Prompting Is Programming: A Query Language for Large Language Models

Author: Luca et al.

Presentor: Dongjae Lee

20240514

Luca Beurer-Kellner, Marc Fischer, & Martin T. Vechev (2023). Prompting Is Programming: A Query Language for Large Language Models. Proc. ACM Program. Lang., 7(PLDI), 1946–1969.

Problem of interaction with language model

- Human in the loop!
    - Grunt work

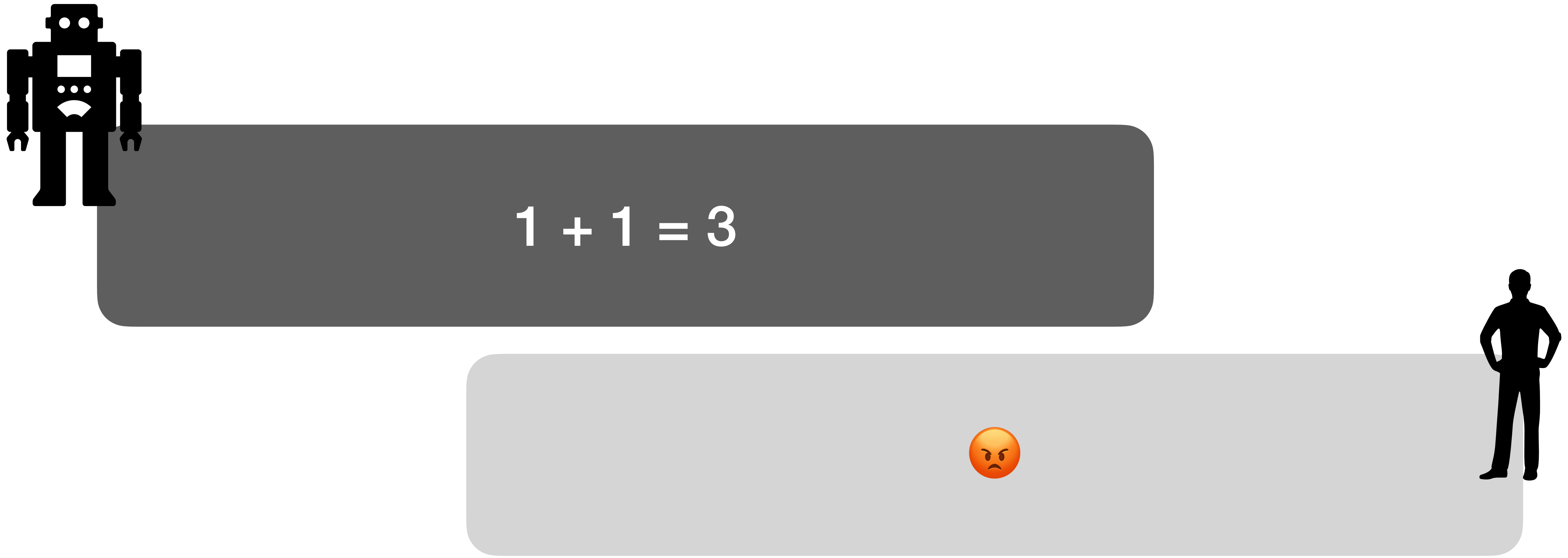
Problem of interaction with language model

- Human in the loop!
    - Grunt work

1 + 1 = 3

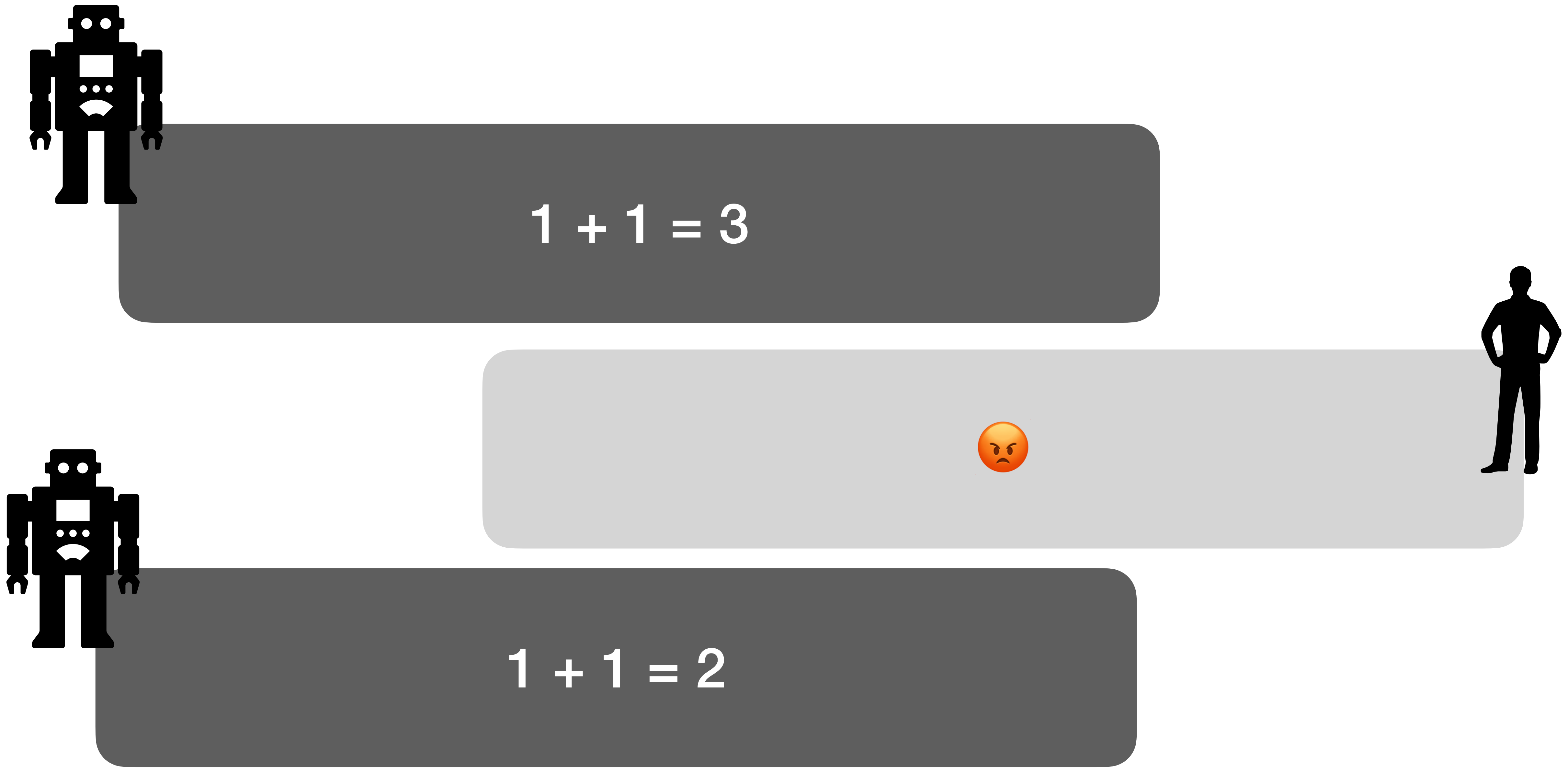
Problem of interaction with language model

- Human in the loop!
  - Grunt work



Problem of interaction with language model

- Human in the loop!
    - Grunt work



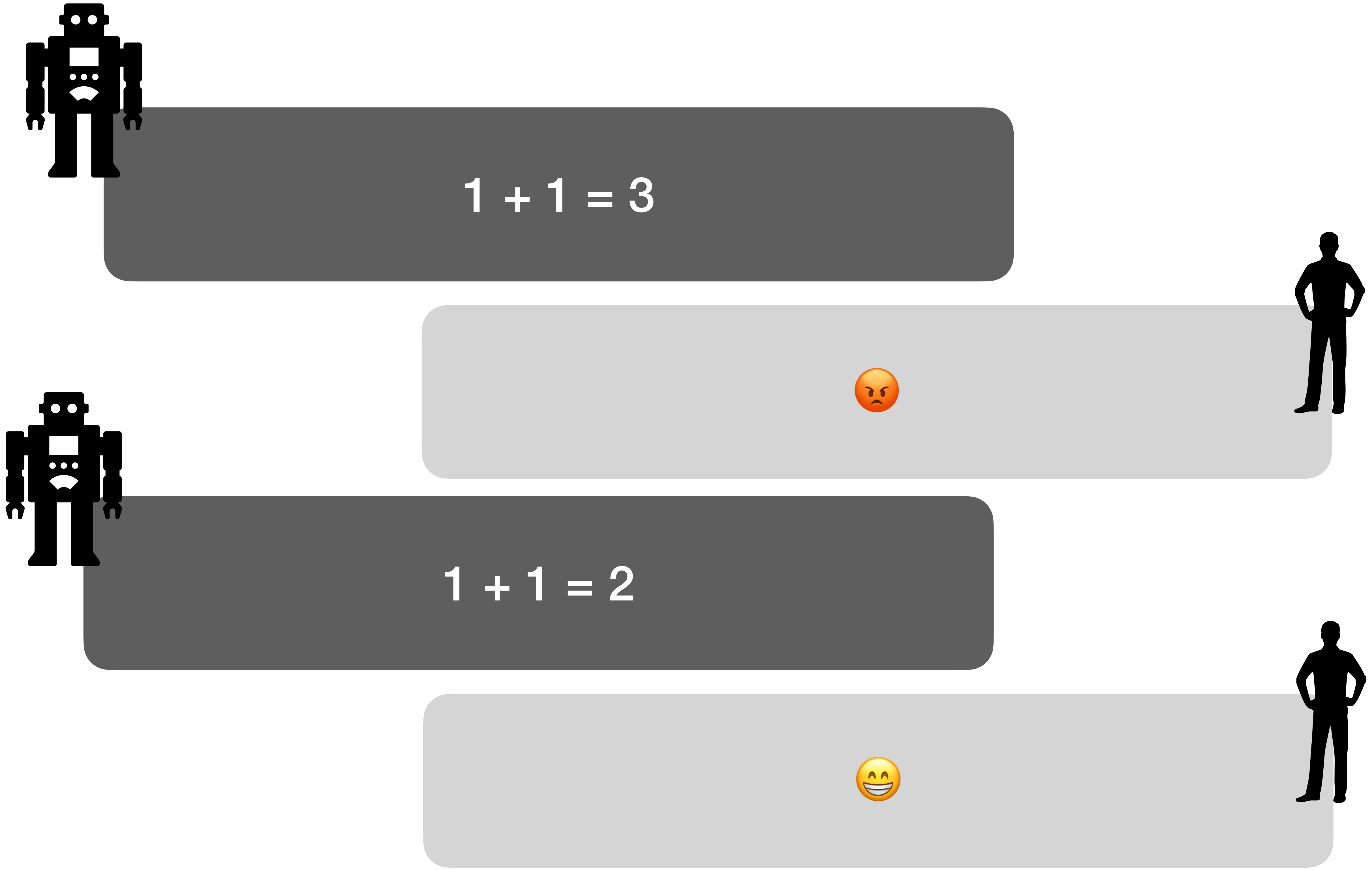
1 + 1 = 3

😡

1 + 1 = 2

Problem of interaction with language model

- Human in the loop!
  - Grunt work



1 + 1 = 3

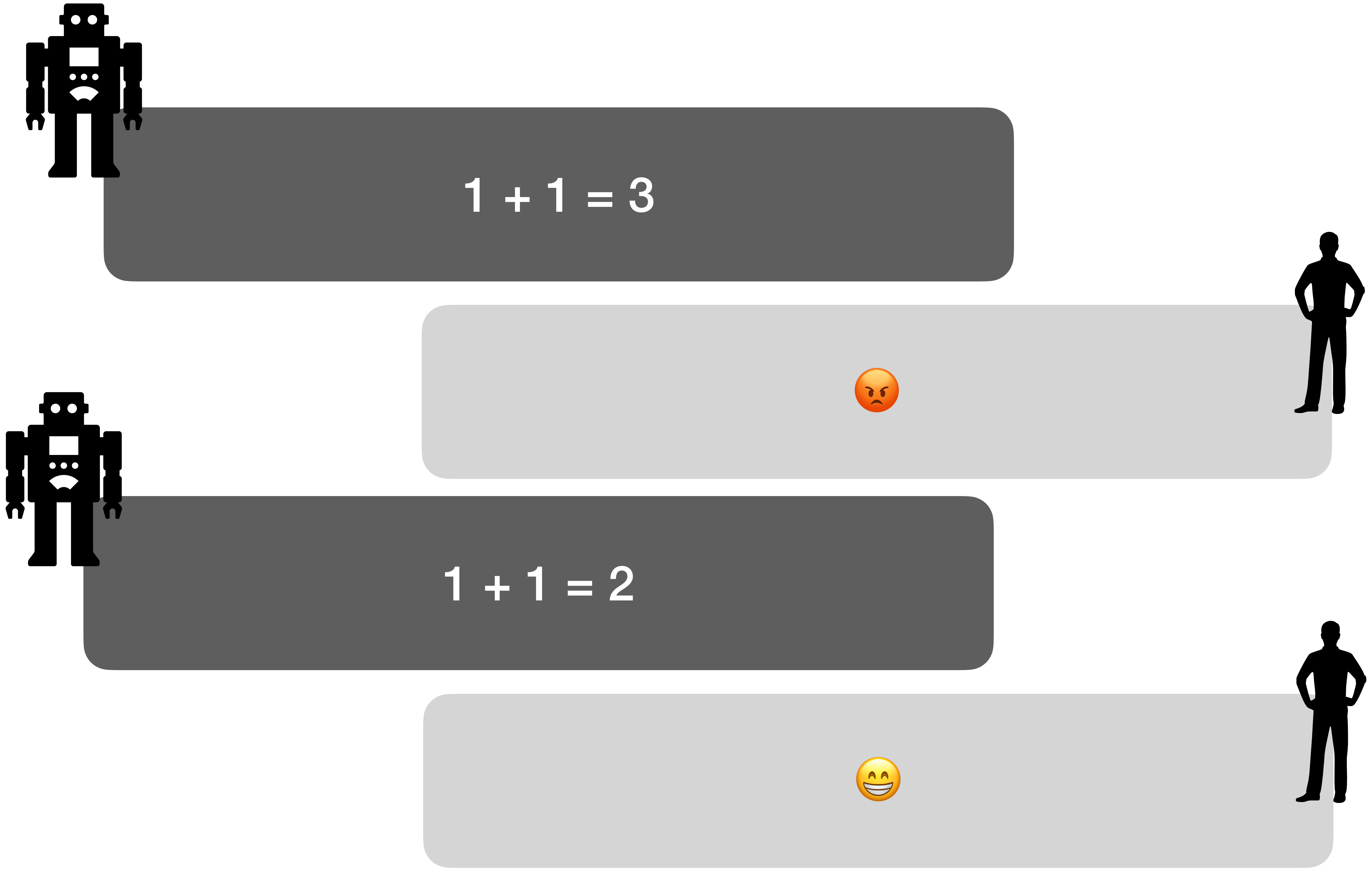
😡

1 + 1 = 2

😁

Problem of interaction with language model

- Human in the loop!
  - Grunt work



1 + 1 = 3

😡

1 + 1 = 2

😁

Require too much labor

Problem of interaction with language model

- Algorithmic way
  - Require lots of reimplementation!

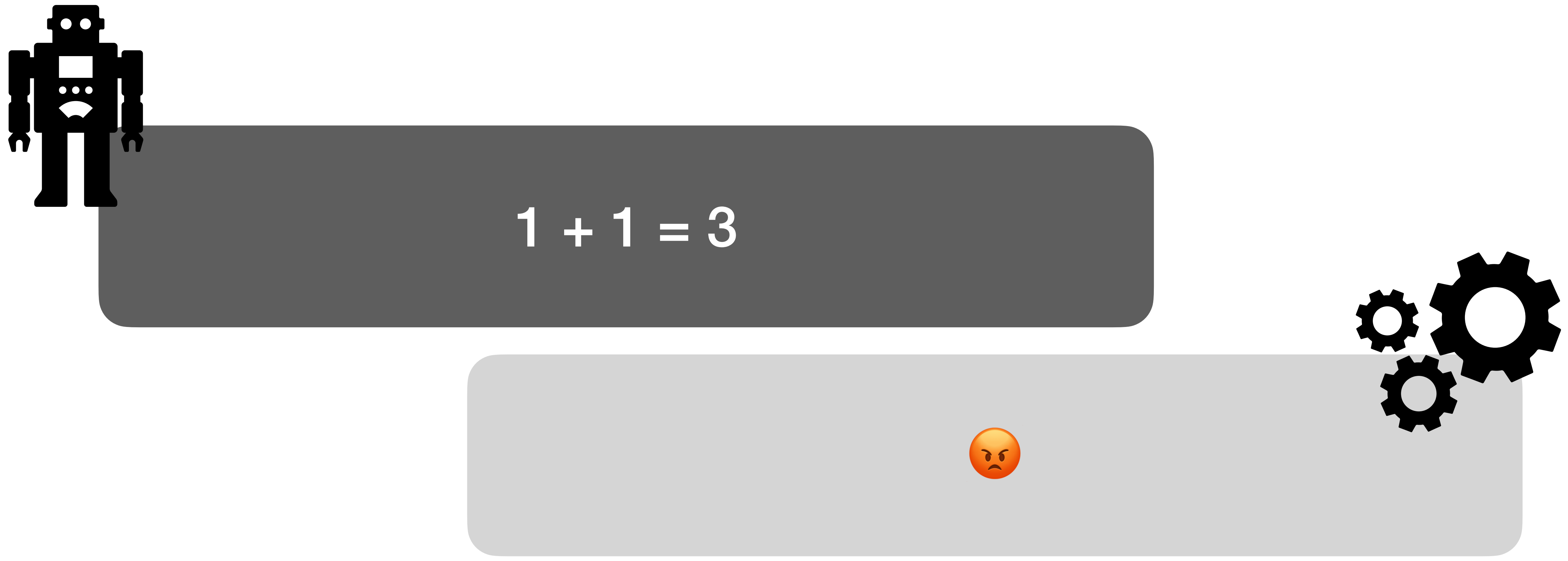
Problem of interaction with language model

- Algorithmic way
  - Require lots of reimplementation!

1 + 1 = 3

Problem of interaction with language model

- Algorithmic way
  - Require lots of reimplementation!

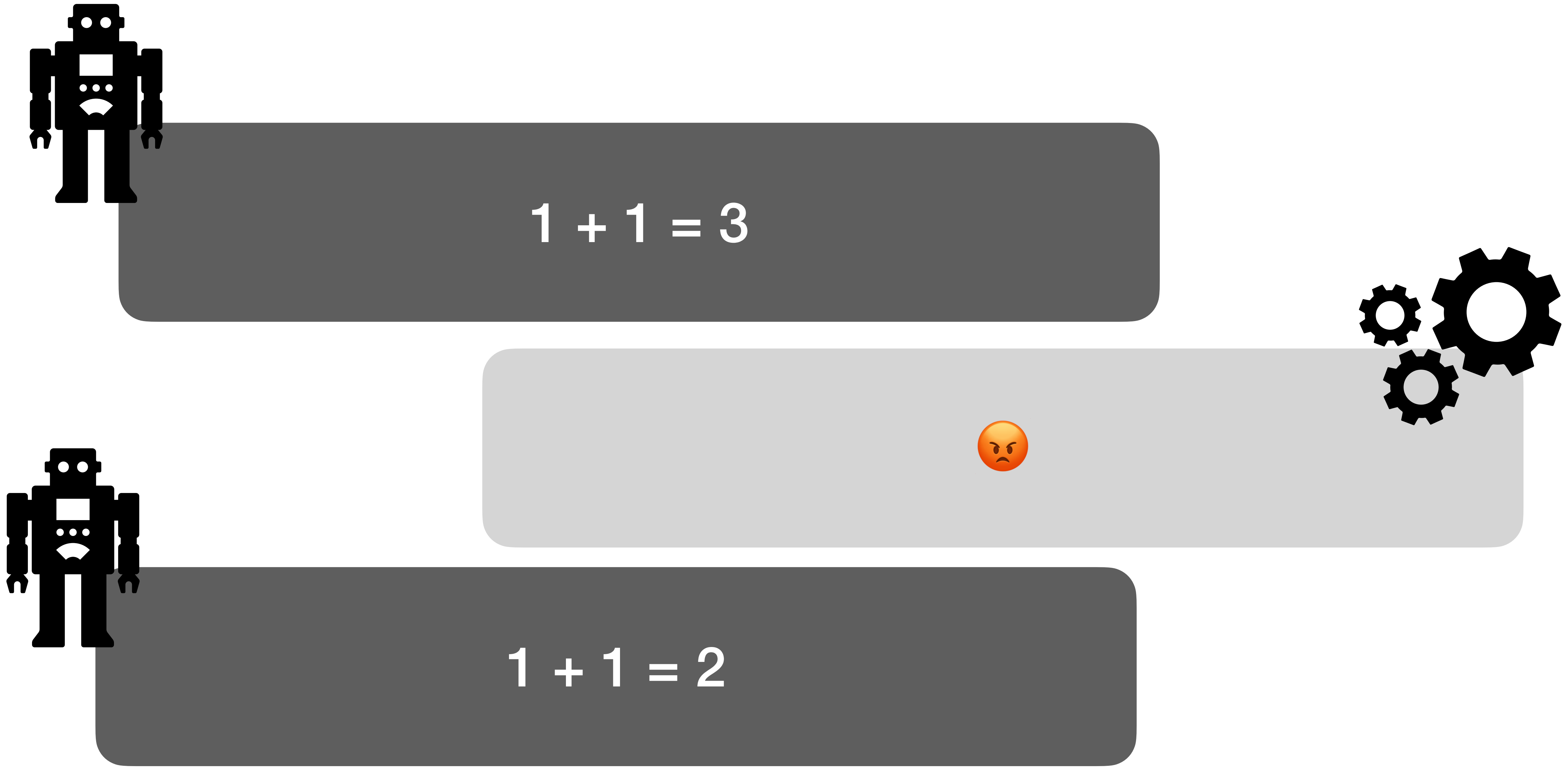


1 + 1 = 3

😡

Problem of interaction with language model

- Algorithmic way
  - Require lots of reimplementation!



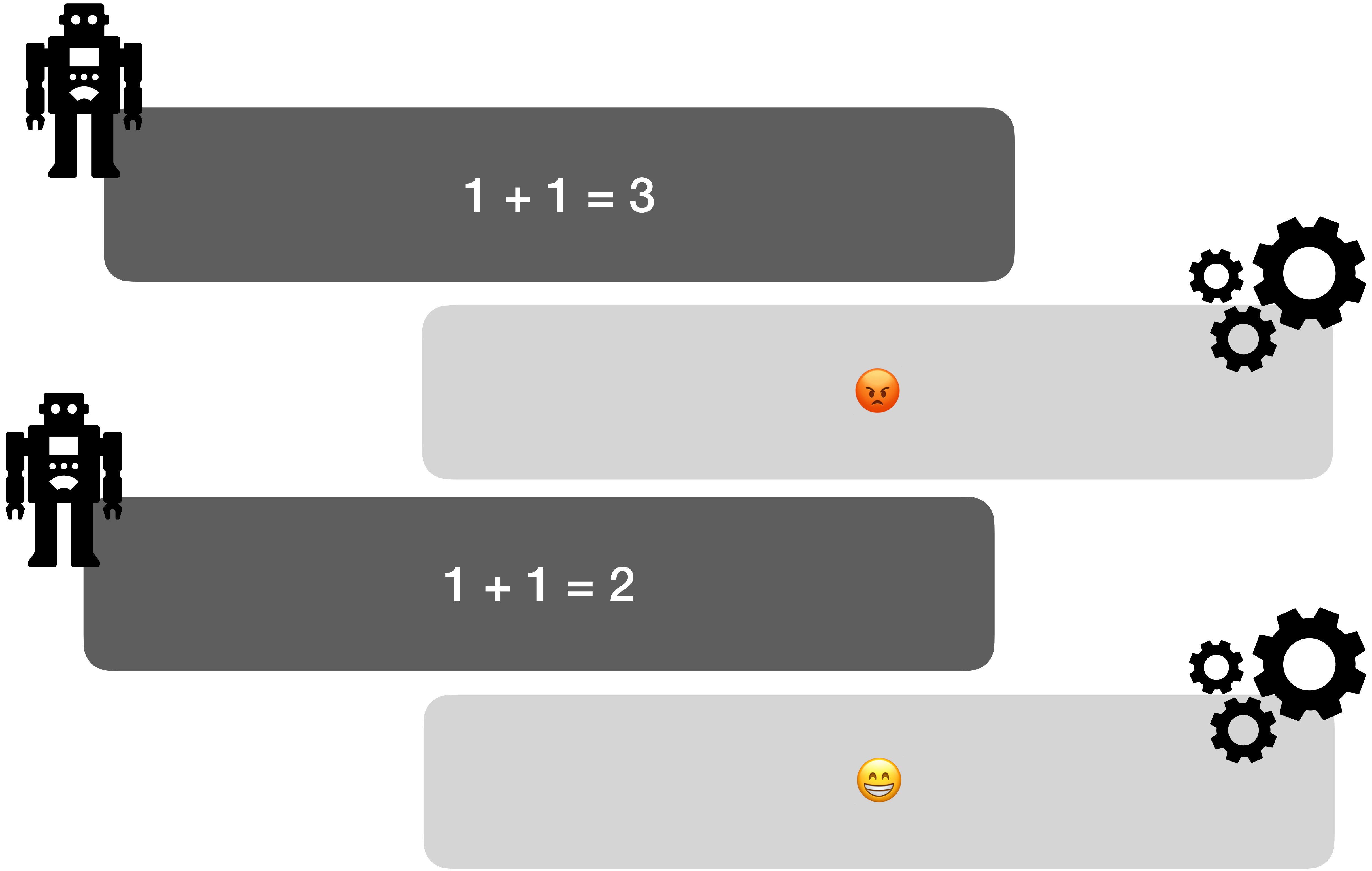
1 + 1 = 3

😡

1 + 1 = 2

Problem of interaction with language model

- Algorithmic way
  - Require lots of reimplementation!



1 + 1 = 3

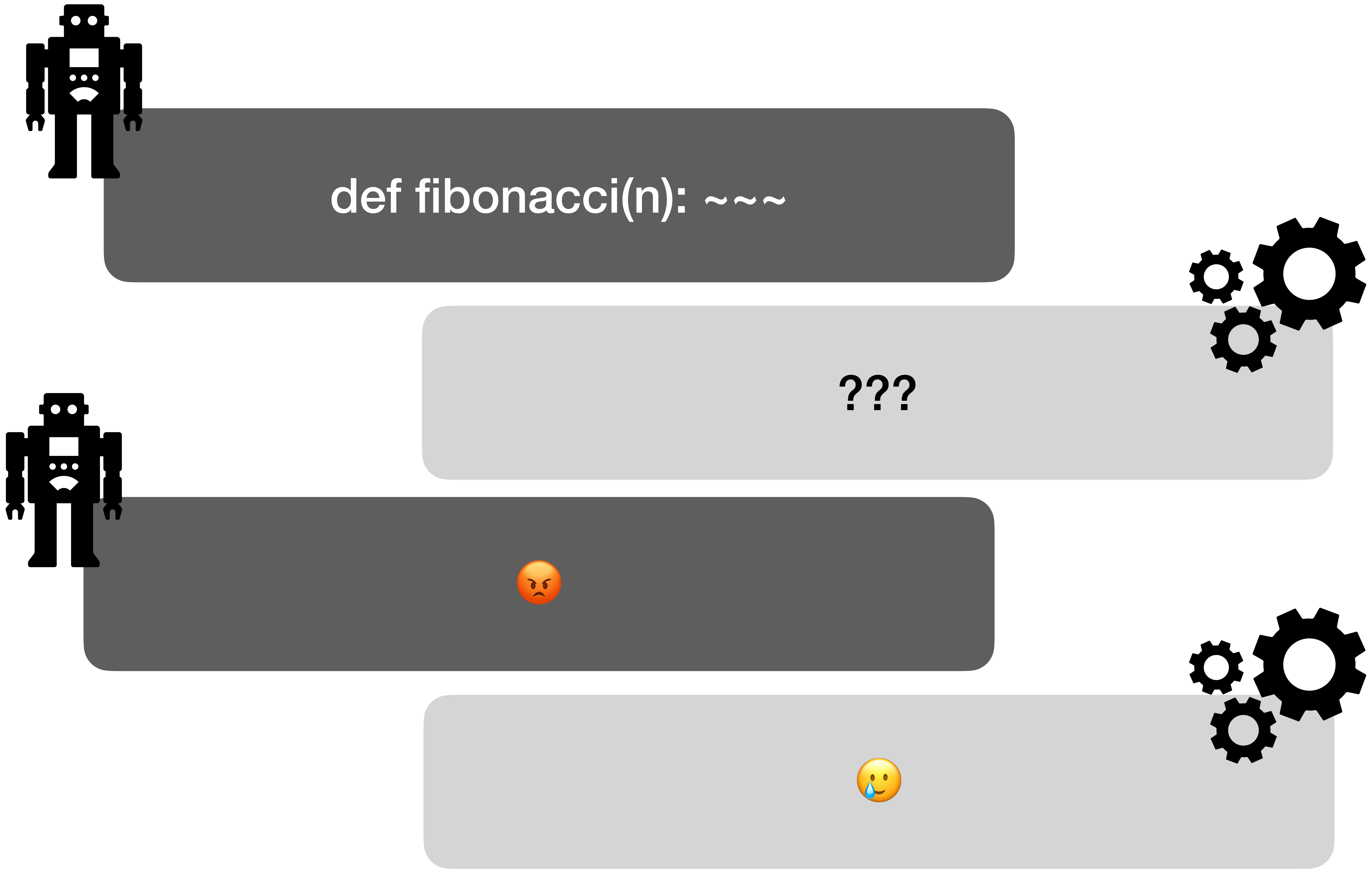
😡

1 + 1 = 2

😁

Problem of interaction with language model

- Algorithmic way
  - Require lots of reimplementation!



def fibonacci(n): ~~~

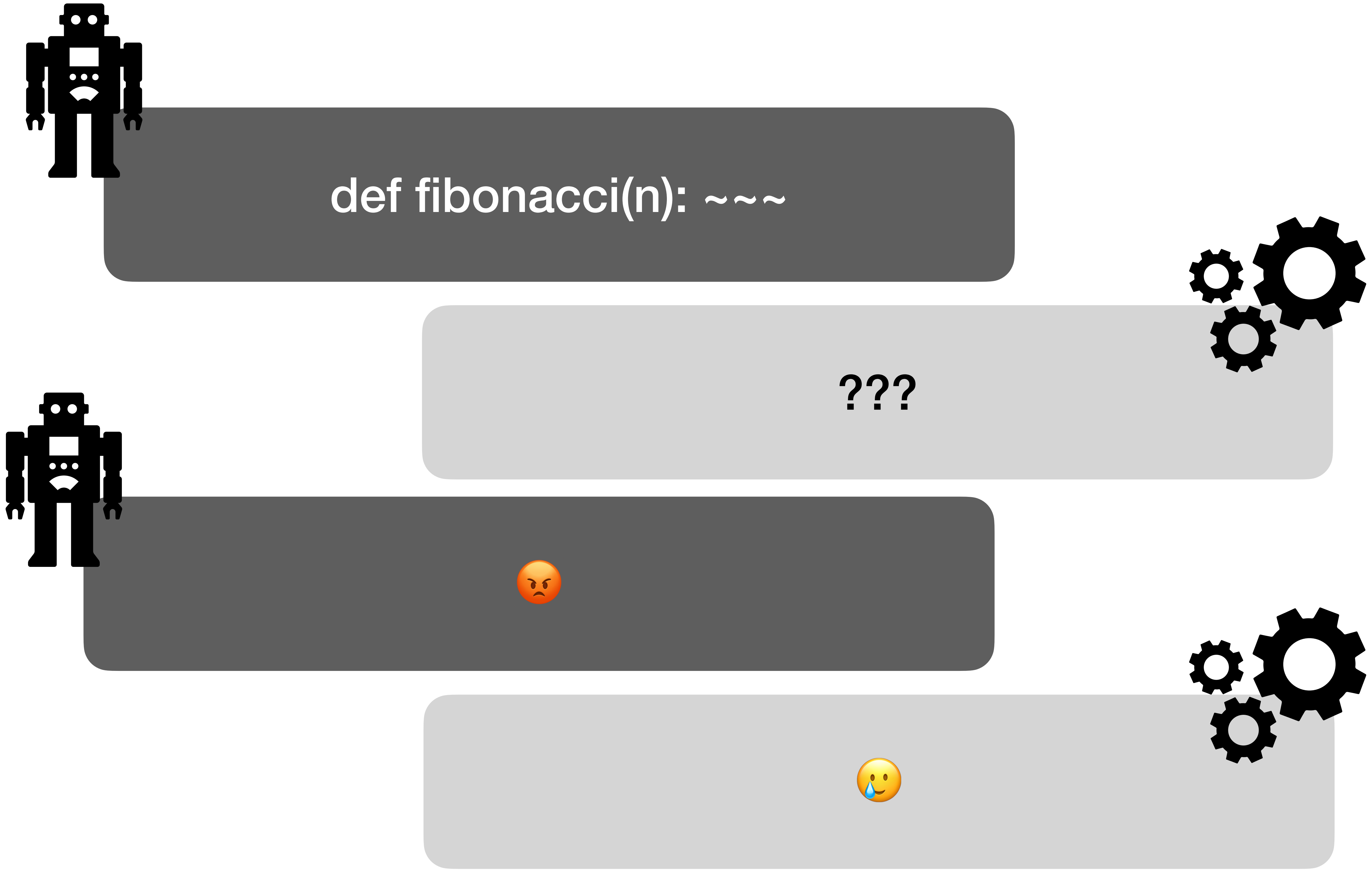
???

😡

🥲

Problem of interaction with language model

- Algorithmic way
  - Require lots of reimplementation!



def fibonacci(n): ~~~

???

😡

🥲

Constraint checker must be reimplemented from the scratch

What is the cause of the problem?

- No abstraction and interface!

What is the cause of the problem?

- No abstraction and interface!
- Computer Science has very well-designed abstraction layers

What is the cause of the problem?

- No abstraction and interface!
- Computer Science has very well-designed abstraction layers
    - Hardware (Architecture) / Software

What is the cause of the problem?

- No abstraction and interface!
- Computer Science has very well-designed abstraction layers
  - Hardware (Architecture) / Software
  - Assembly / Programming Language

What is the cause of the problem?

- No abstraction and interface!
- Computer Science has very well-designed abstraction layers
  - Hardware (Architecture) / Software
  - Assembly / Programming Language
  - ...

What is the cause of the problem?

- No abstraction and interface!
- Computer Science has very well-designed abstraction layers
  - Hardware (Architecture) / Software
  - Assembly / Programming Language
  - ...
- Language Models

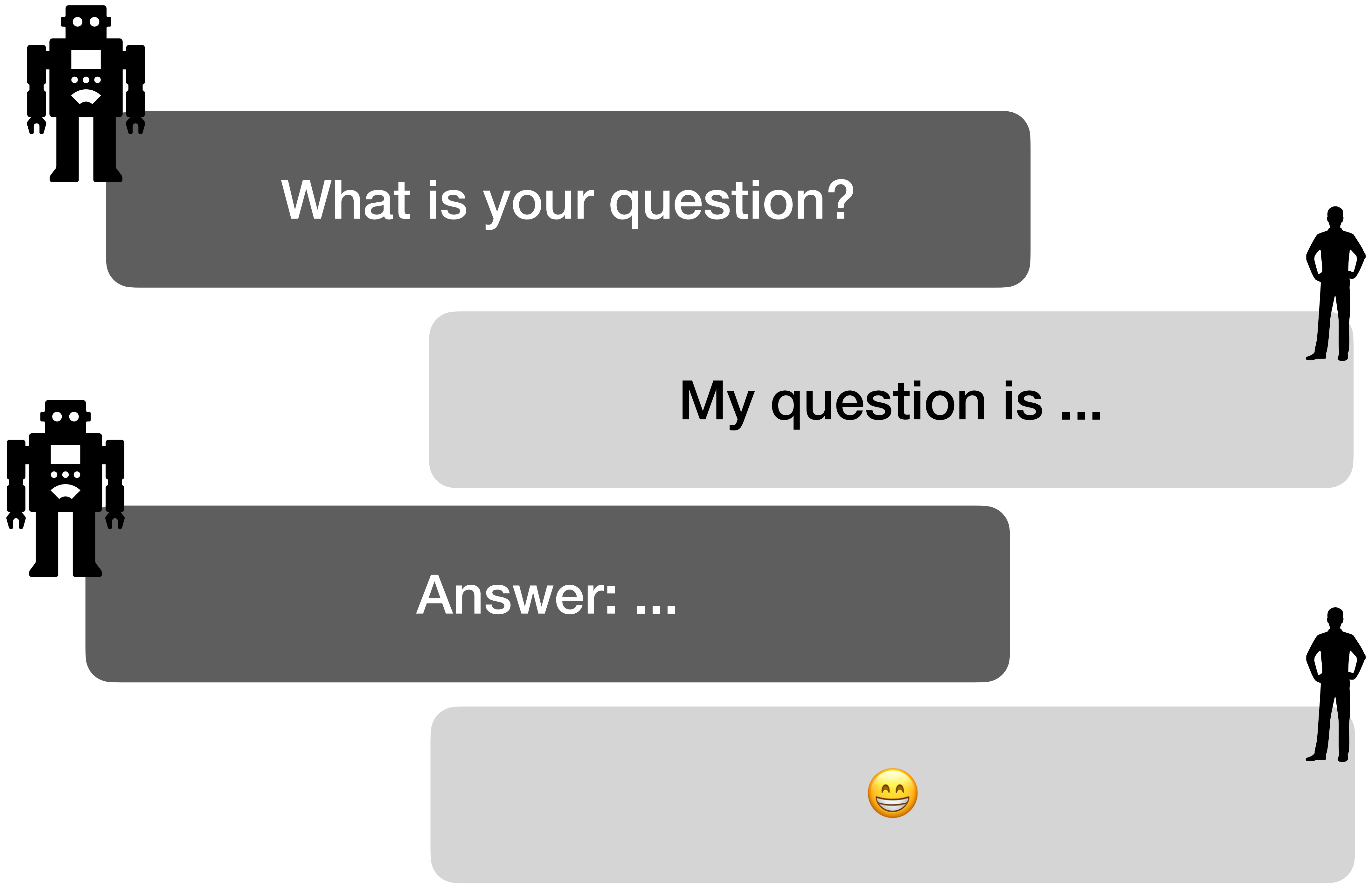
What is the cause of the problem?

- No abstraction and interface!
- Computer Science has very well-designed abstraction layers
  - Hardware (Architecture) / Software
  - Assembly / Programming Language
  - ...
- Language Models
  - No **interface** between language models and applications

What is the cause of the problem?

- No abstraction and interface!
- Computer Science has very well-designed abstraction layers
  - Hardware (Architecture) / Software
  - Assembly / Programming Language
  - ...
- Language Models
  - No **interface** between language models and applications
  - **Then... Let's make it!**

LMQL



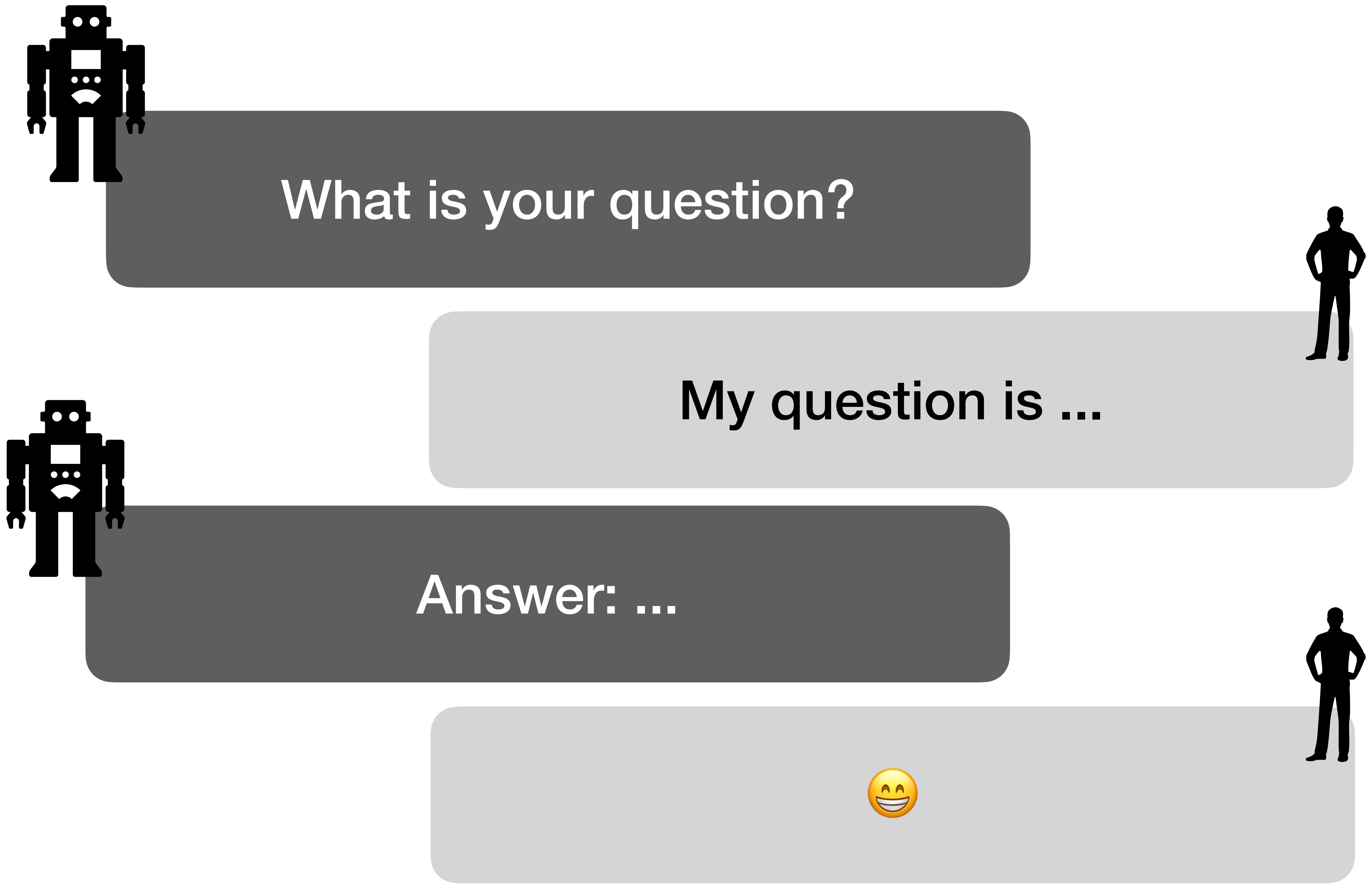
What is your question?

My question is ...

Answer: ...

😁

LMQL



What is your question?

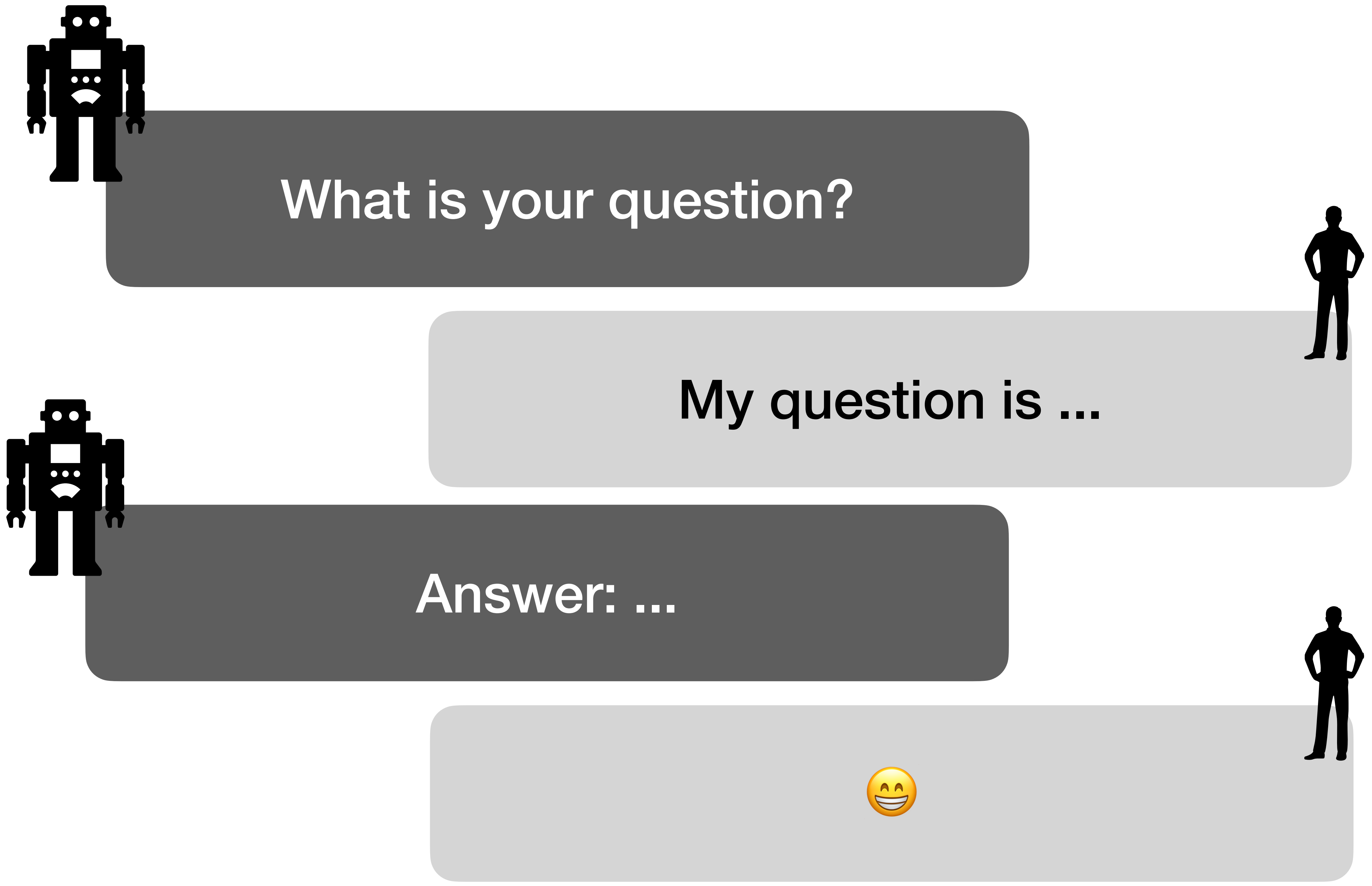
My question is ...

Answer: ...

😁

LMQL Script

LMQL

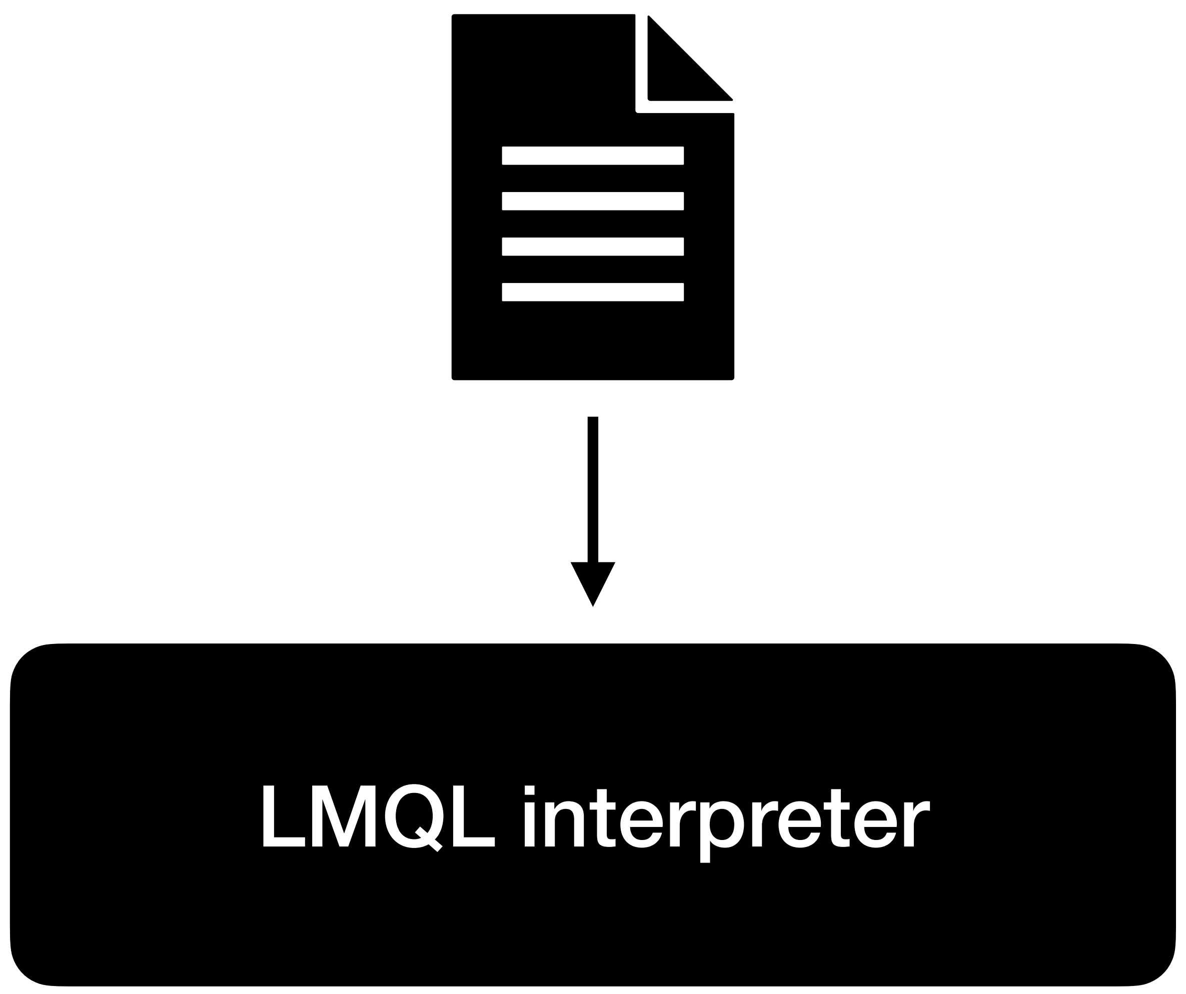

What is your question?

My question is ...

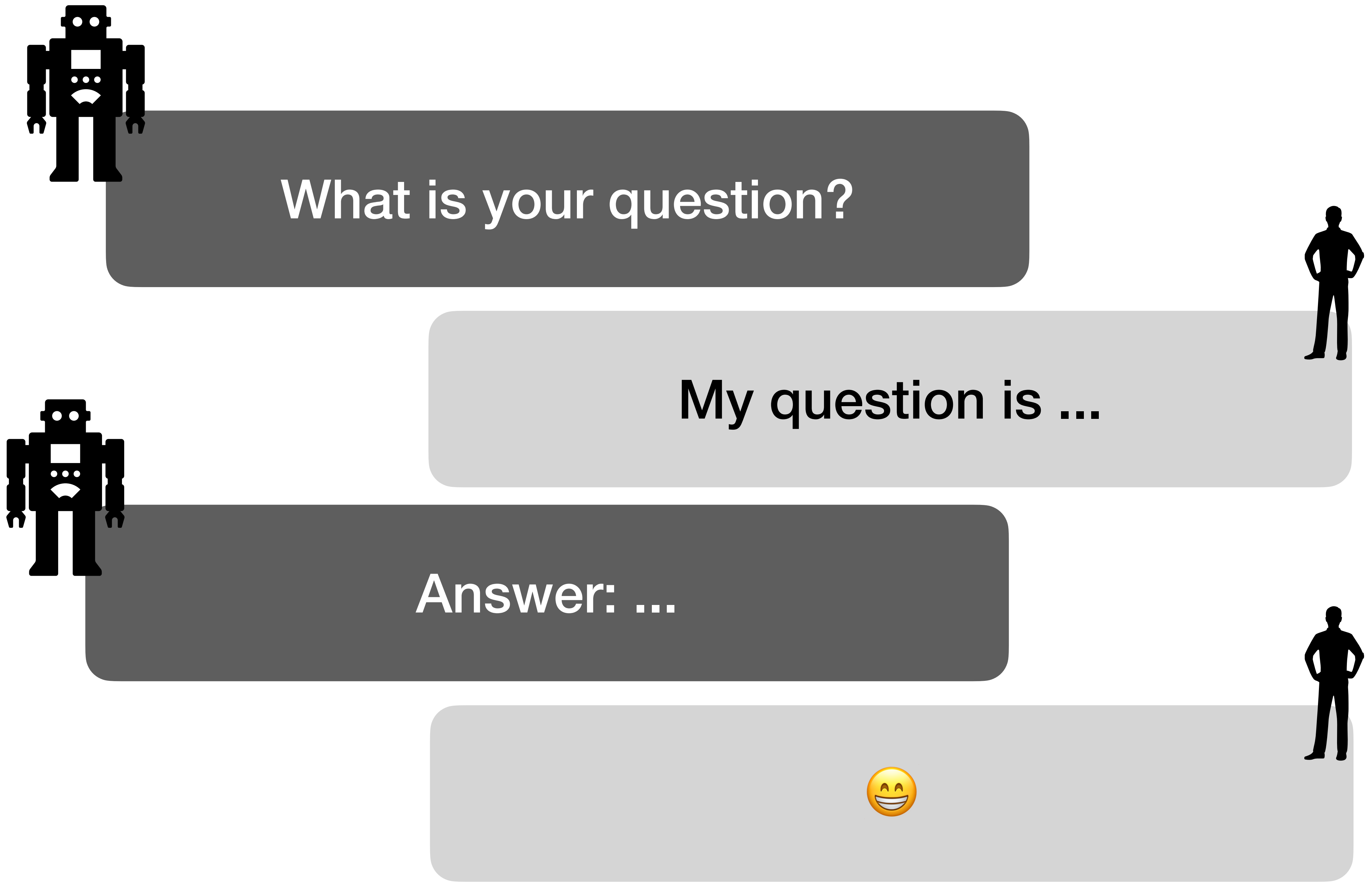
Answer: ...

😁

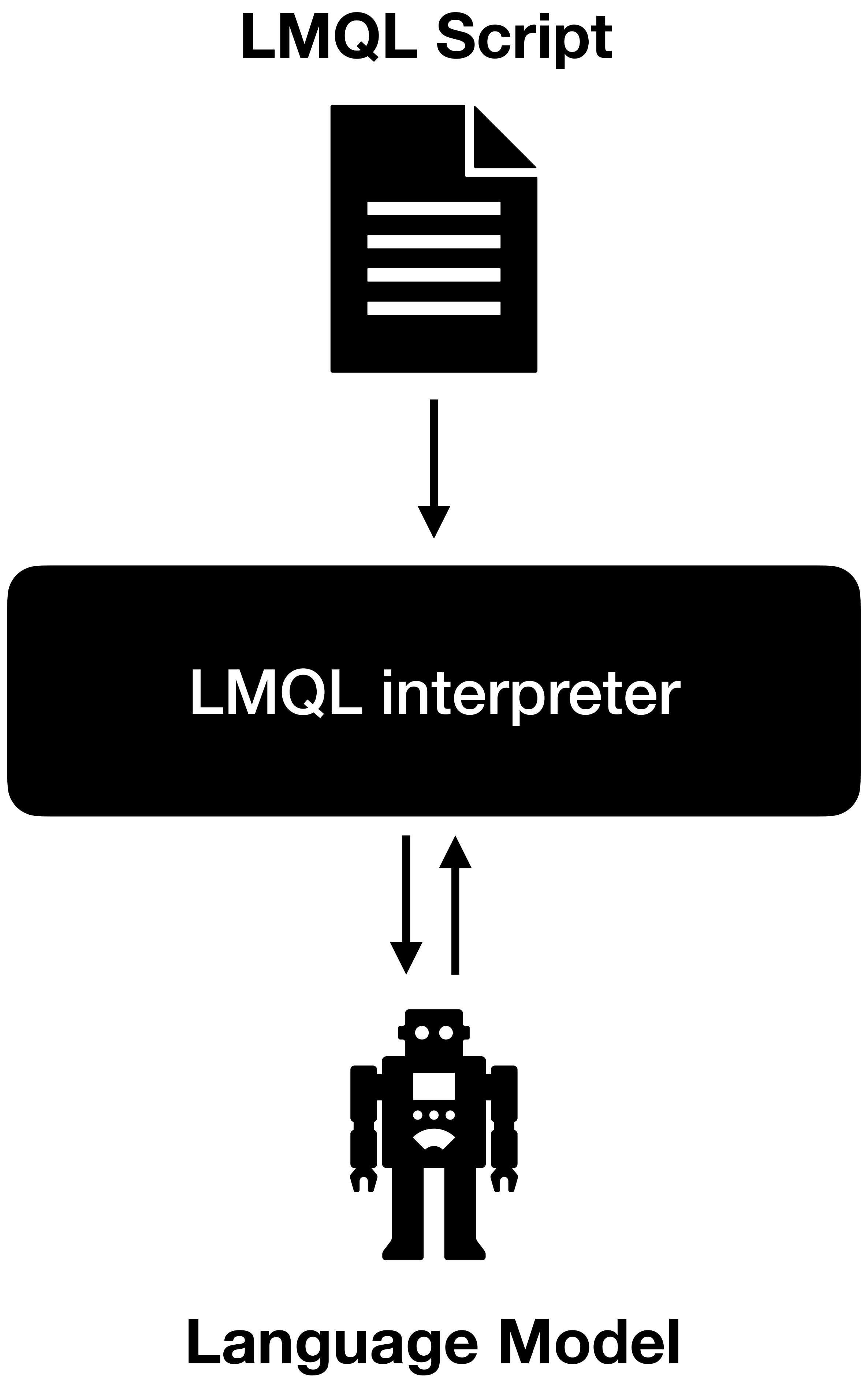
LMQL Script


LMQL interpreter

LMQL



What is your question?
My question is ...
Answer: ...
😁



LMQL Script
LMQL interpreter
Language Model

Interface for language model

- **Langauge model programming (LMP)**

Interface for language model

- **Langauge model programming (LMP)**
    - Abstract language model prompting and interaction

Interface for language model

- **Langauge model programming (LMP)**
  - Abstract language model prompting and interaction
  - Combining text prompting and scripting (some algorithm)

Interface for language model

- **Langauge model programming (LMP)**
  - Abstract language model prompting and interaction
  - Combining text prompting and scripting (some algorithm)
- **Language Model Query Language (LMQL)**

Interface for language model

- **Langauge model programming (LMP)**
  - Abstract language model prompting and interaction
  - Combining text prompting and scripting (some algorithm)
- **Language Model Query Language (LMQL)**
  - **Abstraction of language model**

Interface for language model

- **Langauge model programming (LMP)**
  - Abstract language model prompting and interaction
  - Combining text prompting and scripting (some algorithm)
- **Language Model Query Language (LMQL)**
  - **Abstraction of language model**
  - We can write down **scripts for interacting** with a language model

Interface for language model

- **Langauge model programming (LMP)**
  - Abstract language model prompting and interaction
  - Combining text prompting and scripting (some algorithm)
- **Language Model Query Language (LMQL)**
  - **Abstraction of language model**
  - We can write down **scripts for interacting** with a language model
  - We can give **some constraints** at the high level (word, number, ...)

How to use LMQL

- LMQL is Python + SQL!
- Outline of LMQL Program

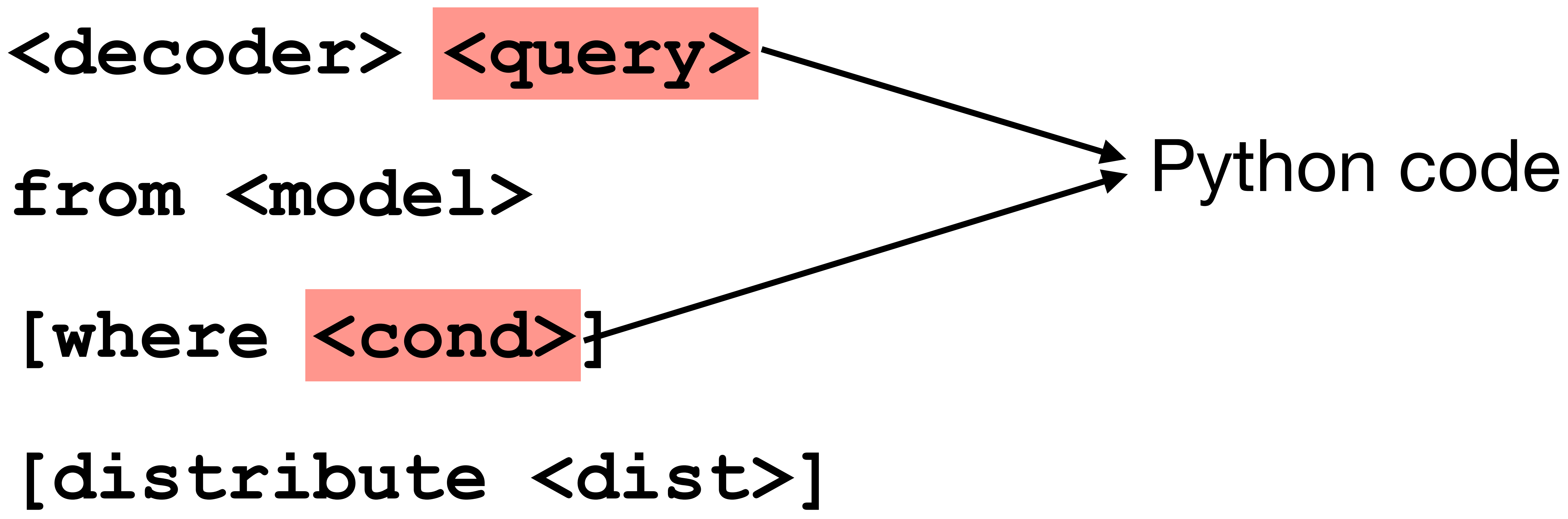
How to use LMQL

- LMQL is Python + SQL!
- Outline of LMQL Program

```
<decoder> <query>

from <model>

[where <cond>]

[distribute <dist>]
```

How to use LMQL

- LMQL is Python + SQL!
- Outline of LMQL Program



```
<decoder> <query>
from <model>
[where <cond>]
[distribute <dist>]
```

Python code

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

We are going to use greedy decoding

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

→ Use language model gpt2-large

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

INT(c) → Variable c must be integer

Example

LMQL Program

```
argmax
  a = 2
  b = 3
  "What is {a} + {b}? [c]"
  "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

[redacted]

Prompt

""

Example

LMQL Program

Environment

[redacted]

Prompt

""

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

a = 2 → Add variable **a** to environment

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

`a = 2`

Prompt

""

Example

LMQL Program

Environment

```
a = 2
```

Prompt

""

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

b = 3 → Add variable **b** to environment

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

```
a = 2
b = 3
```

Prompt

""

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

```
a = 2
b = 3
```

Prompt

""

`{?}`: Get `?` from environment

`[c]`: Get response from LM.
Then, save it in environment

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

```
a = 2
b = 3
```

Prompt

"What is 2 + 3? "

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

```
a = 2
b = 3
```

Prompt

"What is 2 + 3? "

Prompt: "What is 2 + 3? "

gpt2-large: "5"

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

```
a = 2
b = 3
c = 5
```

Prompt

```
"What is 2 + 3? 5"
```

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

{?}: Get ? from environment

Environment

```
a = 2
b = 3
c = 5
```

Prompt

```
"What is 2 + 3? 5"
```

Example

LMQL Program

```
argmax
   a = 2
   b = 3
   "What is {a} + {b}? [c]"
   "The answer is {c}"
from "gpt2-large"
where INT(c)
```

Environment

```
a = 2
b = 3
c = 5
```

Prompt

```
"What is 2 + 3? 5
The answer is 5"
```

Key features

- Interaction with **control flow and environment**

Key features

- Interaction with **control flow and environment**
    - We can use **branch, loop, and function calls** like programming language

Key features

- Interaction with **control flow and environment**
    - We can use **branch, loop, and function calls** like programming language
    - We can use **variable**

Key features

- Interaction with **control flow and environment**
  - We can use **branch, loop, and function calls** like programming language
  - We can use **variable**
- Constraint decoding

Key features

- Interaction with **control flow and environment**
  - We can use **branch, loop, and function calls** like programming language
  - We can use **variable**
- Constraint decoding
  - We can give some constraints through **`where`** statement

Key features

- Interaction with **control flow and environment**
  - We can use **branch, loop, and function calls** like programming language
  - We can use **variable**
- Constraint decoding
  - We can give some constraints through **`where`** statement
  - You can **specify all constraints** that can be expressed with Python Expr

Key features

- Interaction with **control flow and environment**
  - We can use **branch, loop, and function calls** like programming language
  - We can use **variable**
- Constraint decoding
  - We can give some constraints through **`where`** statement
  - You can **specify all constraints** that can be expressed with Python Expr

What is constraint decoding?

- Generate output to satisfy the given constraints

What is constraint decoding?

- Generate output to satisfy the given constraints
- Constraints can be

What is constraint decoding?

- Generate output to satisfy the given constraints
- Constraints can be
  - Format (Syntax)

What is constraint decoding?

- Generate output to satisfy the given constraints
- Constraints can be
  - Format (Syntax)
  - Type

What is constraint decoding?

- Generate output to satisfy the given constraints
- Constraints can be
  - Format (Syntax)
  - Type
  - Length of output

What is constraint decoding?

- Generate output to satisfy the given constraints
- Constraints can be
  - Format (Syntax)
  - Type
  - Length of output
  - Termination Condition

What is constraint decoding?

- Generate output to satisfy the given constraints
- Constraints can be
  - Format (Syntax)
  - Type
  - Length of output
  - Termination Condition
  - …

Naïve approach

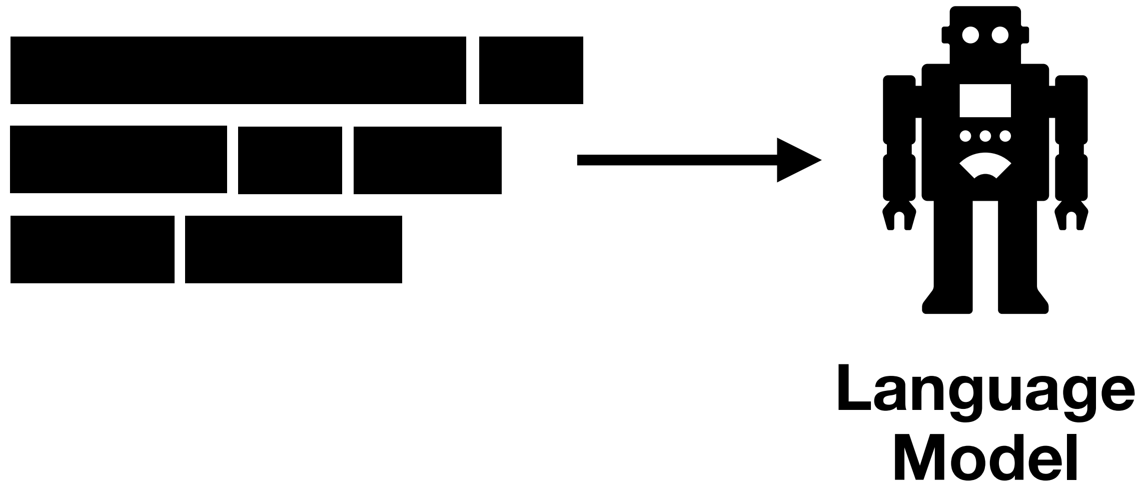
- Check **after** decode

Naïve approach

- Check **after** decode

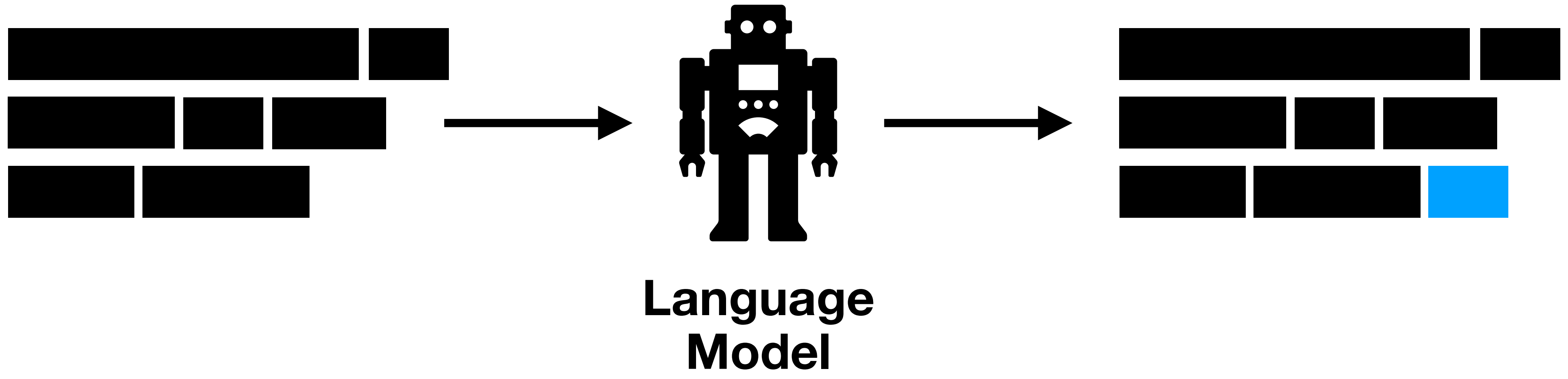
Naïve approach

- Check **after** decode



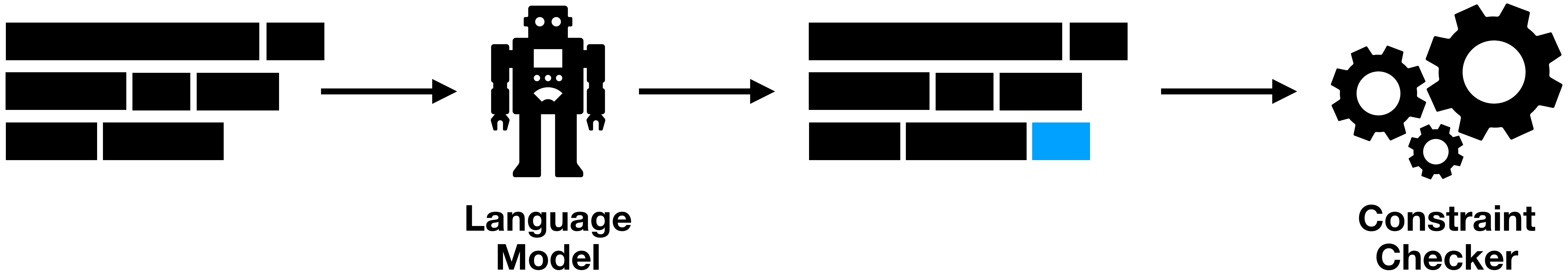
Naïve approach

- Check **after** decode



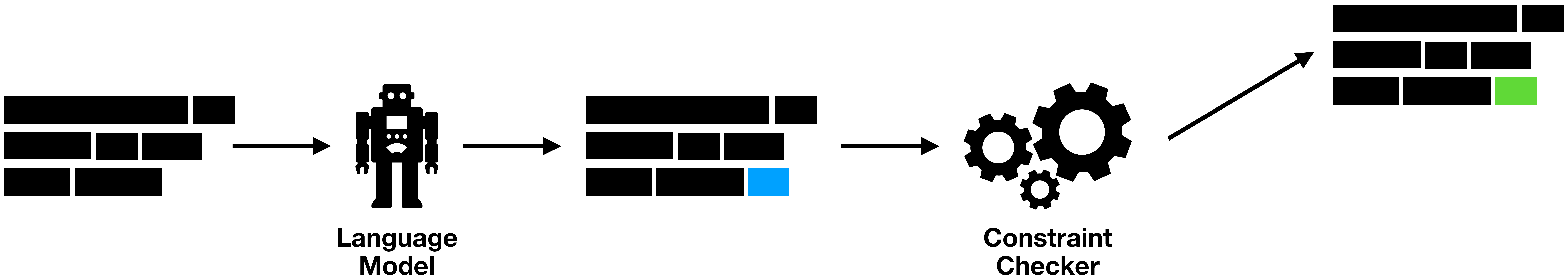
Naïve approach

- Check **after** decode



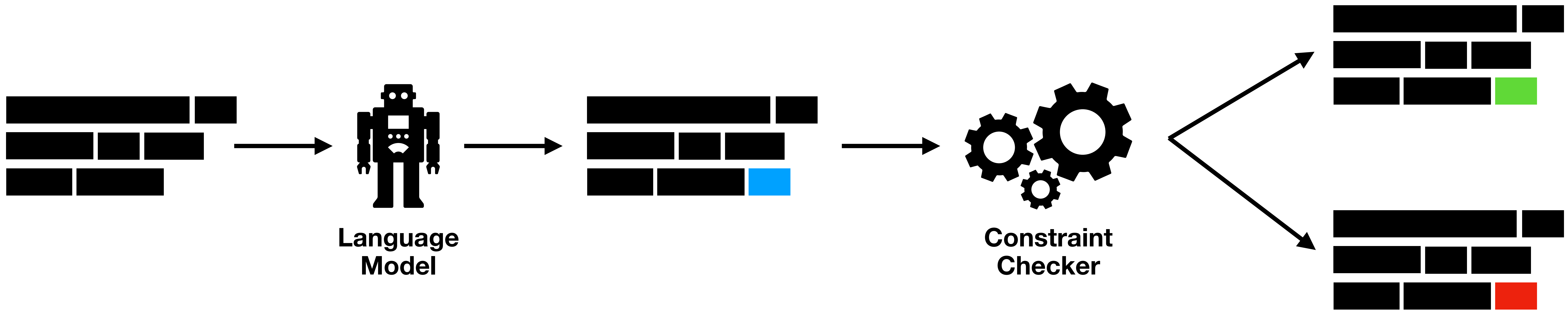
Naïve approach

- Check **after** decode



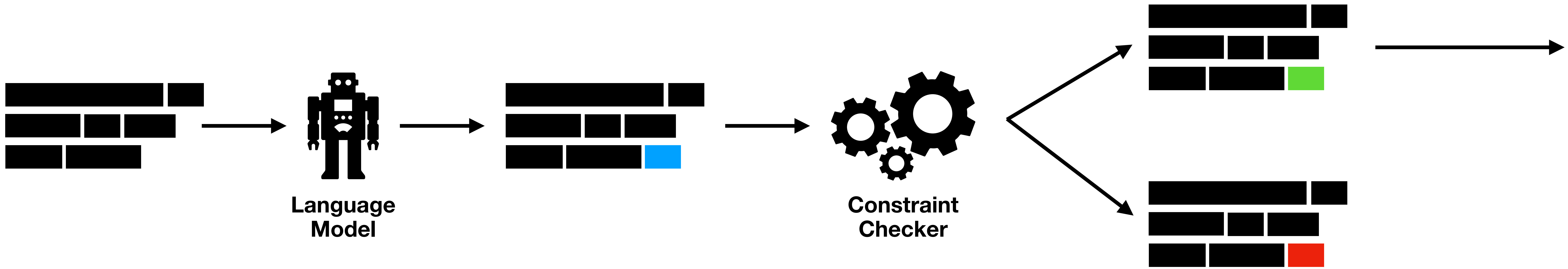
Naïve approach

- Check **after** decode



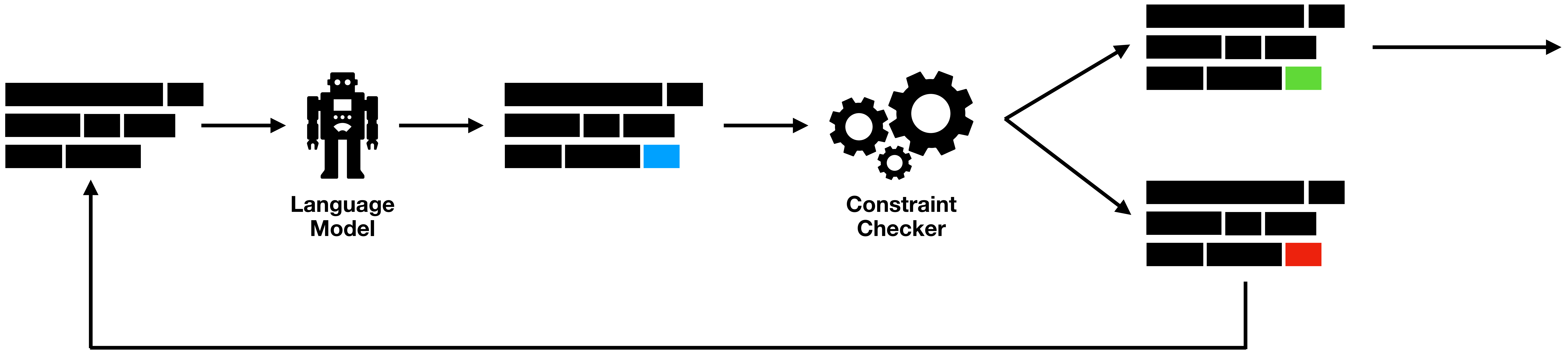
Naïve approach

- Check **after** decode



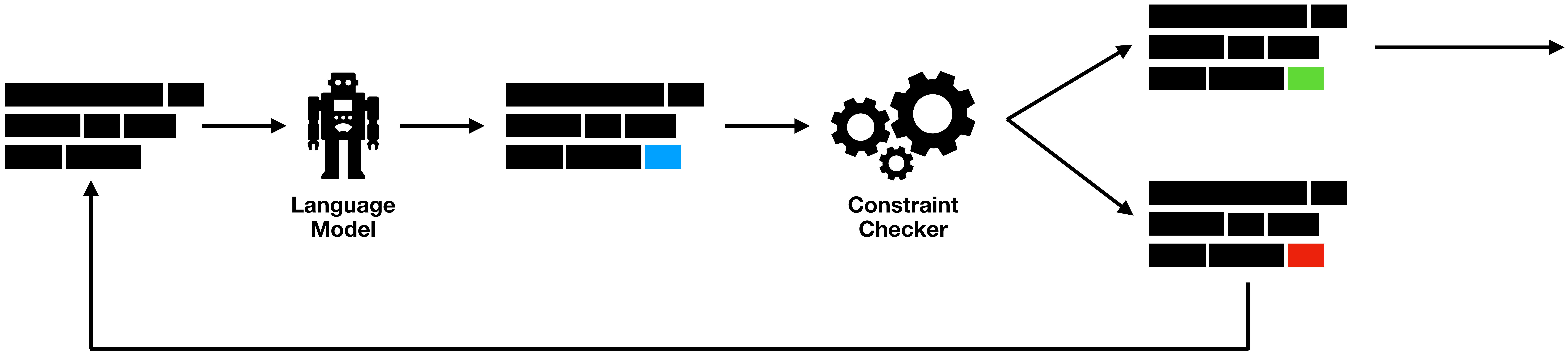
Naïve approach

- Check **after** decode



Naïve approach

- Check **after** decode



- Problem: **Decoding time >>> Checking time**

Efficient approach

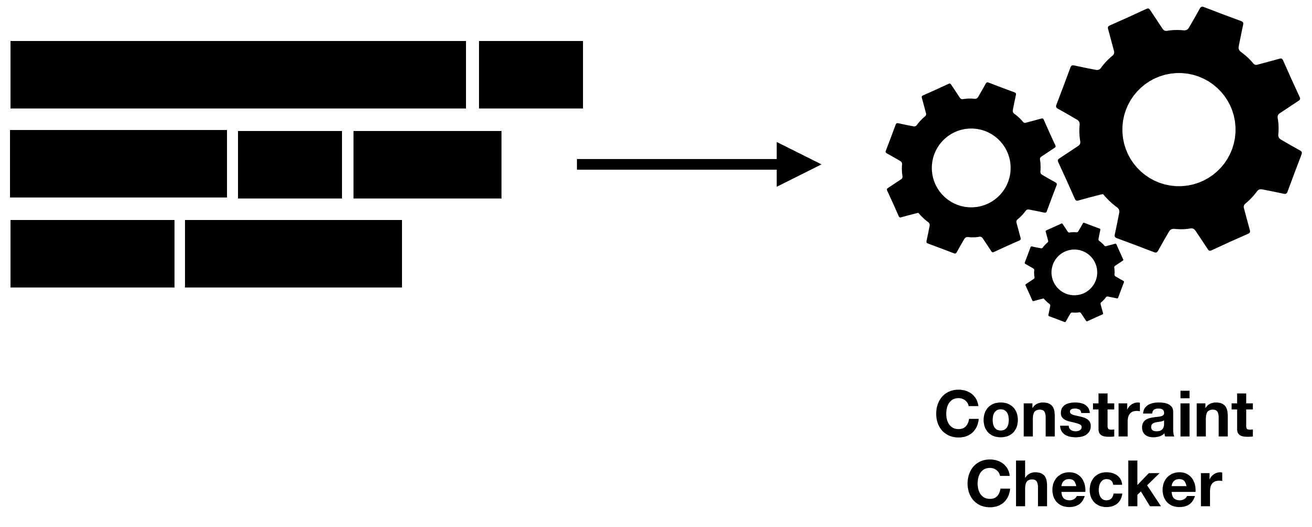
- Check **before** decode

Efficient approach

- Check **before** decode

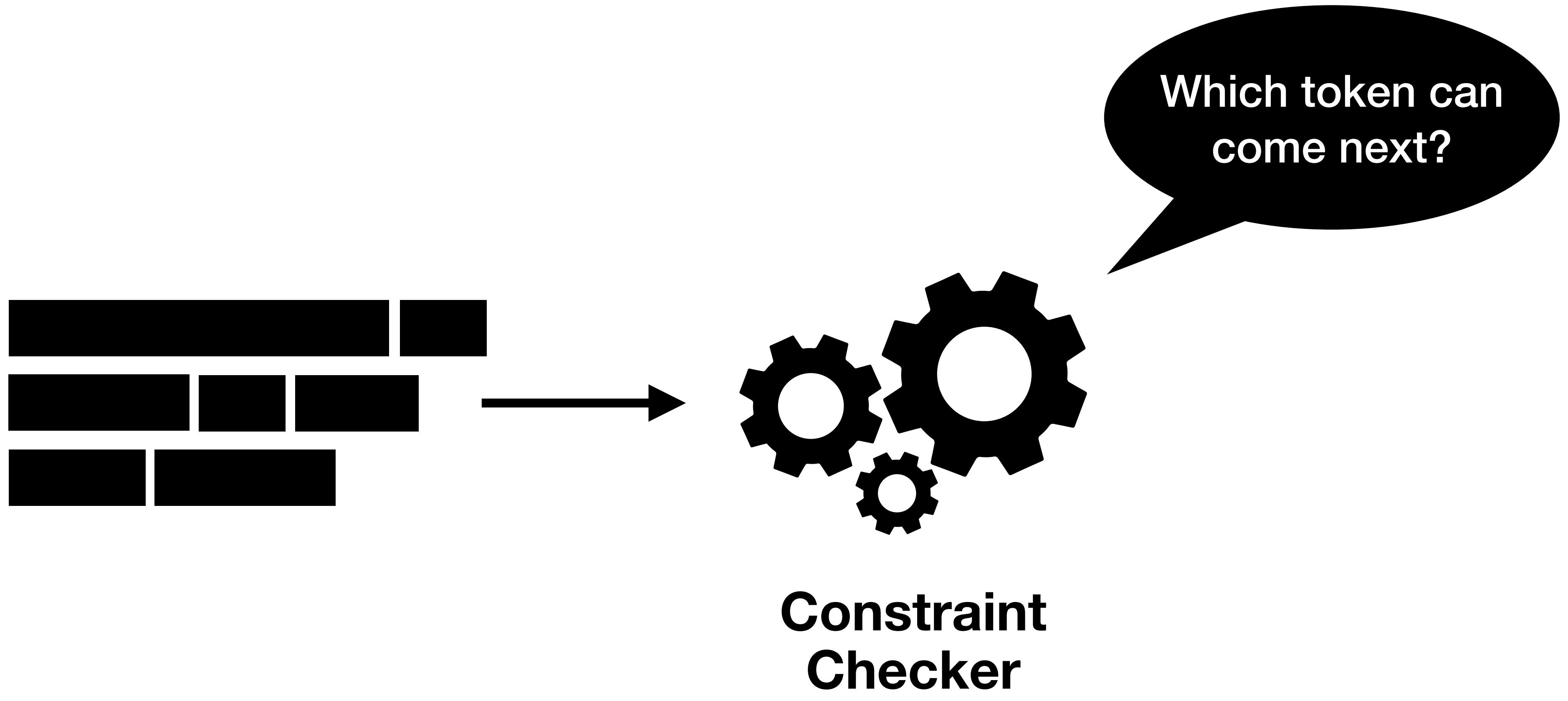
Efficient approach

- Check **before** decode



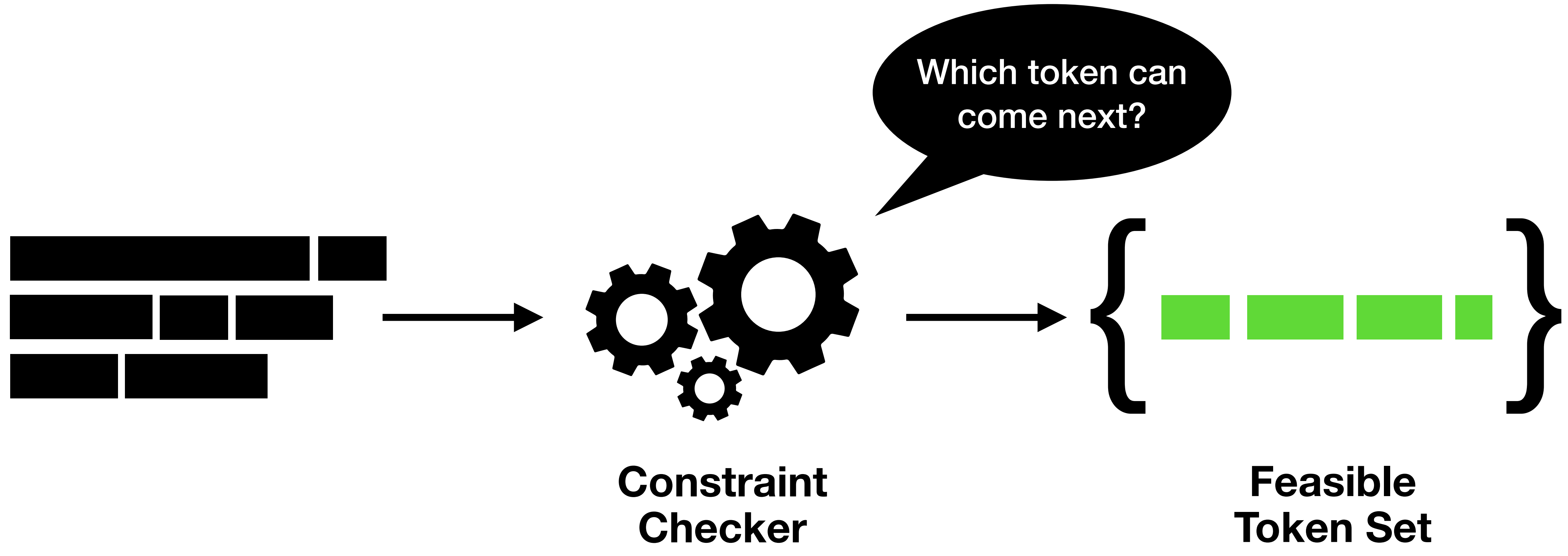
Efficient approach

- Check **before** decode



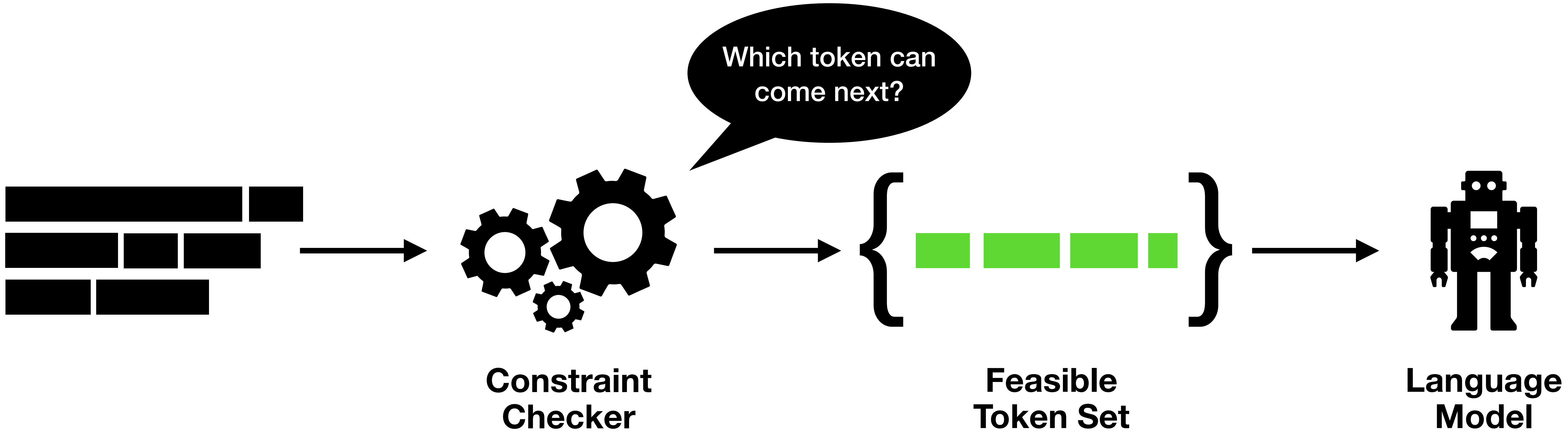
Efficient approach

- Check **before** decode



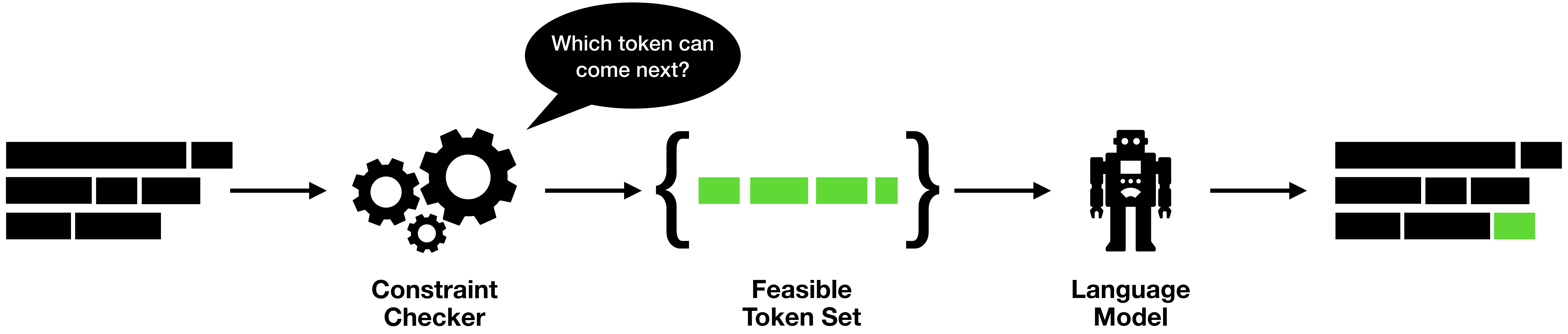
Efficient approach

- Check **before** decode



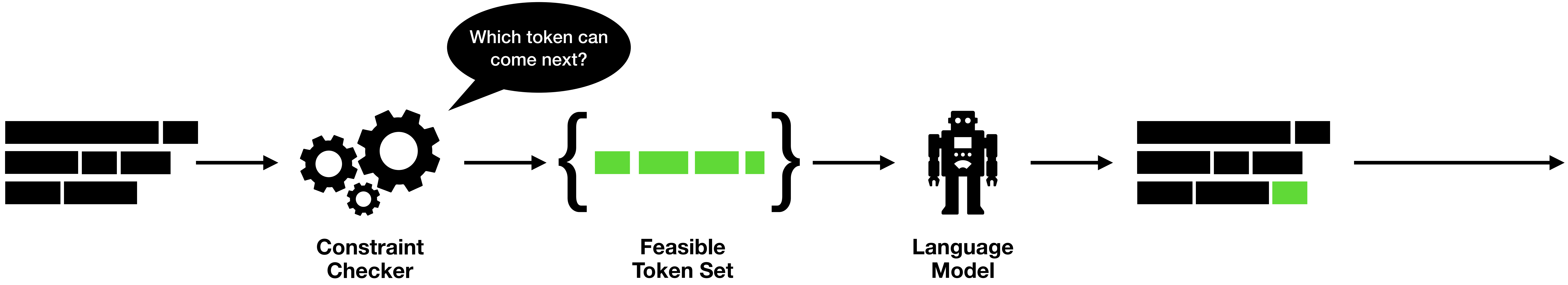
Efficient approach

- Check **before** decode



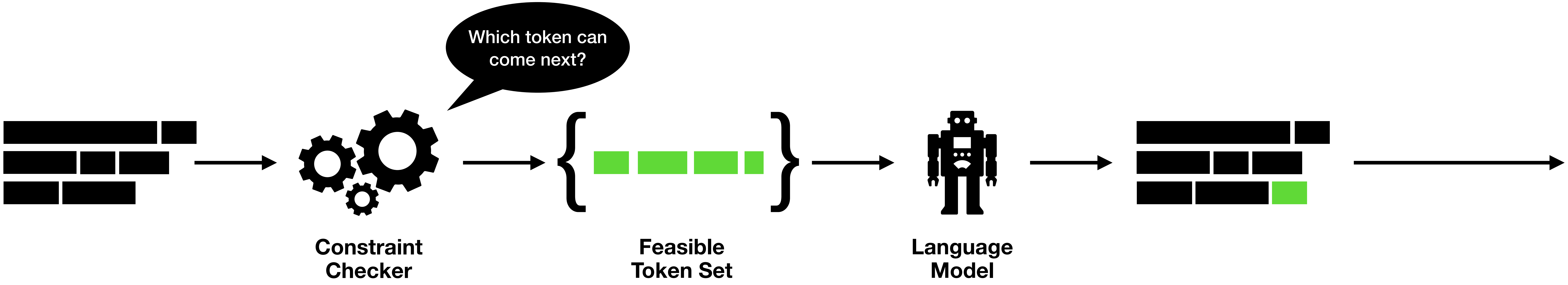
Efficient approach

- Check **before** decode



Efficient approach

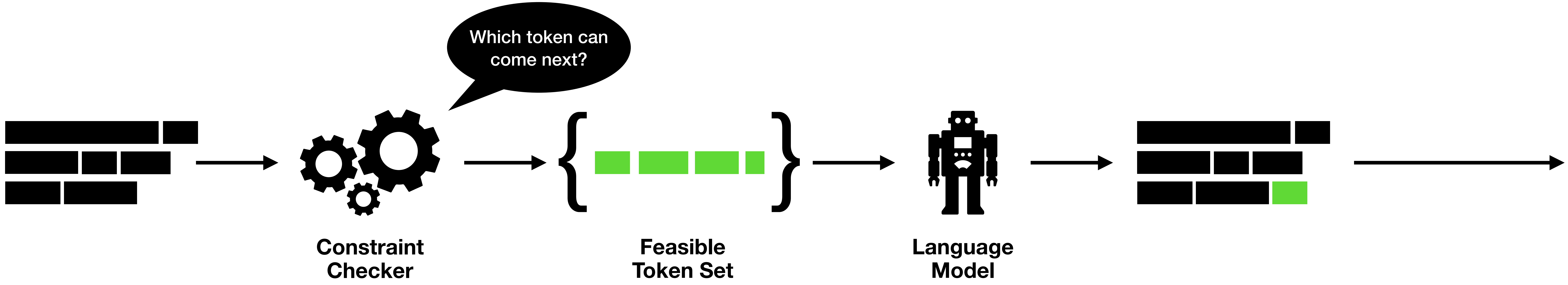
- Check **before** decode



- Feasible Token Set is always finite?

Efficient approach

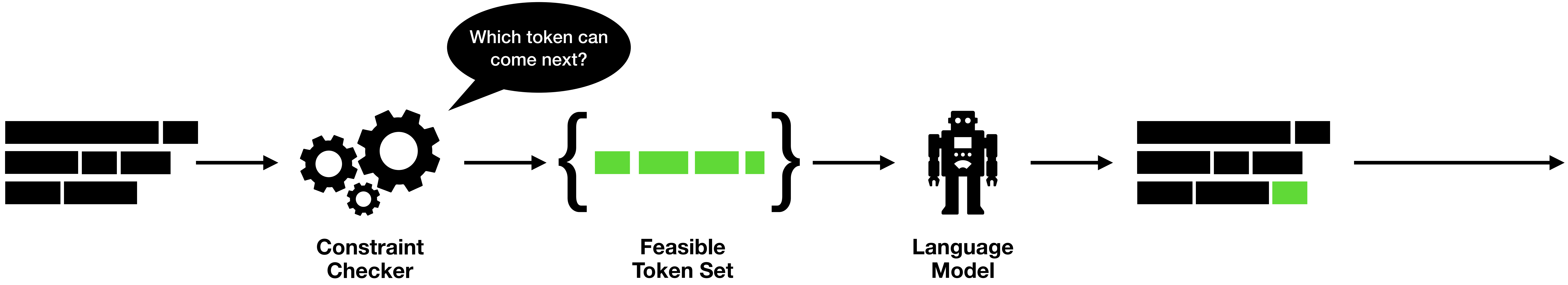
- Check **before** decode



- Feasible Token Set is always finite?
  - Yes

Efficient approach

- Check **before** decode



- Feasible Token Set is always finite?
  - Yes
  - Language Models have a finite token set

How to make a feasible token set?

- 2 Types of token

How to make a feasible token set?

- 2 Types of token
  - **Maybe** satisfying the constraint

How to make a feasible token set?

- 2 Types of token
  - **Maybe** satisfying the constraint
  - **Never** satisfying the constraint

How to make a feasible token set?

- 2 Types of token
  - **Maybe** satisfying the constraint
  - **Never** satisfying the constraint
- We cannot 100% guarantee that some tokens satisfy the given constraint

How to make a feasible token set?

- 2 Types of token
    - **Maybe** satisfying the constraint
    - **Never** satisfying the constraint
- We cannot 100% guarantee that some tokens satisfy the given constraint
- But we can guarantee that some tokens **never satisfy** the constraint

2 Types of token

2 Types of token

Constraint: `eval(Var) < 10 and match(Var, “[0-9+]*”)`

2 Types of token

Constraint: `eval(Var) < 10 and match(Var, “[0-9+]*”)`

Current Output: **`2+0+1+4+??`**

2 Types of token

Constraint: `eval(Var) < 10 and match(Var, “[0-9+]*”)`

Current Output: `2+0+1+4+??`

Maybe satisfying: `0, 1, 2`

2 Types of token

Constraint: `eval(Var) < 10 and match(Var, “[0-9+]*”)`

Current Output: `2+0+1+4+??`

Maybe satisfying: `0, 1, 2` → Depending on the trailing tokens

2 Types of token

Constraint: `eval(Var) < 10 and match(Var, “[0-9+]*”)`

Current Output: `2+0+1+4+??`

Maybe satisfying: `0, 1, 2` → Depending on the trailing tokens

Never satisfying: `3, 4, ...`

2 Types of token

Constraint: `eval(Var) < 10 and match(Var, “[0-9+]*”)`

Current Output: `2+0+1+4+??`

Maybe satisfying: `0, 1, 2` → Depending on the trailing tokens

Never satisfying: `3, 4, ...`

Focus on removing **“never satisfying”** tokens

Partial evalution

- To check the constraint, we should evaluate the current **unfinished output**

Partial evalution

- To check the constraint, we should evaluate the current **unfinished output**
- **2 Semantics** for partial evaluation

Partial evalution

- To check the constraint, we should evaluate the current **unfinished output**
- **2 Semantics** for partial evaluation
  - Final Semantics

Partial evalution

- To check the constraint, we should evaluate the current **unfinished output**
- **2 Semantics** for partial evaluation
    - Final Semantics
        - Denote current expression can be changed or not

Partial evalution

- To check the constraint, we should evaluate the current **unfinished output**
- **2 Semantics** for partial evaluation
    - Final Semantics
        - Denote current expression can be changed or not
        - 4 States: FIN, VAR, INC, DEC

Partial evalution

- To check the constraint, we should evaluate the current **unfinished output**
- **2 Semantics** for partial evaluation
  - Final Semantics
    - Denote current expression can be changed or not
    - 4 States: FIN, VAR, INC, DEC
  - Follow Map

Partial evalution

- To check the constraint, we should evaluate the current **unfinished output**
- **2 Semantics** for partial evaluation
  - Final Semantics
    - Denote current expression can be changed or not
    - 4 States: FIN, VAR, INC, DEC
  - Follow Map
    - Denote how the current state change according to the trailing token

Final Semantics

- Evaluate expression as 4 states

Final Semantics

- Evaluate expression as 4 states
  - FIN: The evaluation result of expression never changing

Final Semantics

- Evaluate expression as 4 states
  - FIN: The evaluation result of expression never changing
  - VAR: The evaluation result of expression changing

Final Semantics

- Evaluate expression as 4 states
  - FIN: The evaluation result of expression never changing
  - VAR: The evaluation result of expression changing
  - INC: The evaluation result of expression keeps increasing

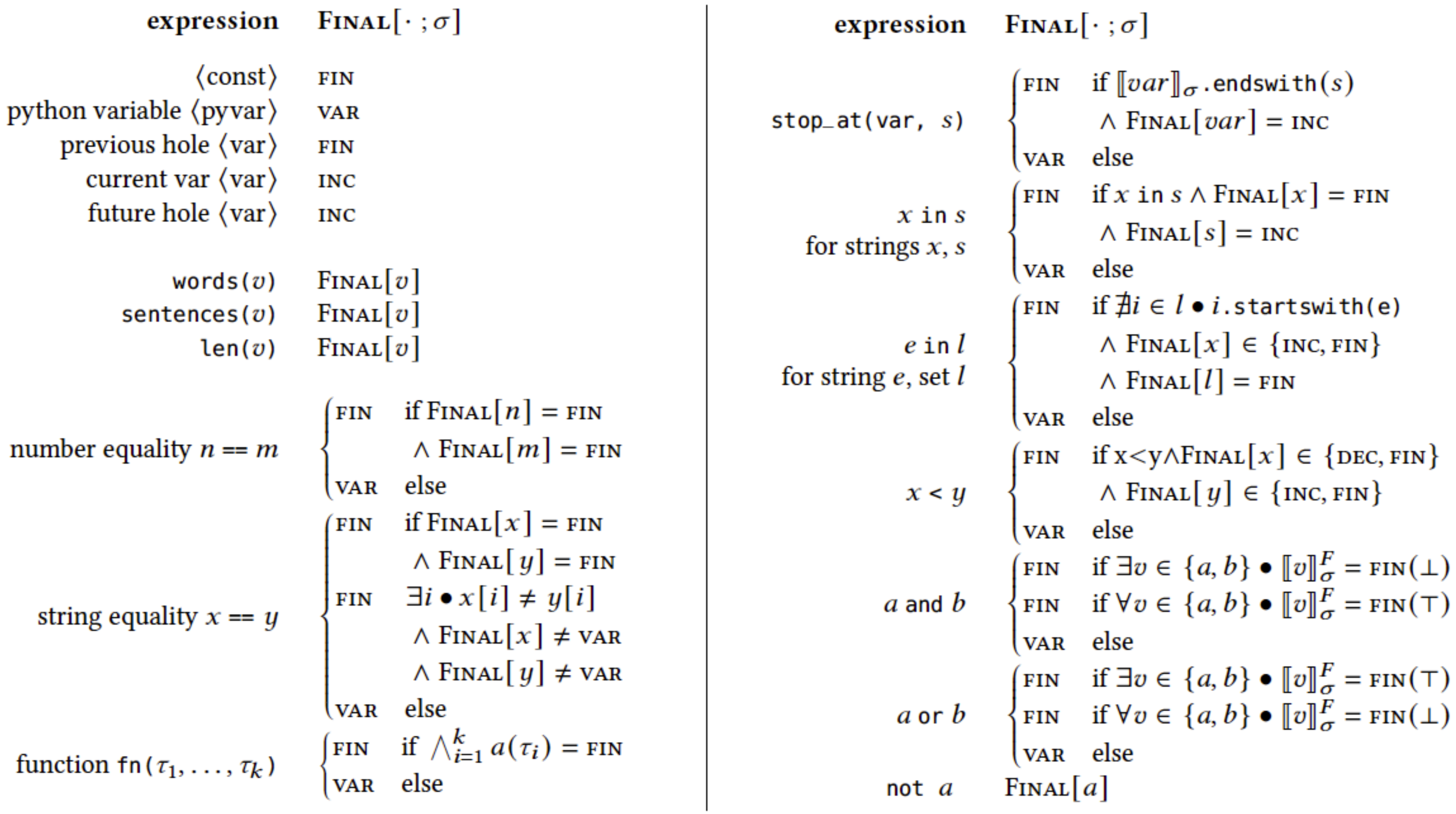
Final Semantics

- Evaluate expression as 4 states
    - FIN: The evaluation result of expression never changing
    - VAR: The evaluation result of expression changing
    - INC: The evaluation result of expression keeps increasing
        - len(output)

Final Semantics

- Evaluate expression as 4 states
  - FIN: The evaluation result of expression never changing
  - VAR: The evaluation result of expression changing
  - INC: The evaluation result of expression keeps increasing
    - len(output)
  - DEC: The evaluation result of expression keeps decreasing

Final Semantics



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

Follow Map

- Follow Map denotes **state transition** according to the trailing token

Follow Map

- Follow Map denotes **state transition** according to the trailing token
    - State: Final semantics's state + Possible values

Follow Map

- Follow Map denotes **state transition** according to the trailing token
    - State: Final semantics’s state + Possible values

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

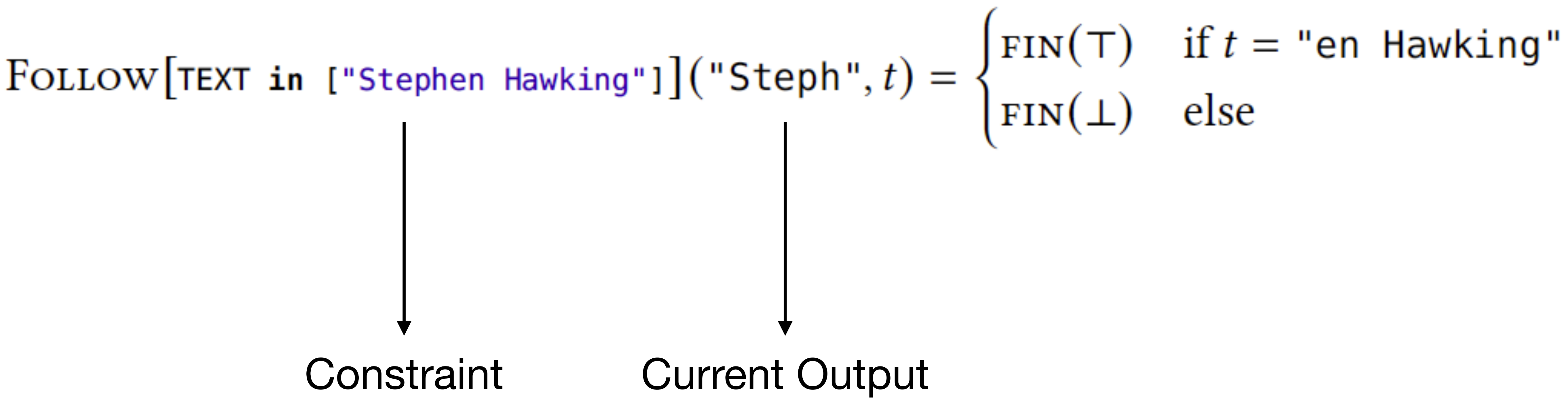
Follow Map

- Follow Map denotes **state transition** according to the trailing token
    - State: Final semantics’s state + Possible values

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

Follow Map

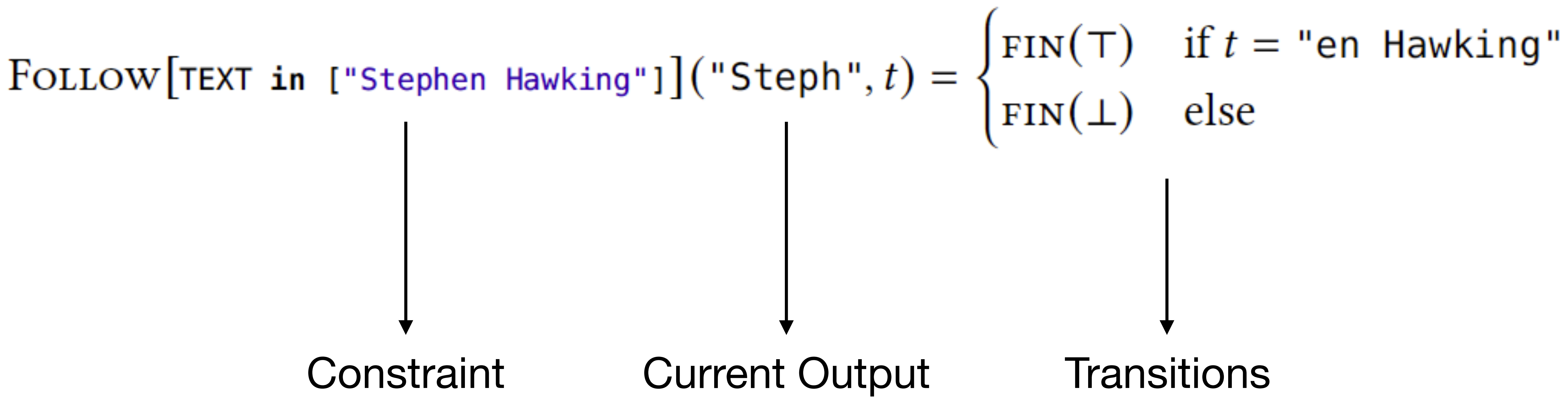
- Follow Map denotes **state transition** according to the trailing token
  - State: Final semantics's state + Possible values



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

Follow Map

- Follow Map denotes **state transition** according to the trailing token
  - State: Final semantics’s state + Possible values



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

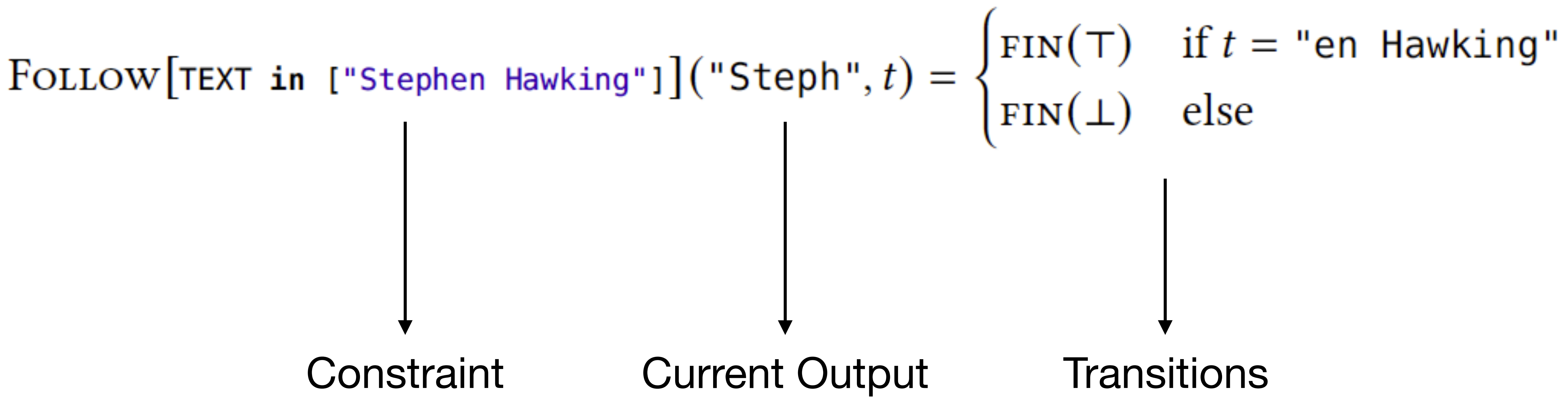
Constraint

Current Output

Transitions

Follow Map

- Follow Map denotes **state transition** according to the trailing token
    - State: Final semantics’s state + Possible values



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

Constraint

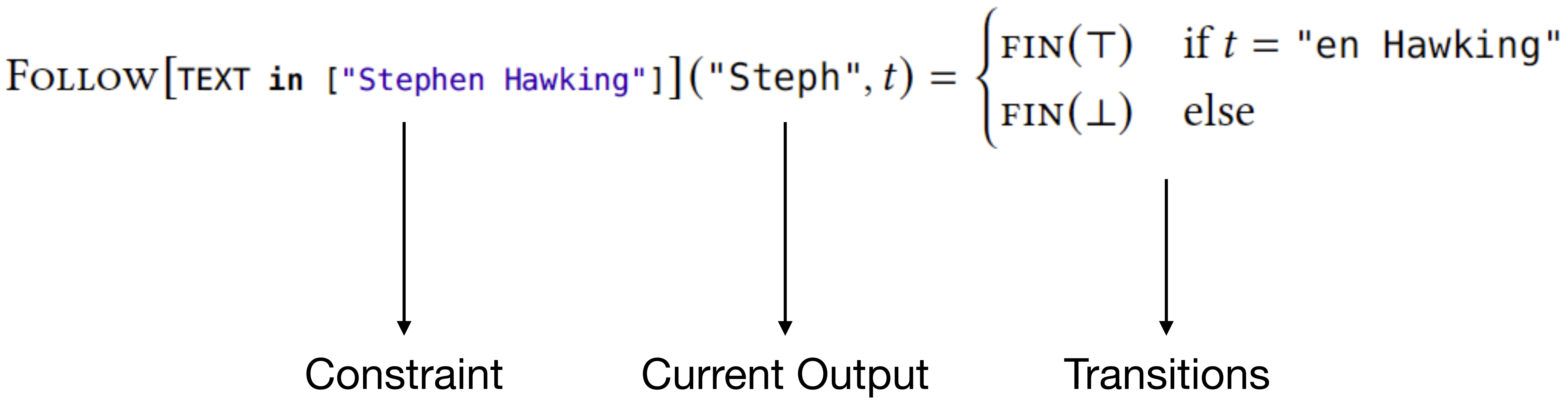
Current Output

Transitions

- If the trailing token is “en Hawking”, the evaluation result is **True (Top)** and **never changes (FIN)**

Follow Map

- Follow Map denotes **state transition** according to the trailing token
  - State: Final semantics’s state + Possible values

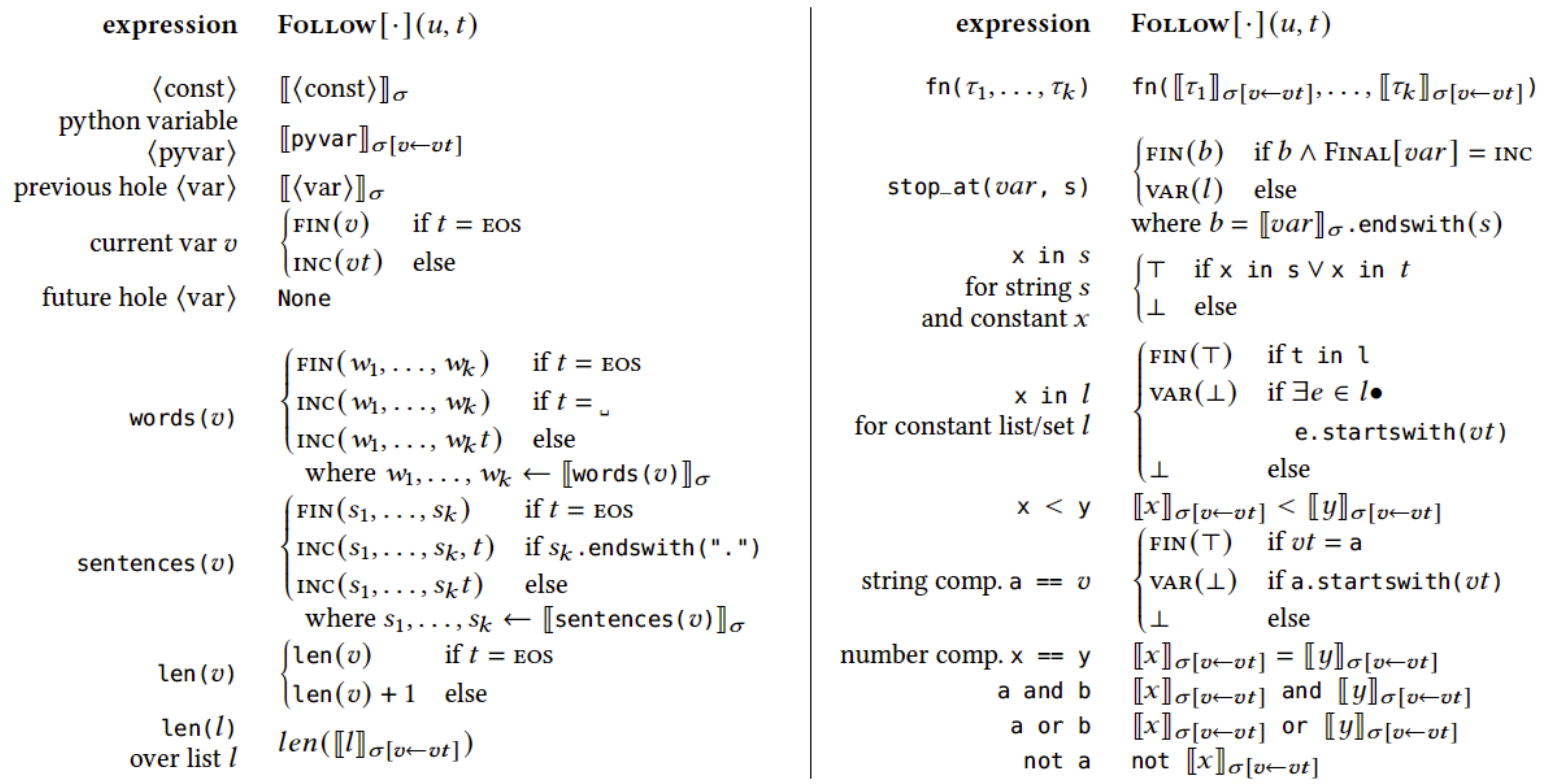


$$\textsc{Follow}[\texttt{TEXT in ["Stephen Hawking"]}](\texttt{"Steph"}, t) = \begin{cases} \textsc{fin}(\top) & \text{if } t = \texttt{"en Hawking"} \\ \textsc{fin}(\bot) & \text{else} \end{cases}$$

- If the trailing token is “en Hawking”, the evaluation result is **True (Top)** and **never changes (FIN)**

- Else, the evaluation result is **False (Bottom)** and **never changes (FIN)**

Follow Map



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

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

Soundness

- How to define soundness?

Soundness

- How to define soundness?
- Soundness: **Removed tokens are must unsafe**

Soundness

- How to define soundness?

- Soundness: **Removed tokens are must unsafe**

$$\forall t \in \mathcal{V} \bullet (\textsc{Follow}[e])(u, t) = \textsc{fin}(\bot) \Rightarrow [\![e]\!]_{\sigma[v \leftarrow ut]} = \textsc{fin}(\bot)$$

Soundness

- How to define soundness?
- Soundness: **Removed tokens are must unsafe**

$$\forall t \in \mathcal{V} \bullet (\textsc{Follow}[e])(u, t) = \textsc{fin}(\bot) \Rightarrow [\![e]\!]_{\sigma[v \leftarrow ut]} = \textsc{fin}(\bot)$$

If the result of the state according to the follow map semantics is **false for all tokens**

Soundness

- How to define soundness?
- Soundness: **Removed tokens are must unsafe**

$$\forall t \in \mathcal{V} \bullet (\textsc{Follow}[e])(u, t) = \textsc{fin}(\bot) \Rightarrow [\![e]\!]_{\sigma[v \leftarrow ut]} = \textsc{fin}(\bot)$$

Concrete evaluation result must be false

Soundness

- How to define soundness?

- Soundness: **Removed tokens are must unsafe**

$$\forall t \in \mathcal{V} \bullet (\textsc{Follow}[e])(u, t) = \textsc{fin}(\bot) \Rightarrow [\![e]\!]_{\sigma[v \leftarrow ut]} = \textsc{fin}(\bot)$$

Concrete evaluation result must be false

- It means the left tokens are still unsafe

Soundness

- How to define soundness?
- Soundness: **Removed tokens are must unsafe**

$$\forall t \in \mathcal{V} \bullet (\textsc{Follow}[e])(u, t) = \textsc{fin}(\bot) \Rightarrow [\![e]\!]_{\sigma[v \leftarrow ut]} = \textsc{fin}(\bot)$$

Concrete evaluation result must be false

- It means the left tokens are still unsafe
  - Backtracking!

Evaluation

- 3 Research Question

Evaluation

- 3 Research Question
  - Expressiveness: Can we implement advanced prompting techniques?

Evaluation

- 3 Research Question
  - Expressiveness: Can we implement advanced prompting techniques?
  - Performance: Can LMQL lower the number of model queries?

Evaluation

- 3 Research Question
  - Expressiveness: Can we implement advanced prompting techniques?
  - Performance: Can LMQL lower the number of model queries?
  - Accuracy: Does LMQL's constraint decoding affect task accuracy?

Evaluation

- 3 Research Question
  - Expressiveness: Can we implement advanced prompting techniques?
  - Performance: Can LMQL lower the number of model queries?
  - Accuracy: Does LMQL's constraint decoding affect task accuracy?
- Baseline

Evaluation

- 3 Research Question
  - Expressiveness: Can we implement advanced prompting techniques?
  - Performance: Can LMQL lower the number of model queries?
  - Accuracy: Does LMQL's constraint decoding affect task accuracy?
- Baseline
  - Huggingface's generate() API

Expressiveness

- Chain-of-Thought Prompting

```
argmax
   "Pick the odd word out: skirt, dress, pen, jacket."
   "skirt is clothing, dress is clothing, pen is an object, jacket is clothing."
   "So the odd one is pen."
   "Pick the odd word out: Spain, France, German, England, Singapore."
   "Spain is a country, France is a country, German is a language, ..."
   "So the odd one is German."
   "Pick the odd word out: {OPTIONS}"
   "[REASONING]"
   "[RESULT]"
from "EleutherAI/gpt-j-6B"
where
   not "Än" in REASONING and not "Pick" in REASONING and
   stops_at(REASONING, "Pick the odd word") and stops_at(REASONING, "Än") and
   stops_at(REASONING, "So the odd one") and stops_at(REASONING, ".") and len(WORDS(REASONING)) < 40
distribute
   RESULT over OPTIONS.split(", “)
```

Expressiveness

- Interactive Prompting

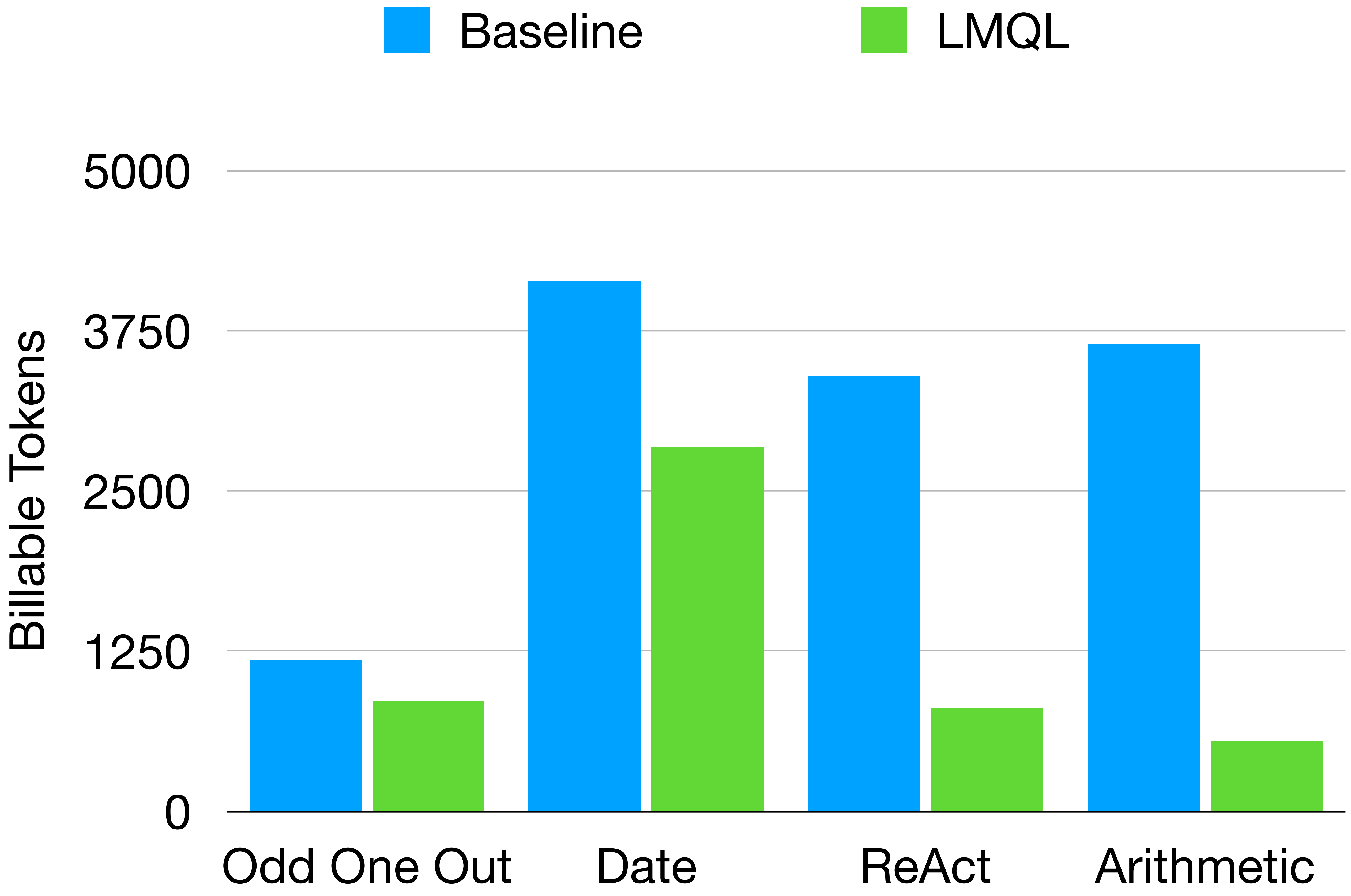
```
import wikipedia_utils
sample(no_repeat_ngram_size=3)
   "What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?"
   "Tho 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado ..."
   "Act 2: Search 'Colorado orogeny'"
   "Obs 2: The Colorado orogeny was an episode of mountain building (an orogeny) ..."
   "Tho 3: It does not mention the eastern sector. So I need to look up eastern sector."
   ...
   "Tho 4: High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft."
   "Act 5: Finish '1,800 to 7,000 ft'"
   "Where is Apple Computers headquartered?"
   for i in range(1024):
      "[MODE] {i}:"
   if MODE == "Tho":
      "[THOUGHT] "
   elif MODE == "Act":
      " [ACTION] '[SUBJECT]Än"
      if ACTION == "Search":
         result = wikipedia_utils.search(SUBJECT[:-1]) # cutting of the consumed '
         "Obs {i}: {result}Än"
      else:
         break # action must be FINISH
from "gpt2-xl"
where
   MODE in ["Tho", "Act"] and stops_at(THOUGHT, "Än") and
   ACTION in ["Search", "Finish"] and len(words(THOUGHT)) > 2 and
   stops_at(SUBJECT, "'") and not "Tho" in THOUGHT
```

Expressiveness

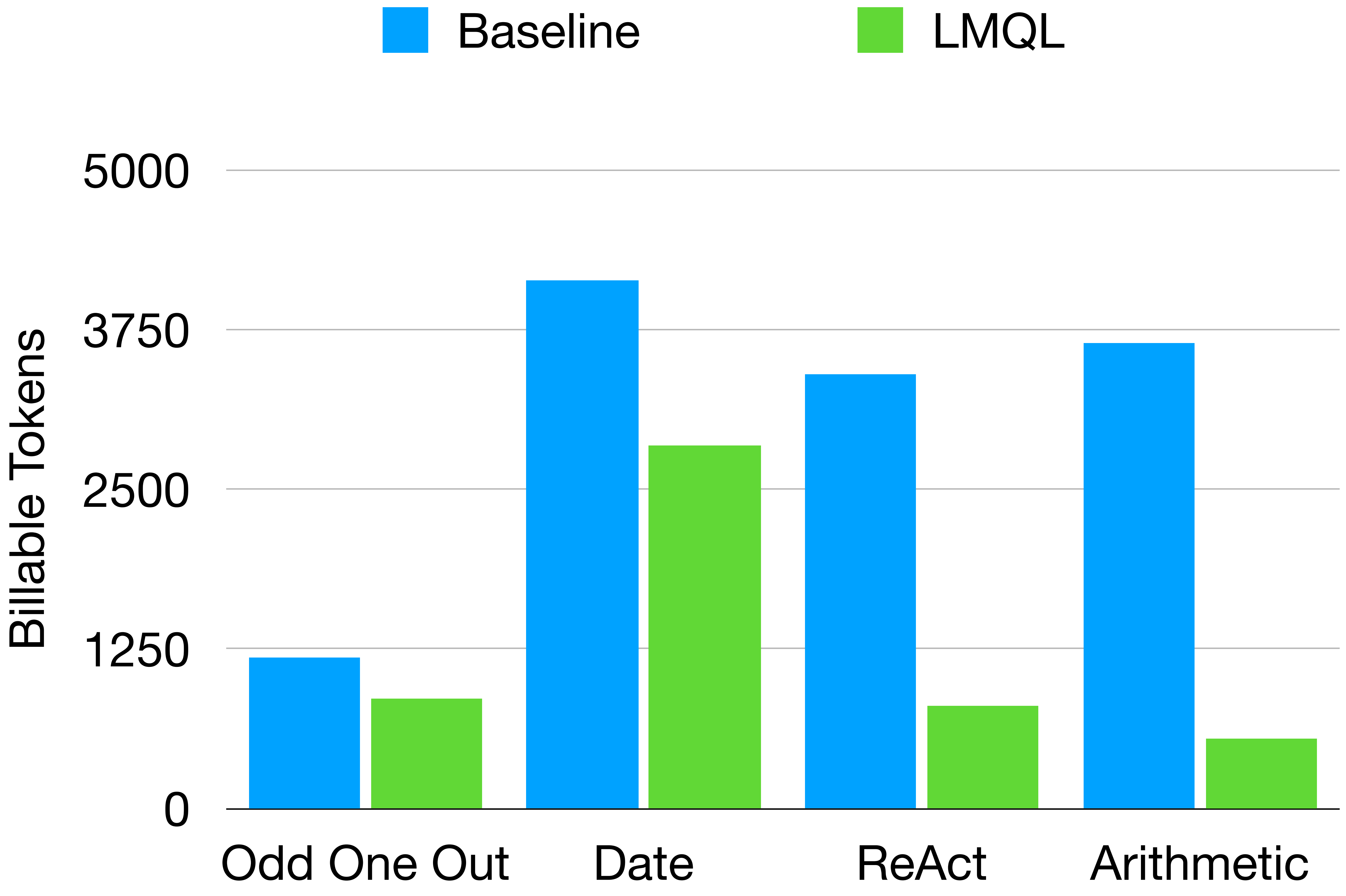
- Arithmetic Reasoning

```
argmax(distribution_batch_size=1, max_length=2048)
   "⟨few-shot examples⟩"
   "Q: {QUESTION}Än"
   "A: Let's think step by step.Än"
   for i in range(1024):
      "[REASON_OR_CALC]"
      if REASON_OR_CALC.endswith("<<"):
         " [EXPR] "
         result = calculator.run(EXPR)
         " {result} >> "
      elif REASON_OR_CALC.endswith("So the answer"):
         break
      " is [RESULT]"
from "EleutherAI/gpt-j-6B"
where
   int(RESULT) and
   stops_at(REASON_OR_CALC, "<<") and
   stops_at(EXPR, "=") and
   stops_at(REASON_OR_CALC, "So the answer")
```

Performance

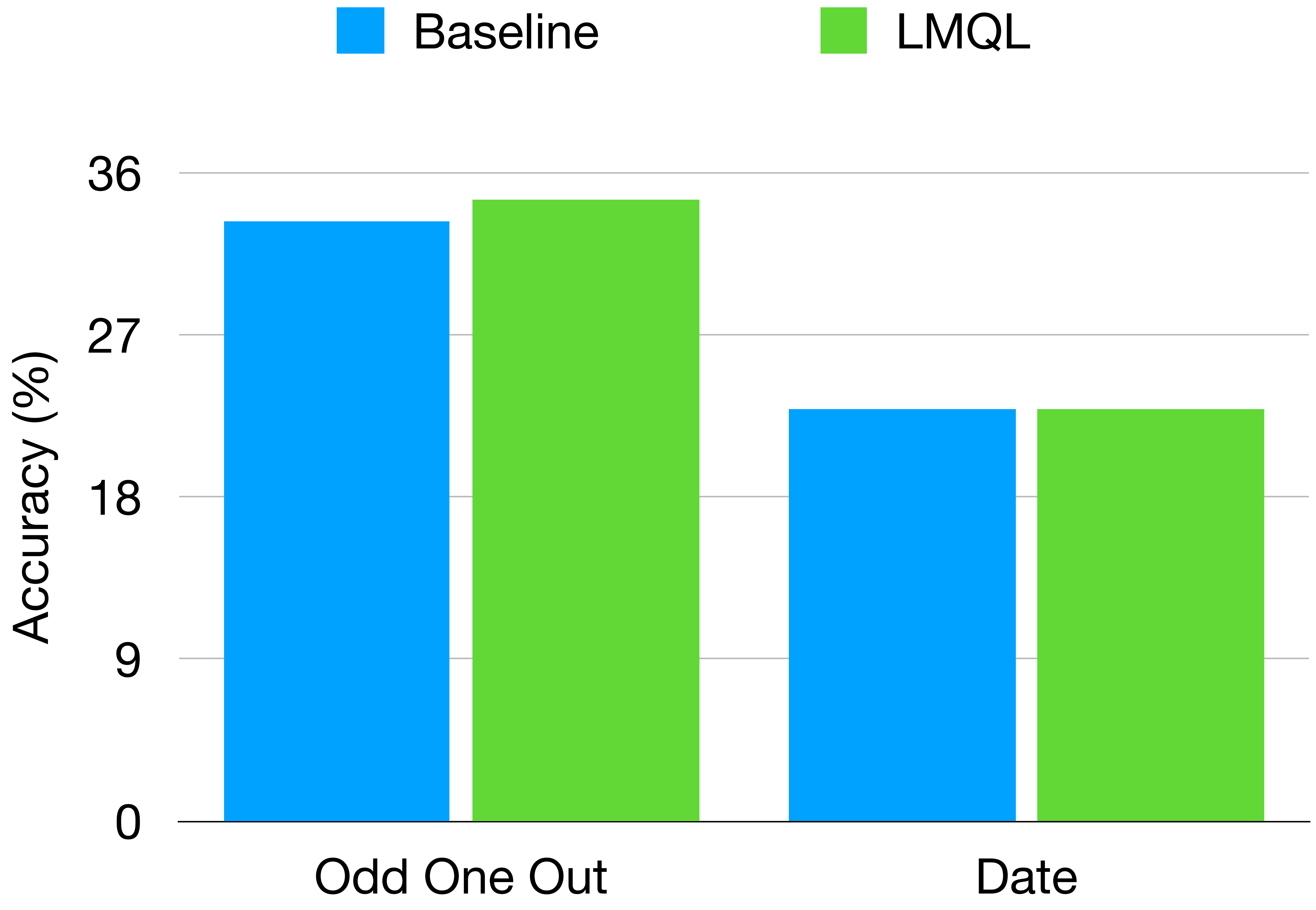


Performance

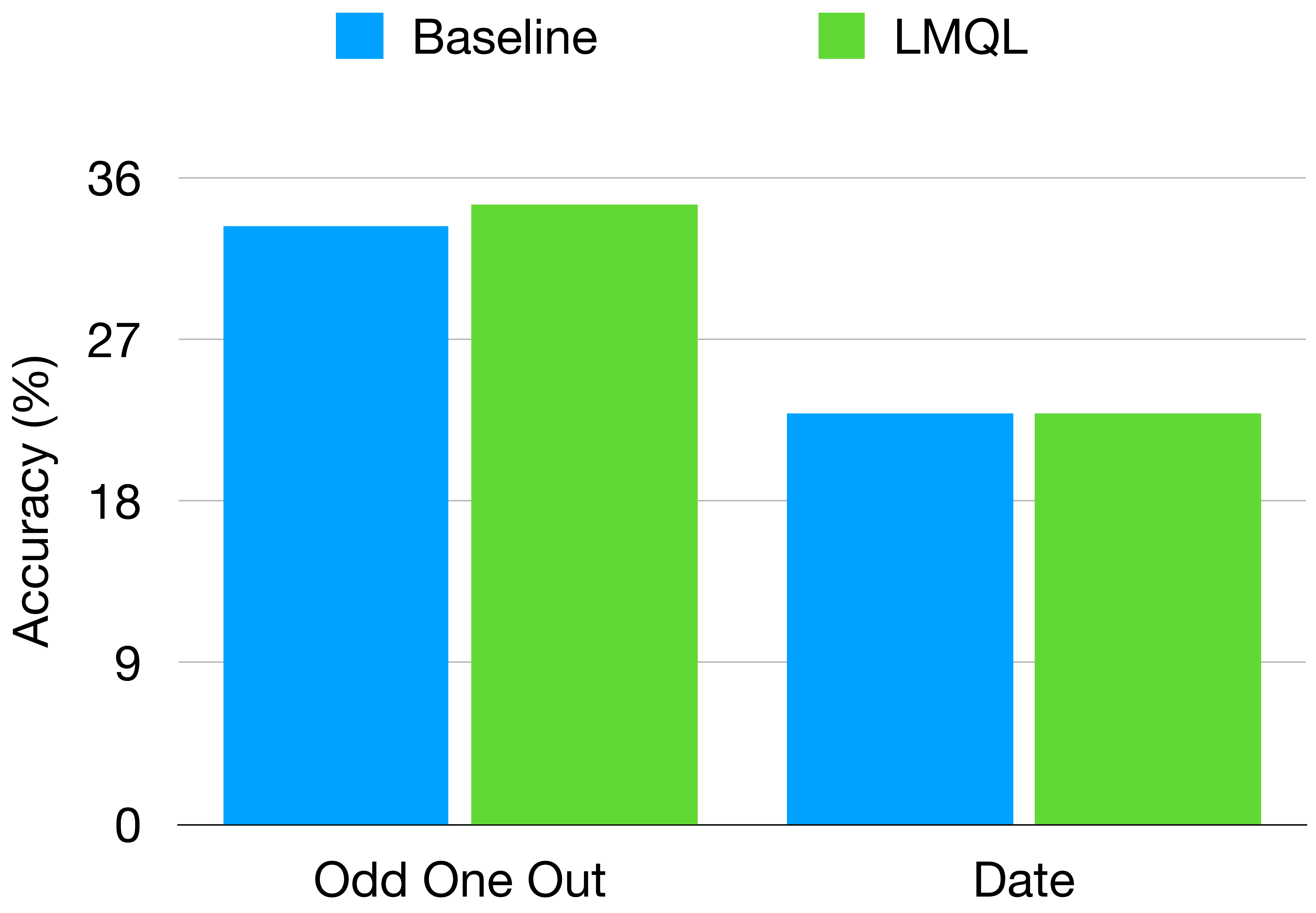


Using much fewer tokens through control flow and variables

Accuracy



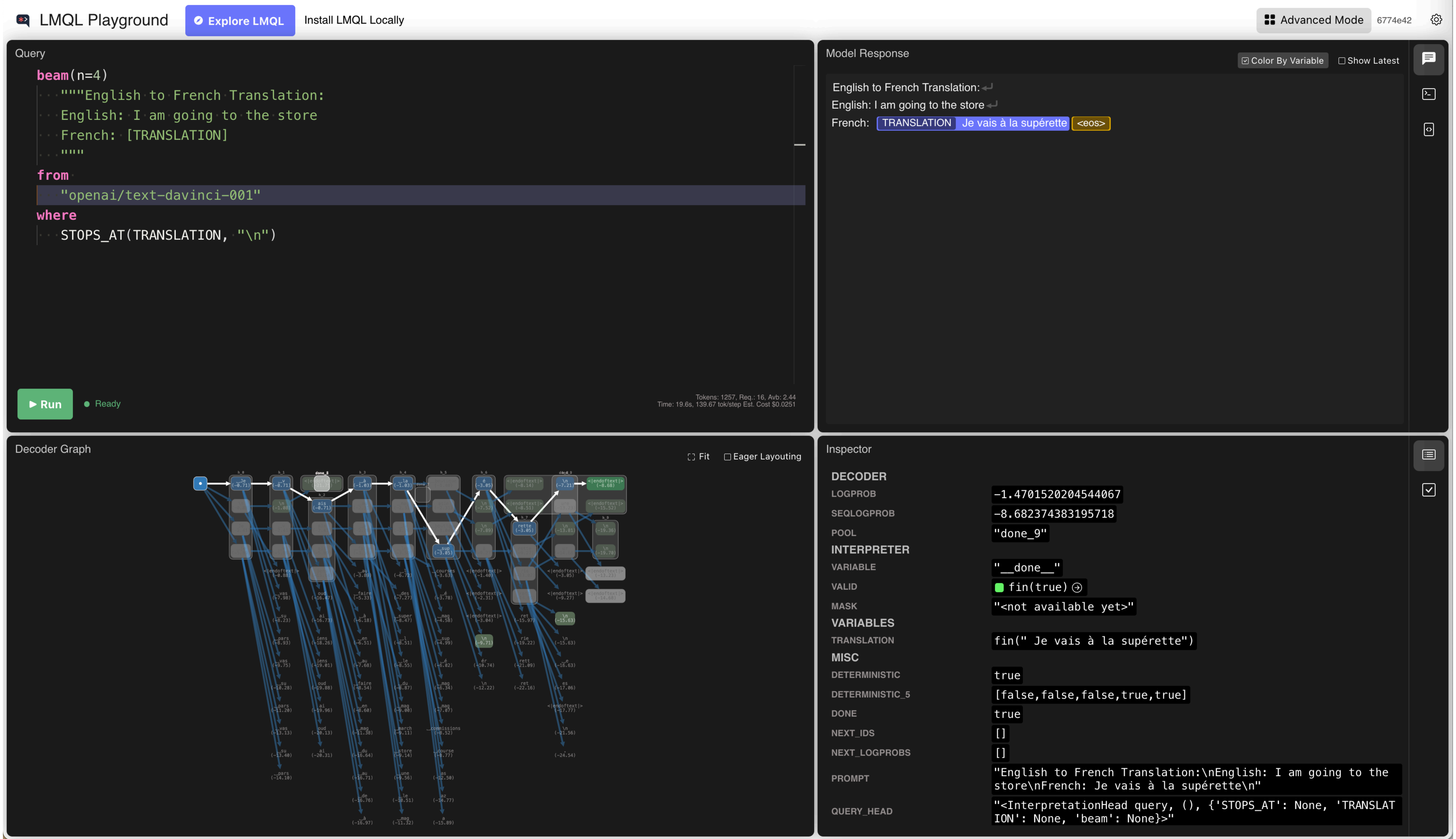
Accuracy



Accuracy isn’t affected by LMQL

Review

- Strong points
  - Formally defined constraint decoding method and soundness
  - Tools that can be useful in the wild
  - Serving nice visualization
- Weak points
  - Evaluated constraints are too easy
  - Not compositional (Nested Query)



```
beam(n=4)
    """English to French Translation:
    English: I am going to the store
    French: [TRANSLATION]
    """
from
    "openai/text-davinci-001"
where
    STOPS_AT(TRANSLATION, "\n")
```

Tokens: 1257, Req.: 16, Avb: 2.44
Time: 19.6s, 139.67 tok/step Est. Cost $0.0251

English to French Translation:
English: I am going to the store
French: TRANSLATION Je vais à la supérette <eos>

DECODER

LOGPROB	-1.4701520204544067
SEQLOGPROB	-8.682374383195718
POOL	"done_9"

INTERPRETER

VARIABLE	"__done__"
VALID	🟩 fin(true)
MASK	"<not available yet>"

VARIABLES

TRANSLATION	fin(" Je vais à la supérette")

MISC

DETERMINISTIC	true
DETERMINISTIC_5	[false,false,false,true,true]
DONE	true
NEXT_IDS	[]
NEXT_LOGPROBS	[]
PROMPT	"English to French Translation:\nEnglish: I am going to the store\nFrench: Je vais à la supérette\n"
QUERY_HEAD	"<InterpretationHead query, (), {'STOPS_AT': None, 'TRANSLATION': None, 'beam': None}>"

Summary

- The paper suggests **a new paradigm** of interacting with the language model and its **implementation**
    - **LMP**: Language Model + Programming
    - **LMQL**: Add abstraction of language model to programming language
- Provide **control flow + variable + constraint decoding**
- We can use LMQL for **most of the existing prompting strategies**
- Lowering cost and slightly increasing accuracy

Example2

```
argmax
   "A list of things not to forget
     when travelling:\n"
   things = []
   for _ in range(2):
     "- [THING]\n"
     things.append(THING)
from "gpt-3"
where
   len(words(THING)) <= 2
```

Example2

```
argmax  →  We are going to use greedy decoding
  "A list of things not to forget
    when travelling:\n"
  things = []
  for _ in range(2):
    "- [THING]\n"
    things.append(THING)
from "gpt-3"
where
  len(words(THING)) <= 2
```

Example2

```
argmax
   "A list of things not to forget
     when travelling:\n"
   things = []
   for _ in range(2):
      "- [THING]\n"
      things.append(THING)
from "gpt-3"
where
   len(words(THING)) <= 2
```

from "gpt-3" → Use language model gpt2-large

Example2

```
argmax
   "A list of things not to forget
     when travelling:\n"
   things = []
   for _ in range(2):
      "- [THING]\n"
      things.append(THING)
from "gpt-3"
where
   len(words(THING)) <= 2
```

→ THING is 2 words or less

Example2

```
argmax
   "A list of things not to forget
     when travelling:\n"
   things = []
   for _ in range(2):
      "- [THING]\n"
      things.append(THING)
from "gpt-3"
where
   len(words(THING)) <= 2
```

→ Add variable **things** to environment

Environment

[redacted]

Prompt

“A list of …”

Example2

```
argmax
   "A list of things not to forget
     when travelling:\n"
   things = []
   for _ in range(2):
      "- [THING]\n"
      things.append(THING)
from "gpt-3"
where
   len(words(THING)) <= 2
```

"- [THING]\n" → Call LM and Assign response to **THING**

Environment

`things = []`

Prompt

“A list of …
_ ”

Example2

```
argmax
   "A list of things not to forget
     when travelling:\n"
   things = []
   for _ in range(2):
      "- [THING]\n"
      things.append(THING)
from "gpt-3"
where
   len(words(THING)) <= 2
```

Append **THING** to **things**

Environment

```
things = []
THING = “money”
```

Prompt

“A list of …
- money”

Example2

```
argmax
   "A list of things not to forget
     when travelling:\n"
   things = []
   for _ in range(2):
      "- [THING]\n"
      things.append(THING)
from "gpt-3"
where
   len(words(THING)) <= 2
```

"- [THING]\n" → Append **THING** to **things**

Environment

```
things =
["money"]
THING = "money"
```

Prompt

```
"A list of …
- money
- "
```