*, pages arXiv–2307, 2023.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1101. URL https://aclanthology.org/N18-1101.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. Can neural networks understand monotonicity reasoning? In Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, editors, *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 31–40, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4804. URL https://aclanthology.org/W19-4804.