Related Work Presentation - Language Model Programming

Zilong Li & Scott McCall

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting Is Programming: A Query Language for Large Language Models. Proc. ACM Program. Lang. 7, PLDI, Article 186 (June 2023), 24 pages. https://doi.org/10.1145/3591300

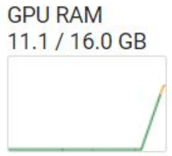
Background

- Large Language Models (LLMs) - natural language input is subdivided into a set of tokens Response is based on probability of subsequent tokens (usually, the highest one).
- Trained on a large scale of data. Can be used for downstream tasks such as code generation, summarizing text, answering questions, and more
- Training LLM to perform a specific task has two different approaches: fine-tuning and prompting

Background (cont.)

Fine-tuning: training the model on a specific dataset relevant to the task

- Prohibitively expensive for most users in terms of computing power
- Billions of parameters, not always clear how to fine tune
- Not possible on closed-source models such as GPT



Memory use of roberta-base (a small size model)

Prompting: natural language prompts as a method of instruction

- User may need to understand various differences between models to interact with them effectively
- LLMs may require back and forth interaction, limiting the generality of the types of prompts that can be provided

Motivation

As usage of LLMs such as ChatGPT increase, having way to interact with various models in an identical fashion would be extremely useful, user wouldn't have to learn model internals or implementation details.

Novel idea called language model programming with three primary goals:

- **Interaction**. Automatically interact with LLMs.
- **Constraint.** Impose constraints on LLMs.
- **Efficiency and Cost.** Reduce the computational and financial cost of calling language models.

Challenges with LMP (Language Model Programming)

- Providing an abstract, generalized interface to the user for prompting that can work with the specific implementation details of different models
- Providing decoding methods that limit the chance of combinatorial explosions and high query costs
- Decoding LLMs refers to the methodology of predicting most next probable token or sequence of tokens in a prompt

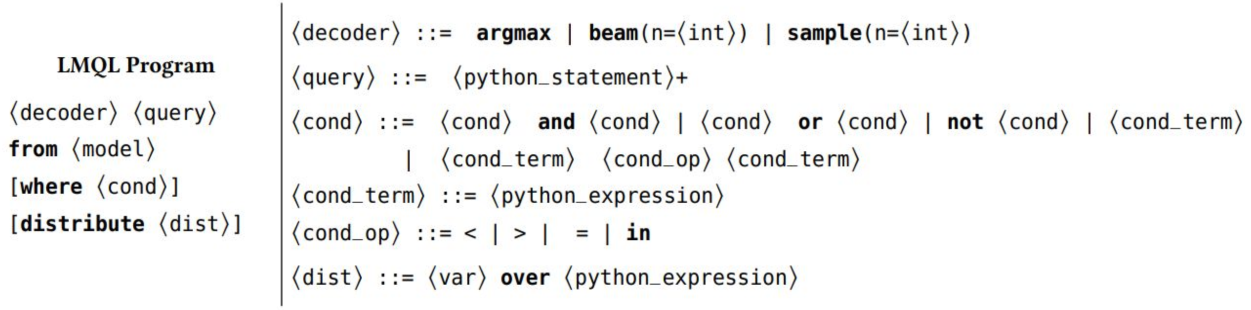
Syntax of LMQL (Language Model Query Language)

- ⟨decoder⟩ denotes the decoding procedure employed by LMQL.

Argmax, sample and beam are supported here.

- ⟨query⟩ models the interaction with the language model.

It can be thought of as the body of a python function subject to some restrictions and additions.



LMQL Program

```
⟨decoder⟩ ⟨query⟩
from ⟨model⟩
[where ⟨cond⟩]
[distribute ⟨dist⟩]
```

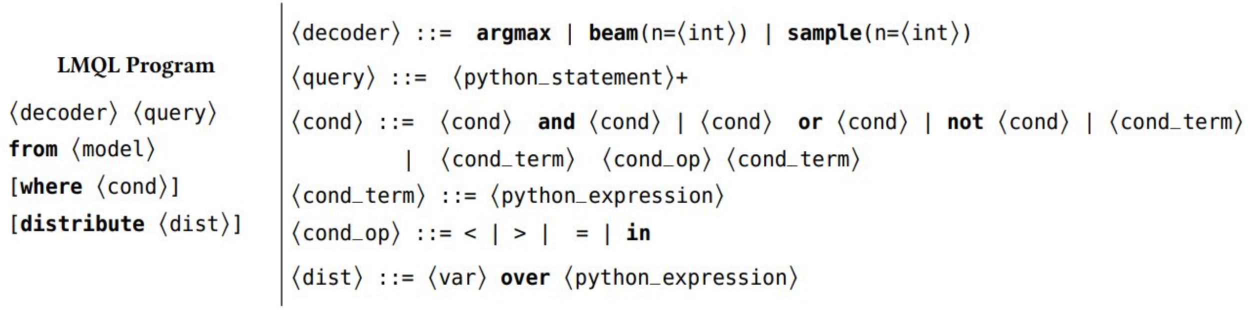
```
⟨decoder⟩ ::=  argmax | beam(n=⟨int⟩) | sample(n=⟨int⟩)
⟨query⟩ ::=  ⟨python_statement⟩+
⟨cond⟩ ::=  ⟨cond⟩  and ⟨cond⟩ | ⟨cond⟩  or ⟨cond⟩ | not ⟨cond⟩ | ⟨cond_term⟩
         |  ⟨cond_term⟩  ⟨cond_op⟩ ⟨cond_term⟩
⟨cond_term⟩ ::= ⟨python_expression⟩
⟨cond_op⟩ ::= < | > |  = | in
⟨dist⟩ ::= ⟨var⟩ over ⟨python_expression⟩
```

Syntax of LMQL (Language Model Query Language)

- from ⟨model⟩ denotes which LM to use.

We can use the Hugging Face Model repository or the GPT family.

- where ⟨condition⟩ places constraints on the variables in the ⟨query⟩.
- distribute ⟨dist⟩ is an optional instruction that returns the probability distribution of the last variable in the query.

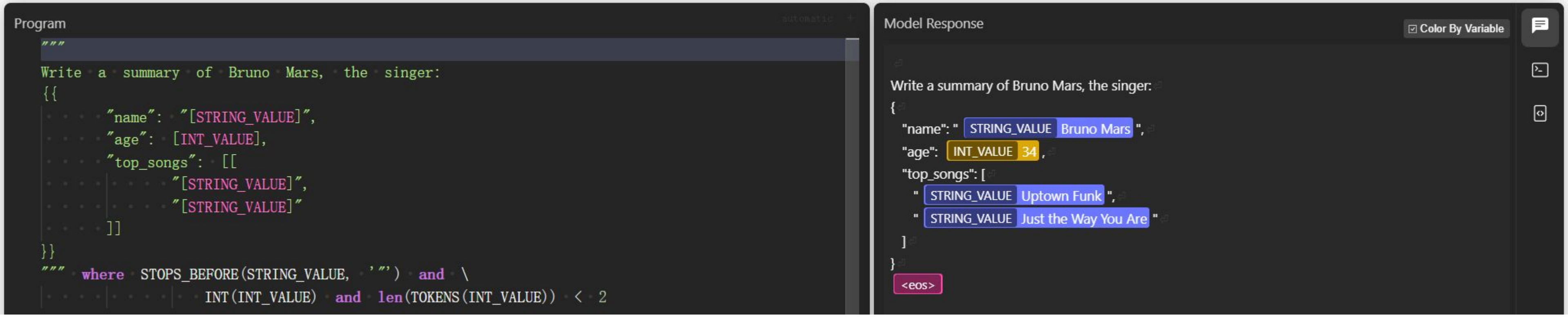


LMQL Program

```
⟨decoder⟩ ⟨query⟩
from ⟨model⟩
[where ⟨cond⟩]
[distribute ⟨dist⟩]
```

```
⟨decoder⟩ ::=  argmax | beam(n=⟨int⟩) | sample(n=⟨int⟩)
⟨query⟩ ::=  ⟨python_statement⟩+
⟨cond⟩ ::=  ⟨cond⟩  and ⟨cond⟩ | ⟨cond⟩  or ⟨cond⟩ | not ⟨cond⟩ | ⟨cond_term⟩
          |  ⟨cond_term⟩  ⟨cond_op⟩ ⟨cond_term⟩
⟨cond_term⟩ ::= ⟨python_expression⟩
⟨cond_op⟩ ::= < | > |  = | in
⟨dist⟩ ::= ⟨var⟩ over ⟨python_expression⟩
```

LMQL Example



Program

```
"""
Write a summary of Bruno Mars, the singer:
{{
    "name": "[STRING_VALUE]",
    "age": [INT_VALUE],
    "top_songs": [[
        "[STRING_VALUE]",
        "[STRING_VALUE]"
    ]]
}}
""" where STOPS_BEFORE(STRING_VALUE, '"') and \
            INT(INT_VALUE) and len(TOKENS(INT_VALUE)) < 2
```

Model Response

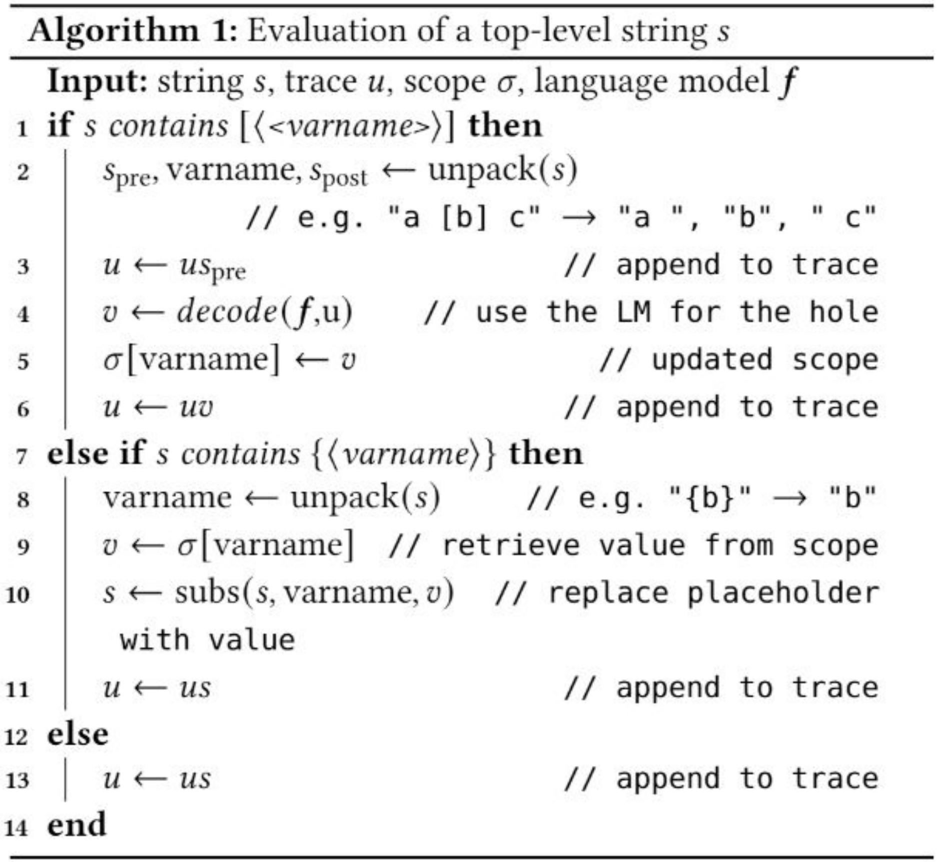
```
Write a summary of Bruno Mars, the singer:
{
  "name": " STRING_VALUE Bruno Mars ",
  "age": INT_VALUE 34 ,
  "top_songs": [
    " STRING_VALUE Uptown Funk ",
    " STRING_VALUE Just the Way You Are "
  ]
}
<eos>
```

LMQL Runtime

Execution of ⟨query⟩ body

Two kinds of variables:

- [variable] - decode and assign
- {variable} - retrieve/substitute



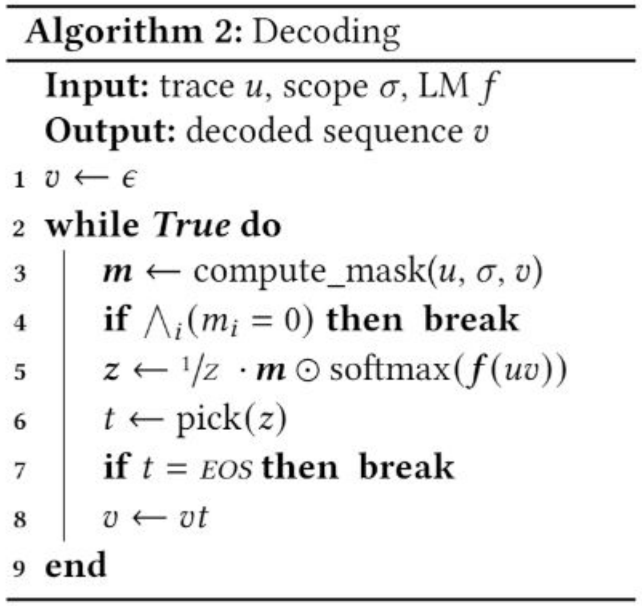
Algorithm 1: Evaluation of a top-level string s

```
Input: string $s$, trace $u$, scope $\sigma$, language model $f$
if $s$ contains [⟨<varname>⟩] then
    $s_{pre}$, varname, $s_{post} \leftarrow$ unpack($s$)
                        // e.g. "a [b] c" → "a ", "b", " c"
    $u \leftarrow us_{pre}$                          // append to trace
    $v \leftarrow decode(f,u)$            // use the LM for the hole
    $\sigma$[varname] $\leftarrow v$                       // updated scope
    $u \leftarrow uv$                           // append to trace
else if $s$ contains {⟨varname⟩} then
    varname $\leftarrow$ unpack($s$)        // e.g. "{b}" → "b"
    $v \leftarrow \sigma$[varname]   // retrieve value from scope
    $s \leftarrow$ subs($s$, varname, $v$)   // replace placeholder
     with value
    $u \leftarrow us$                           // append to trace
else
    $u \leftarrow us$                           // append to trace
end
```

LMQL Runtime

Decoding of language model

Autoregressively generating token



Algorithm 2: Decoding

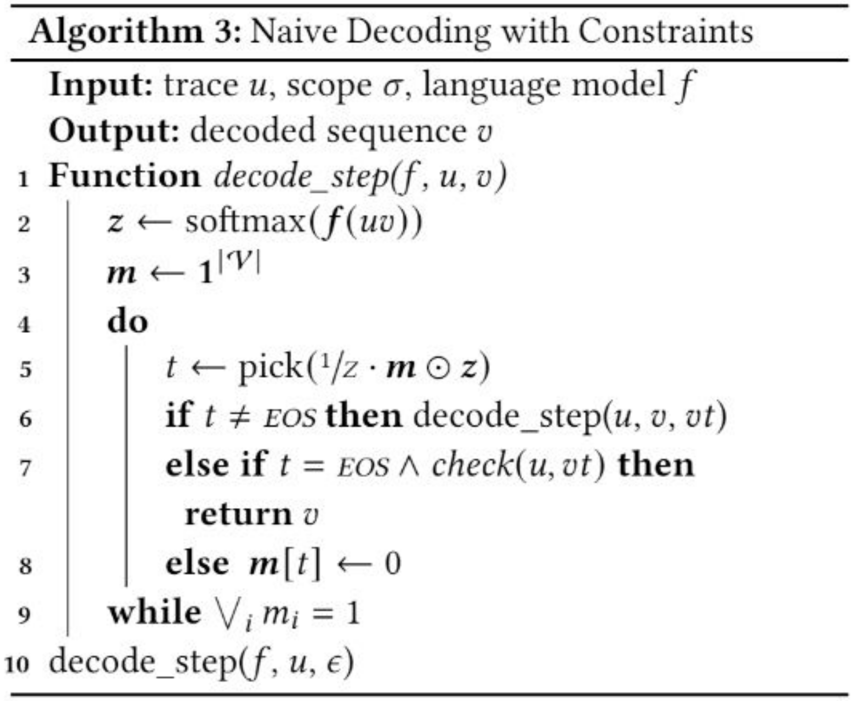
```
Input: trace $u$, scope $\sigma$, LM $f$
Output: decoded sequence $v$
$v \leftarrow \epsilon$
while True do
    $\boldsymbol{m} \leftarrow \text{compute\_mask}(u, \sigma, v)$
    if $\bigwedge_i (m_i = 0)$ then break
    $\boldsymbol{z} \leftarrow {}^1/_Z \cdot \boldsymbol{m} \odot \text{softmax}(f(uv))$
    $t \leftarrow \text{pick}(\boldsymbol{z})$
    if $t = \text{EOS}$ then break
    $v \leftarrow vt$
end
```

LMQL Runtime

Execution of ⟨query⟩ body with the constraint from **where**

- Naïve: not strong enough

- Go to the end of sentence (EOS)
- Explosion of function call
- Explosion of cost



Algorithm 3: Naive Decoding with Constraints

```
Input: trace $u$, scope $\sigma$, language model $f$
Output: decoded sequence $v$
Function decode_step($f$, $u$, $v$)
    $z \leftarrow \mathrm{softmax}(f(uv))$
    $m \leftarrow \mathbf{1}^{|\mathcal{V}|}$
    do
        $t \leftarrow \mathrm{pick}(1/Z \cdot m \odot z)$
        if $t \neq$ EOS then decode_step($u$, $v$, $vt$)
        else if $t =$ EOS $\wedge$ check($u$, $vt$) then
            return $v$
        else $m[t] \leftarrow 0$
    while $\bigvee_i m_i = 1$
decode_step($f$, $u$, $\epsilon$)
```

Final Semantics

Assign annotators below to queries and constraints

FIN - fixed, unchanging

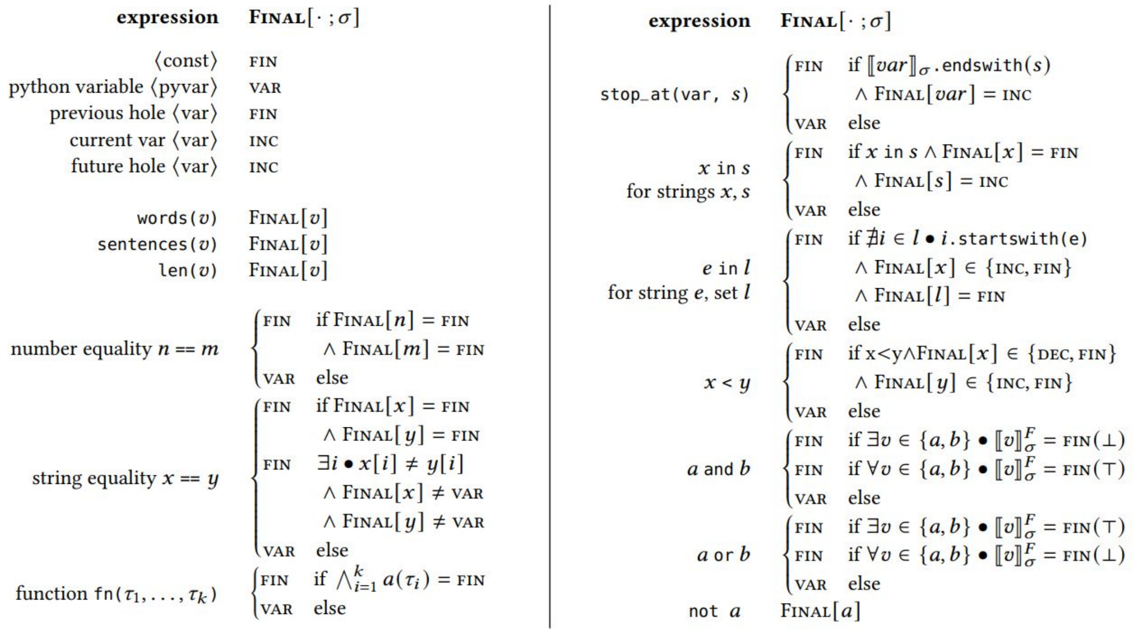
VAR - variable

INC – incrementing

DEC - decreasing

⊤: True

⊥: False



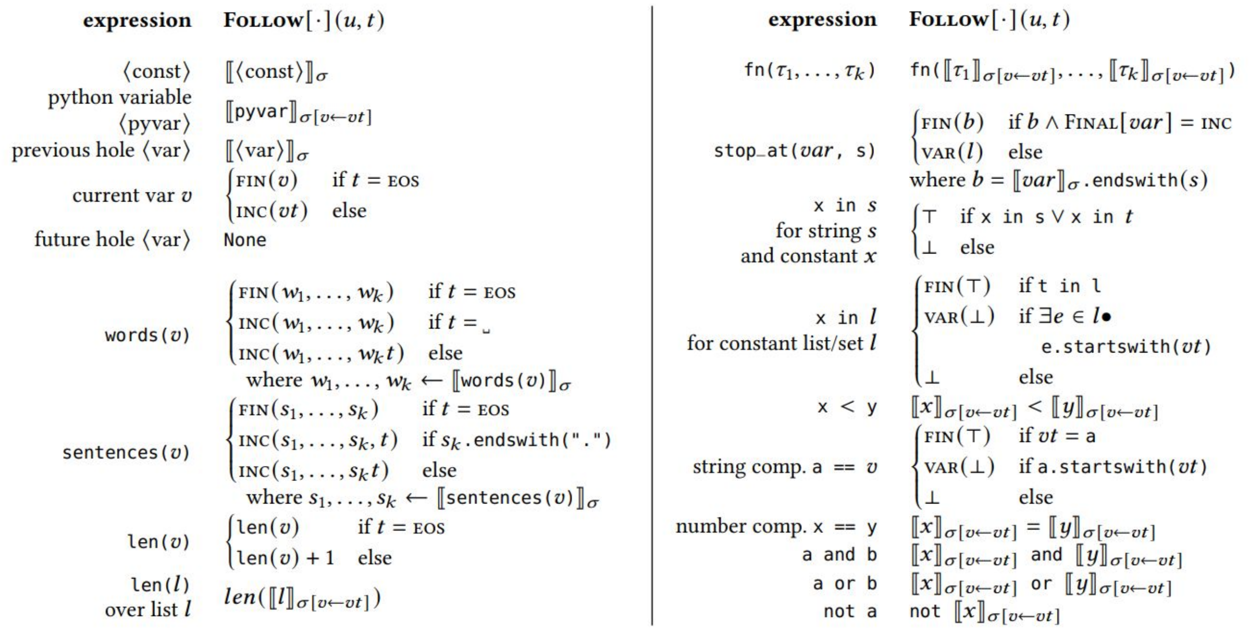
expression	$\textsc{Final}[\cdot\,;\sigma]$
⟨const⟩	FIN
python variable ⟨pyvar⟩	VAR
previous hole ⟨var⟩	FIN
current var ⟨var⟩	INC
future hole ⟨var⟩	INC
`words`(v)	$\textsc{Final}[v]$
`sentences`(v)	$\textsc{Final}[v]$
`len`(v)	$\textsc{Final}[v]$
number equality $n == m$	$\begin{cases} \text{FIN} & \text{if } \textsc{Final}[n] = \text{FIN} \\ & \wedge\ \textsc{Final}[m] = \text{FIN} \\ \text{VAR} & \text{else} \end{cases}$
string equality $x == y$	$\begin{cases} \text{FIN} & \text{if } \textsc{Final}[x] = \text{FIN} \\ & \wedge\ \textsc{Final}[y] = \text{FIN} \\ \text{FIN} & \exists i \bullet x[i] \neq y[i] \\ & \wedge\ \textsc{Final}[x] \neq \text{VAR} \\ & \wedge\ \textsc{Final}[y] \neq \text{VAR} \\ \text{VAR} & \text{else} \end{cases}$
function `fn`$(\tau_1, \ldots, \tau_k)$	$\begin{cases} \text{FIN} & \text{if } \bigwedge_{i=1}^{k} a(\tau_i) = \text{FIN} \\ \text{VAR} & \text{else} \end{cases}$

expression	$\textsc{Final}[\cdot\,;\sigma]$
`stop_at(var,` s`)`	$\begin{cases} \text{FIN} & \text{if } [\![var]\!]_\sigma\texttt{.endswith}(s) \\ & \wedge\ \textsc{Final}[var] = \text{INC} \\ \text{VAR} & \text{else} \end{cases}$
x `in` s for strings x, s	$\begin{cases} \text{FIN} & \text{if } x \texttt{ in } s \wedge \textsc{Final}[x] = \text{FIN} \\ & \wedge\ \textsc{Final}[s] = \text{INC} \\ \text{VAR} & \text{else} \end{cases}$
e `in` l for string e, set l	$\begin{cases} \text{FIN} & \text{if } \nexists i \in l \bullet i\texttt{.startswith(e)} \\ & \wedge\ \textsc{Final}[x] \in \{\text{INC}, \text{FIN}\} \\ & \wedge\ \textsc{Final}[l] = \text{FIN} \\ \text{VAR} & \text{else} \end{cases}$
$x < y$	$\begin{cases} \text{FIN} & \text{if x<y} \wedge \textsc{Final}[x] \in \{\text{DEC}, \text{FIN}\} \\ & \wedge\ \textsc{Final}[y] \in \{\text{INC}, \text{FIN}\} \\ \text{VAR} & \text{else} \end{cases}$
a `and` b	$\begin{cases} \text{FIN} & \text{if } \exists v \in \{a, b\} \bullet [\![v]\!]_\sigma^F = \text{FIN}(\bot) \\ \text{FIN} & \text{if } \forall v \in \{a, b\} \bullet [\![v]\!]_\sigma^F = \text{FIN}(\top) \\ \text{VAR} & \text{else} \end{cases}$
a `or` b	$\begin{cases} \text{FIN} & \text{if } \exists v \in \{a, b\} \bullet [\![v]\!]_\sigma^F = \text{FIN}(\top) \\ \text{FIN} & \text{if } \forall v \in \{a, b\} \bullet [\![v]\!]_\sigma^F = \text{FIN}(\bot) \\ \text{VAR} & \text{else} \end{cases}$
`not` a	$\textsc{Final}[a]$

FollowMap Semantics

Evaluation rules for followmap semantics

Check whether the constraint is met.



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

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

LMQL Runtime

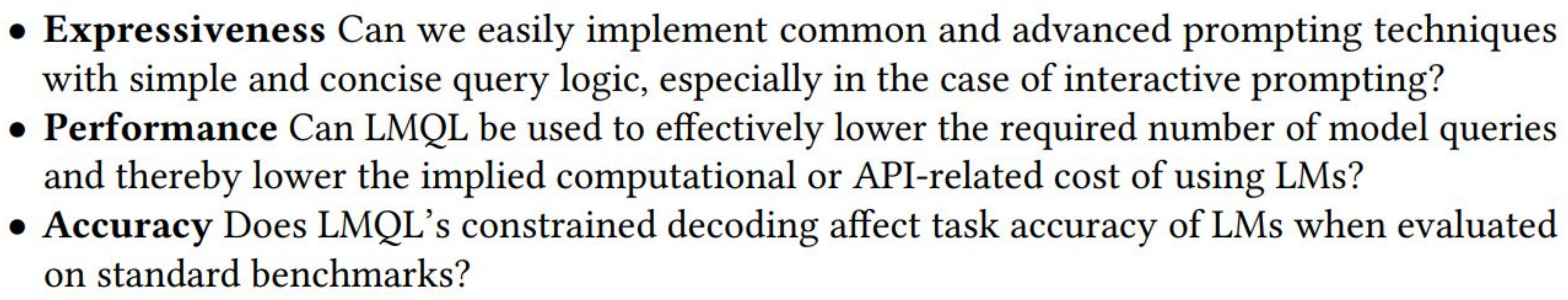
This is an example of the followmap semantics

$$\textsc{Follow}[\texttt{TEXT in ["Stephen Hawking"]}](\texttt{"Steph"}, t) = \begin{cases} \texttt{FIN}(\top) & \text{if } t = \texttt{"en Hawking"} \\ \texttt{FIN}(\bot) & \text{else} \end{cases}$$

LMQL Evaluation

Evaluate the effectiveness of LMQL as a language as well as a tool for prompt engineers.

Evaluation standard:



- **Expressiveness** Can we easily implement common and advanced prompting techniques with simple and concise query logic, especially in the case of interactive prompting?
- **Performance** Can LMQL be used to effectively lower the required number of model queries and thereby lower the implied computational or API-related cost of using LMs?
- **Accuracy** Does LMQL's constrained decoding affect task accuracy of LMs when evaluated on standard benchmarks?

LMQL Evaluation

- Baseline: generate() function in many huggingface language models.

  generate() function generates output chunk-wise

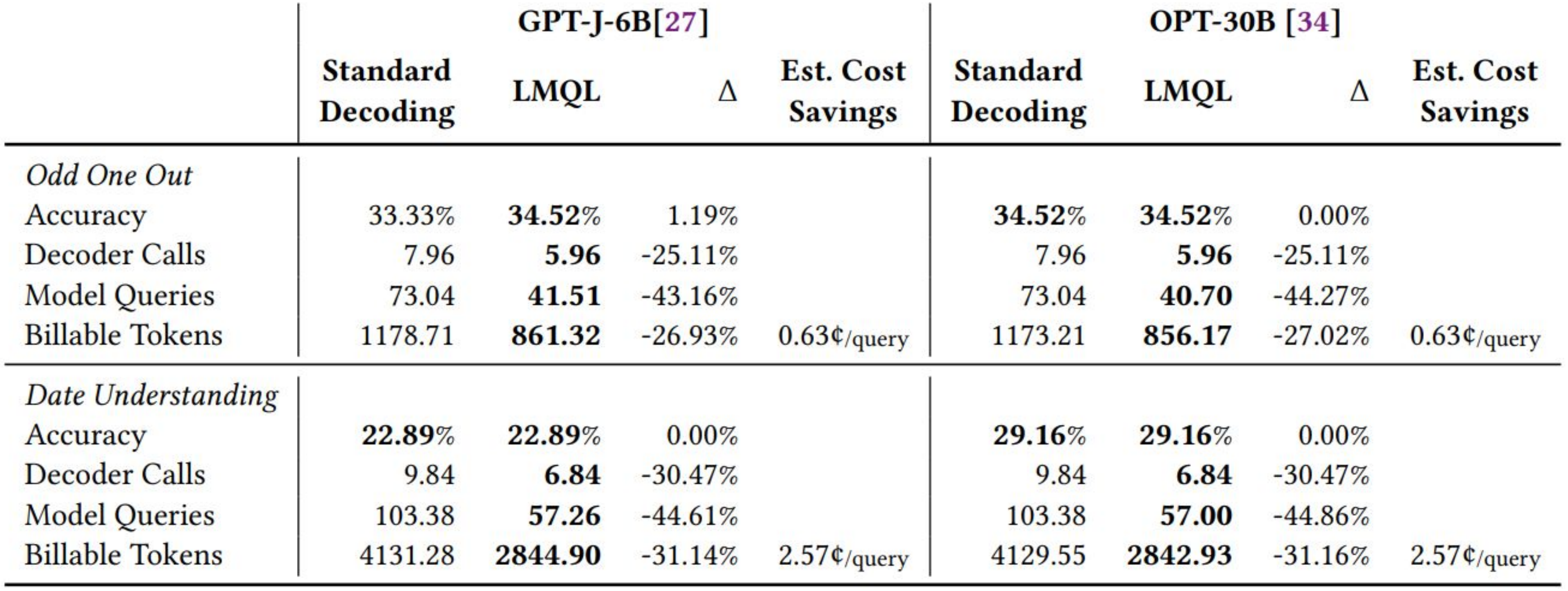
  do parsing and validation manually
- Dataset: general and date understanding, question answering and arithmetic math
- Model: GPT-J 6B, OPT 30B, GPT-2

LMQL Evaluation

Evaluation Metrics:

- LOC: the number of functional lines of code
- Number of Model Queries: the number of times the model f is invoked for next
- Number of Decoder Calls: the number of times a new decoding loop is started
- Billable Tokens: count the number of tokens per Decoder Call plus the number of tokens that are generated.

Results



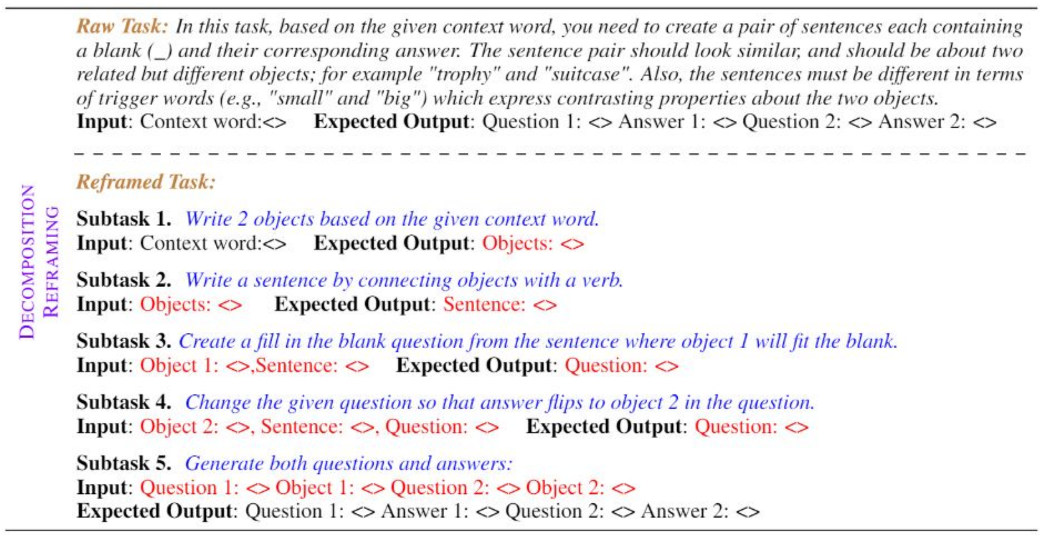
	GPT-J-6B[27]				OPT-30B [34]			
	Standard Decoding	**LMQL**	**Δ**	**Est. Cost Savings**	**Standard Decoding**	**LMQL**	**Δ**	**Est. Cost Savings**
Odd One Out								
Accuracy	33.33%	**34.52%**	1.19%		**34.52%**	**34.52%**	0.00%	
Decoder Calls	7.96	**5.96**	-25.11%		7.96	**5.96**	-25.11%	
Model Queries	73.04	**41.51**	-43.16%		73.04	**40.70**	-44.27%	
Billable Tokens	1178.71	**861.32**	-26.93%	0.63¢/query	1173.21	**856.17**	-27.02%	0.63¢/query
Date Understanding								
Accuracy	**22.89%**	**22.89%**	0.00%		**29.16%**	**29.16%**	0.00%	
Decoder Calls	9.84	**6.84**	-30.47%		9.84	**6.84**	-30.47%	
Model Queries	103.38	**57.26**	-44.61%		103.38	**57.00**	-44.86%	
Billable Tokens	4131.28	**2844.90**	-31.14%	2.57¢/query	4129.55	**2842.93**	-31.16%	2.57¢/query

Contributions Summary

- Novel paradigm of language model programming, addressing several challenges that arise with recent LM prompting techniques
- LMQL, an efficient, high-level query language for LMs with support for scripted prompting and output constraining.
- A formal model of eager, partial evaluation semantics based on so-called final and follow abstractions.
- Reduction in inference cost and latency by 26-80% with retention or slight improvement on task accuracy.

Limitations and Future Work

(1) Task Decomposition and Chain of Thought
Decomposing a task and solving it step by step can improve the performance of LLMs.


DECOMPOSITION
REFRAMING

Raw Task: *In this task, based on the given context word, you need to create a pair of sentences each containing a blank (_) and their corresponding answer. The sentence pair should look similar, and should be about two related but different objects; for example "trophy" and "suitcase". Also, the sentences must be different in terms of trigger words (e.g., "small" and "big") which express contrasting properties about the two objects.*
Input: Context word:<> **Expected Output**: Question 1: <> Answer 1: <> Question 2: <> Answer 2: <>

Reframed Task:

Subtask 1. *Write 2 objects based on the given context word.*
Input: Context word:<> **Expected Output**: Objects: <>

Subtask 2. *Write a sentence by connecting objects with a verb.*
Input: Objects: <> **Expected Output**: Sentence: <>

Subtask 3. *Create a fill in the blank question from the sentence where object 1 will fit the blank.*
Input: Object 1: <>,Sentence: <> **Expected Output**: Question: <>

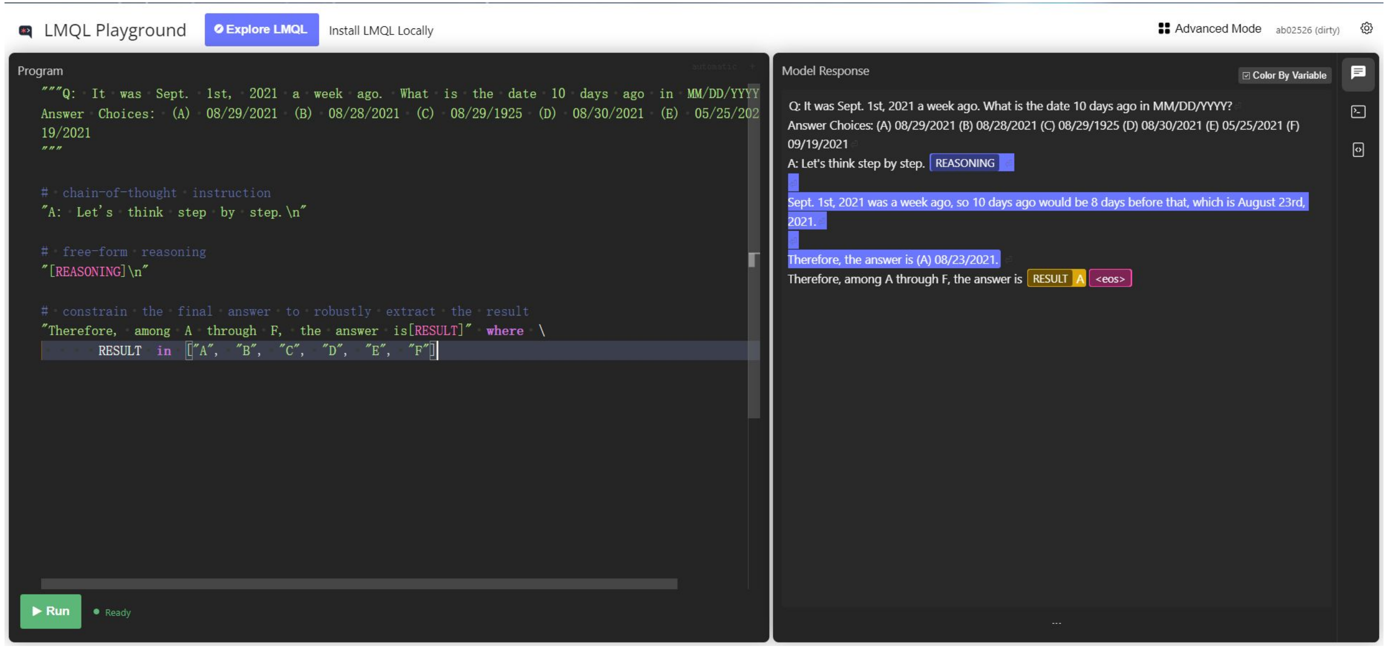
Subtask 4. *Change the given question so that answer flips to object 2 in the question.*
Input: Object 2: <>, Sentence: <>, Question: <> **Expected Output**: Question: <>

Subtask 5. *Generate both questions and answers:*
Input: Question 1: <> Object 1: <> Question 2: <> Object 2: <>
Expected Output: Question 1: <> Answer 1: <> Question 2: <> Answer 2: <>

Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. 2022. Reframing Instructional Prompts to GPTk's Language.

Limitations and Future Work

The chain of thought in LMQL is implemented by an instruction, not currently formalized in the language.



```
"""Q: It was Sept. 1st, 2021 a week ago. What is the date 10 days ago in MM/DD/YYYY
Answer Choices: (A) 08/29/2021 (B) 08/28/2021 (C) 08/29/1925 (D) 08/30/2021 (E) 05/25/202
19/2021
"""

# chain-of-thought instruction
"A: Let's think step by step.\n"

# free-form reasoning
"[REASONING]\n"

# constrain the final answer to robustly extract the result
"Therefore, among A through F, the answer is[RESULT]" where \
        RESULT in ["A", "B", "C", "D", "E", "F"]
```

Q: It was Sept. 1st, 2021 a week ago. What is the date 10 days ago in MM/DD/YYYY?
Answer Choices: (A) 08/29/2021 (B) 08/28/2021 (C) 08/29/1925 (D) 08/30/2021 (E) 05/25/2021 (F) 09/19/2021
A: Let's think step by step. REASONING

Sept. 1st, 2021 was a week ago, so 10 days ago would be 8 days before that, which is August 23rd, 2021.

Therefore, the answer is (A) 08/23/2021.
Therefore, among A through F, the answer is RESULT A <eos>

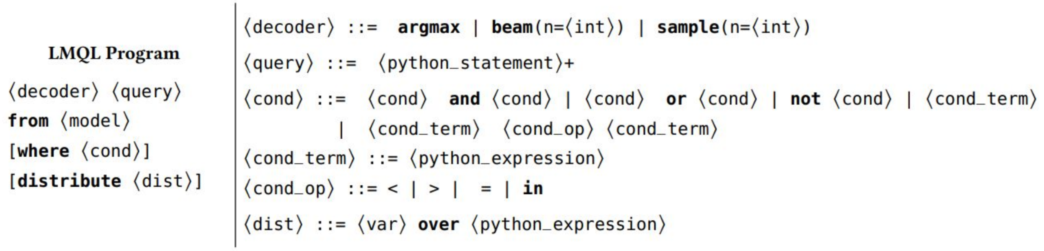
Future Work

A query can use another query's output as its input.
Different queries can have difference constraints.

A Stack of Queries

Limitations and Future Work

(2) The “or” Operator is Redundant here



```
LMQL Program

⟨decoder⟩ ⟨query⟩
from ⟨model⟩
[where ⟨cond⟩]
[distribute ⟨dist⟩]
```

```
⟨decoder⟩ ::=  argmax | beam(n=⟨int⟩) | sample(n=⟨int⟩)
⟨query⟩ ::=  ⟨python_statement⟩+
⟨cond⟩ ::=  ⟨cond⟩  and ⟨cond⟩ | ⟨cond⟩  or ⟨cond⟩ | not ⟨cond⟩ | ⟨cond_term⟩
          |  ⟨cond_term⟩  ⟨cond_op⟩ ⟨cond_term⟩
⟨cond_term⟩ ::= ⟨python_expression⟩
⟨cond_op⟩ ::= < | > |  = | in
⟨dist⟩ ::= ⟨var⟩ over ⟨python_expression⟩
```

The goal of LMQL:
Control the flow of language models and constrain the response of language models

… [variable1] …… [variable2] …… [variable3] …

where (variable1 … or variable3 …) and variable2 …

Limitations and Future Work

The “and” doesn’t have such a problem:

… [variable1] …… [variable2] …… [variable3] …

where variable1 … and variable2 … and variable3 …

Solutions:

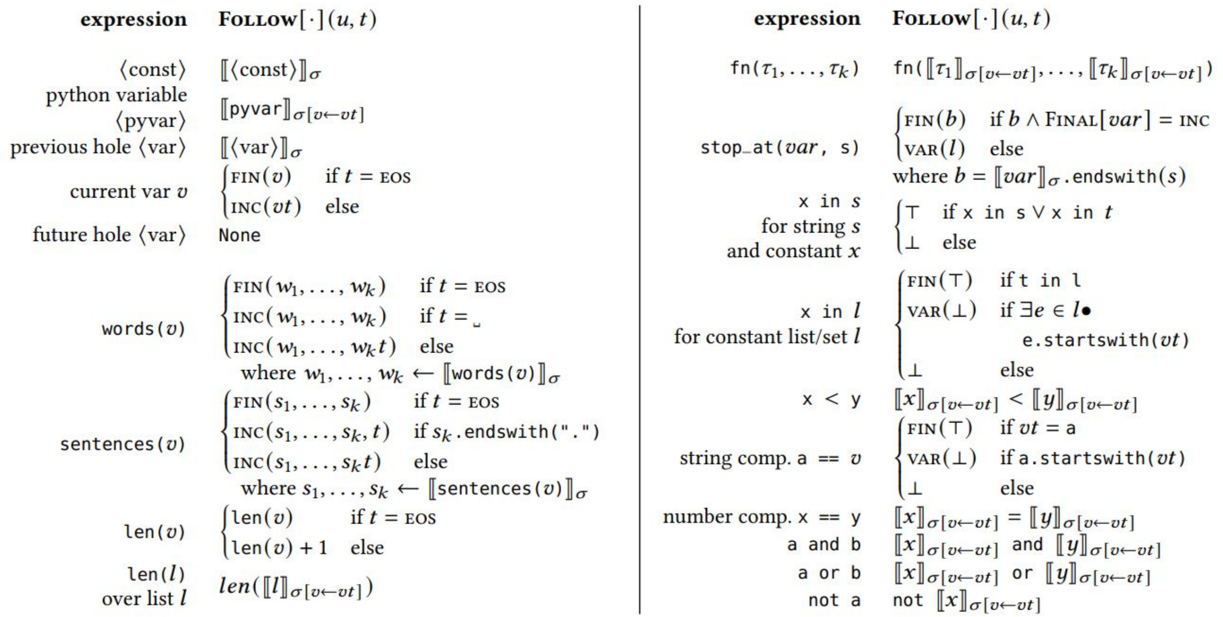
- Abolish the “or”.
- Limit the use of “or”.

Also, [variable1] == [variable2] has the same problem.

Limitations and Future Work

(3) Inconsistency between the FollowMap Semantics and the Language Implementation

Check the constraints token by token.

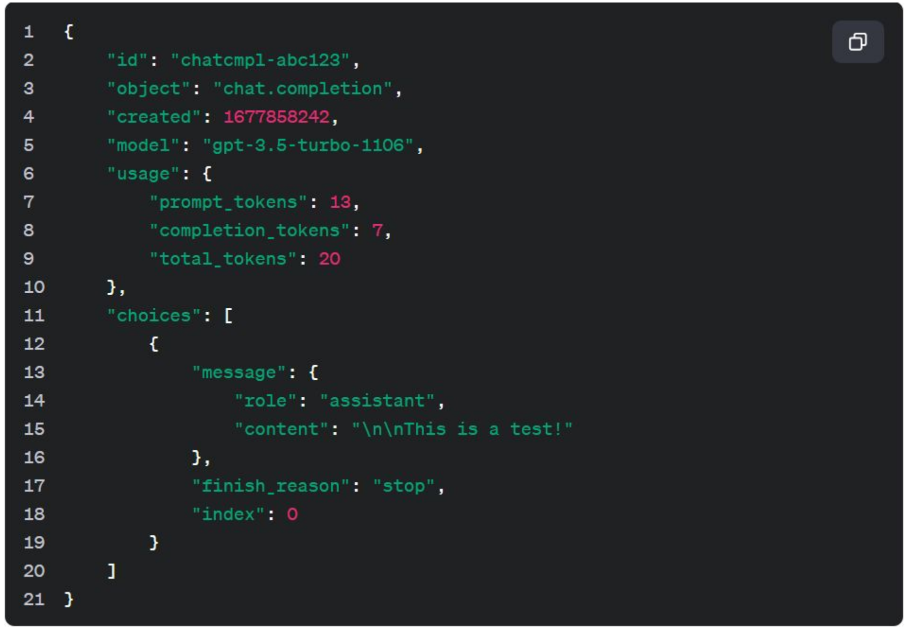


expression	Follow$[\cdot](u, t)$
$\langle$const$\rangle$	$\llbracket \langle \text{const} \rangle \rrbracket_\sigma$
python variable $\langle$pyvar$\rangle$	$\llbracket \texttt{pyvar} \rrbracket_{\sigma[v \leftarrow vt]}$
previous hole $\langle$var$\rangle$	$\llbracket \langle \text{var} \rangle \rrbracket_\sigma$
current var v	$\begin{cases} \text{FIN}(v) & \text{if } t = \text{EOS} \\ \text{INC}(vt) & \text{else} \end{cases}$
future hole $\langle$var$\rangle$	None
`words(`v`)`	$\begin{cases} \text{FIN}(w_1, \ldots, w_k) & \text{if } t = \text{EOS} \\ \text{INC}(w_1, \ldots, w_k) & \text{if } t = ␣ \\ \text{INC}(w_1, \ldots, w_k t) & \text{else} \end{cases}$ where $w_1, \ldots, w_k \leftarrow \llbracket \texttt{words}(v) \rrbracket_\sigma$
`sentences(`v`)`	$\begin{cases} \text{FIN}(s_1, \ldots, s_k) & \text{if } t = \text{EOS} \\ \text{INC}(s_1, \ldots, s_k, t) & \text{if } s_k\texttt{.endswith(".")} \\ \text{INC}(s_1, \ldots, s_k t) & \text{else} \end{cases}$ where $s_1, \ldots, s_k \leftarrow \llbracket \texttt{sentences}(v) \rrbracket_\sigma$
`len(`v`)`	$\begin{cases} \texttt{len}(v) & \text{if } t = \text{EOS} \\ \texttt{len}(v) + 1 & \text{else} \end{cases}$
`len(`l`)` over list l	$len(\llbracket l \rrbracket_{\sigma[v \leftarrow vt]})$
`fn(`$\tau_1, \ldots, \tau_k$`)`	$\texttt{fn}(\llbracket \tau_1 \rrbracket_{\sigma[v \leftarrow vt]}, \ldots, \llbracket \tau_k \rrbracket_{\sigma[v \leftarrow vt]})$
`stop_at(`var`, s)`	$\begin{cases} \text{FIN}(b) & \text{if } b \wedge \text{Final}[var] = \text{INC} \\ \text{VAR}(l) & \text{else} \end{cases}$ where $b = \llbracket var \rrbracket_\sigma\texttt{.endswith}(s)$
`x in` s for string s and constant x	$\begin{cases} \top & \text{if } \texttt{x in s} \vee \texttt{x in } t \\ \bot & \text{else} \end{cases}$
`x in` l for constant list/set l	$\begin{cases} \text{FIN}(\top) & \text{if } \texttt{t in l} \\ \text{VAR}(\bot) & \text{if } \exists e \in l \bullet \texttt{e.startswith}(vt) \\ \bot & \text{else} \end{cases}$
`x < y`	$\llbracket x \rrbracket_{\sigma[v \leftarrow vt]} < \llbracket y \rrbracket_{\sigma[v \leftarrow vt]}$
string comp. `a ==` v	$\begin{cases} \text{FIN}(\top) & \text{if } vt = \texttt{a} \\ \text{VAR}(\bot) & \text{if } \texttt{a.startswith}(vt) \\ \bot & \text{else} \end{cases}$
number comp. `x == y`	$\llbracket x \rrbracket_{\sigma[v \leftarrow vt]} = \llbracket y \rrbracket_{\sigma[v \leftarrow vt]}$
`a and b`	$\llbracket x \rrbracket_{\sigma[v \leftarrow vt]}$ `and` $\llbracket y \rrbracket_{\sigma[v \leftarrow vt]}$
`a or b`	$\llbracket x \rrbracket_{\sigma[v \leftarrow vt]}$ `or` $\llbracket y \rrbracket_{\sigma[v \leftarrow vt]}$
`not a`	`not` $\llbracket x \rrbracket_{\sigma[v \leftarrow vt]}$

Limitations and Future Work

Most languages models have the generate() function, but GPT API's do not.

- If we evaluate token by token, it will increase the cost.
- They do not use these rules when calling the GPT APIs.
- This is limitation of GPT APIs. It cannot be resolved.



```
{
    "id": "chatcmpl-abc123",
    "object": "chat.completion",
    "created": 1677858242,
    "model": "gpt-3.5-turbo-1106",
    "usage": {
        "prompt_tokens": 13,
        "completion_tokens": 7,
        "total_tokens": 20
    },
    "choices": [
        {
            "message": {
                "role": "assistant",
                "content": "\n\nThis is a test!"
            },
            "finish_reason": "stop",
            "index": 0
        }
    ]
}
```

Citations

- Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting Is Programming: A Query Language for Large Language Models. Proc. ACM Program. Lang. 7, PLDI, Article 186 (June 2023), 24 pages. https://doi.org/10.1145/3591300

- Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. 2022. Reframing Instructional Prompts to GPTk's Language.