Multi-modal Program Inference:

A Marriage of Pre-trained Language Models and Component-based Synthesis

Kia Rahmani, Mohammad Raza, Sumit Gulwani, Vu Le,
Daniel Morris, Arjun Radhakrishna, Gustavo Soares, Ashish Tiwari

CHICAGO
SPLASH2021

Microsoft

PURDUE UNIVERSITY

THE STORY OF TRANSFORMERS

PRE-TRAINED NATURAL LANGUAGE MODELS (PTM)

- BERT, ELMo and ERNIE
- Neural architectures optimized for **language understanding**

ERNIE: Enhanced Language Representation with Informative Entities

Zhengyan Zhang[1,2,3*], Xu Han[1,2,3*], Zhiyuan Liu[1,2,3†], Xin Jiang[4], Maosong Sun[1,2,3], Qun Liu[4]
[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]Institute for Artificial Intelligence, Tsinghua University, Beijing, China
[3]State Key Lab on Intelligent Technology and Systems, Tsinghua University, Beijing, China
[4]Huawei Noah's Ark Lab
{zhangzhengyan14,hanxu17}@mails.tsinghua.edu.cn

arXiv:1905.07129v3 [cs.CL] 4 Jun 2019

Abstract

Neural language representation models such as BERT pre-trained on large-scale corpora can well capture rich semantic patterns from plain text, and be fine-tuned to consistently improve the performance of various NLP tasks. However, the existing pre-trained language models rarely consider incorporating knowledge graphs (KGs), which can provide rich structured knowledge facts for better language understanding. We argue that informative entities in KGs can enhance language representation with external knowledge. In this paper, we utilize both large-scale textual corpora and KGs to train an enhanced language representation model (ERNIE), which can take full advantage of lexical, syntactic, and knowledge information simultaneously. The experimental results have demonstrated that ERNIE achieves significant improvements on various knowledge-driven tasks, and meanwhile is comparable with the state-of-the-art model BERT on other common NLP tasks. The source code and experiment details of this paper can be obtained from https://github.com/thunlp/ERNIE.

1 Introduction

Pre-trained language representation models, including feature-based (Mikolov et al., 2013; Pennington et al., 2014; Peters et al., 2017, 2018) and fine-tuning (Dai and Le, 2015; Howard and Ruder, 2018; Radford et al., 2018; Devlin et al., 2019) approaches, can capture rich language information from text and then benefit many NLP applications. BERT (Devlin et al., 2019), as one of the most recently proposed models, obtains the state-of-the-art results on various NLP applications by simple fine-tuning, including named entity recognition (Sang and De Meulder, 2003), question answering (Rajpurkar et al., 2016; Zellers et al., 2018), natural language inference (Bowman et al., 2015), and text classification (Wang et al., 2018).

Bob Dylan wrote *Blowin' in the Wind* in 1962, and wrote *Chronicles: Volume One* in 2004.

Figure 1: An example of incorporating extra knowledge information for language understanding. The solid lines present the existing knowledge facts. The red dotted lines present the facts extracted from the sentence in red. The green dot-dash lines present the facts extracted from the sentence in green.

Although pre-trained language representation models have achieved promising results and worked as a routine component in many NLP tasks, they neglect to incorporate knowledge information for language understanding. As shown in Figure 1, without knowing *Blowin' in the Wind* and *Chronicles: Volume One* are *song* and *book* respectively, it is difficult to recognize the two occupations of *Bob Dylan*, i.e., songwriter and writer, on the entity typing task. Furthermore, it is nearly impossible to extract the fine-grained relations, such as composer and author on the relation classification task. For the existing pre-trained language representation models, these two sentences are syntactically ambiguous, like "*UNK* wrote *UNK* in *UNK*". Hence, considering rich knowledge information can lead to better language understanding and accordingly benefits various knowledge-driven applications, e.g. entity typing and relation classification.

For incorporating external knowledge into language representation models, there are two main

[*] indicates equal contribution
[†] Corresponding author: Z.Liu(liuzy@tsinghua.edu.cn)

2019

Deep contextualized word representations

Matthew E. Peters[†], Mark Neumann[†], Mohit Iyyer[†], Matt Gardner[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], Kenton Lee[*], Luke Zettlemoyer[†*]
{csquared,kentonl,lsz}@cs.washington.edu

[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington

Abstract

We introduce a new type of *deep contextualized* word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). Our word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. We show that these representations can be easily added to existing models and significantly improve the state of the art across six challenging NLP problems, including question answering, textual entailment and sentiment analysis. We also present an analysis showing that exposing the deep internals of the pre-trained network is crucial, allowing downstream models to mix different types of semi-supervision signals.

1 Introduction

Pre-trained word representations (Mikolov et al., 2013; Pennington et al., 2014) are a key component in many neural language understanding models. However, learning high quality representations can be challenging. They should ideally model both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). In this paper, we introduce a new type of *deep contextualized* word representation that directly addresses both challenges, can be easily integrated into existing models, and significantly improves the state of the art in every considered case across a range of challenging language understanding problems.

Our representations differ from traditional word type embeddings in that each token is assigned a representation that is a function of the entire input sentence. We use vectors derived from a bidirectional LSTM that is trained with a coupled language model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations. Unlike previous approaches for learning contextualized word vectors (Peters et al., 2017; McCann et al., 2017), ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM. More specifically, we learn a linear combination of the vectors stacked above each input word for each end task, which markedly improves performance over just using the top LSTM layer.

Combining the internal states in this manner allows for very rich word representations. Using intrinsic evaluations, we show that the higher-level LSTM states capture context-dependent aspects of word meaning (e.g., they can be used without modification to perform well on supervised word sense disambiguation tasks) while lower-level states model aspects of syntax (e.g., they can be used to do part-of-speech tagging). Simultaneously exposing all of these signals is highly beneficial, allowing the learned models select the types of semi-supervision that are most useful for each end task.

Extensive experiments demonstrate that ELMo representations work extremely well in practice. We first show that they can be easily added to existing models for six diverse and challenging language understanding problems, including textual entailment, question answering and sentiment analysis. The addition of ELMo representations alone significantly improves the state of the art in every case, including up to 20% relative error reductions. For tasks where direct comparisons are possible, ELMo outperforms CoVe (McCann et al., 2017), which computes contextualized representations using a neural machine translation encoder. Finally, an analysis of both ELMo and CoVe reveals that deep representations outperform

2227

Proceedings of NAACL-HLT 2018, pages 2227–2237
New Orleans, Louisiana, June 1 - 6, 2018. ©2018 Association for Computational Linguistics

2018

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

arXiv:1810.04805v2 [cs.CL] 24 May 2019

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

1 Introduction

Language model pre-training has been shown to be effective for improving many natural language processing tasks (Dai and Le, 2015; Peters et al., 2018a; Radford et al., 2018; Howard and Ruder, 2018). These include sentence-level tasks such as natural language inference (Bowman et al., 2015; Williams et al., 2018) and paraphrasing (Dolan and Brockett, 2005), which aim to predict the relationships between sentences by analyzing them holistically, as well as token-level tasks such as named entity recognition and question answering, where models are required to produce fine-grained output at the token level (Tjong Kim Sang and De Meulder, 2003; Rajpurkar et al., 2016).

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

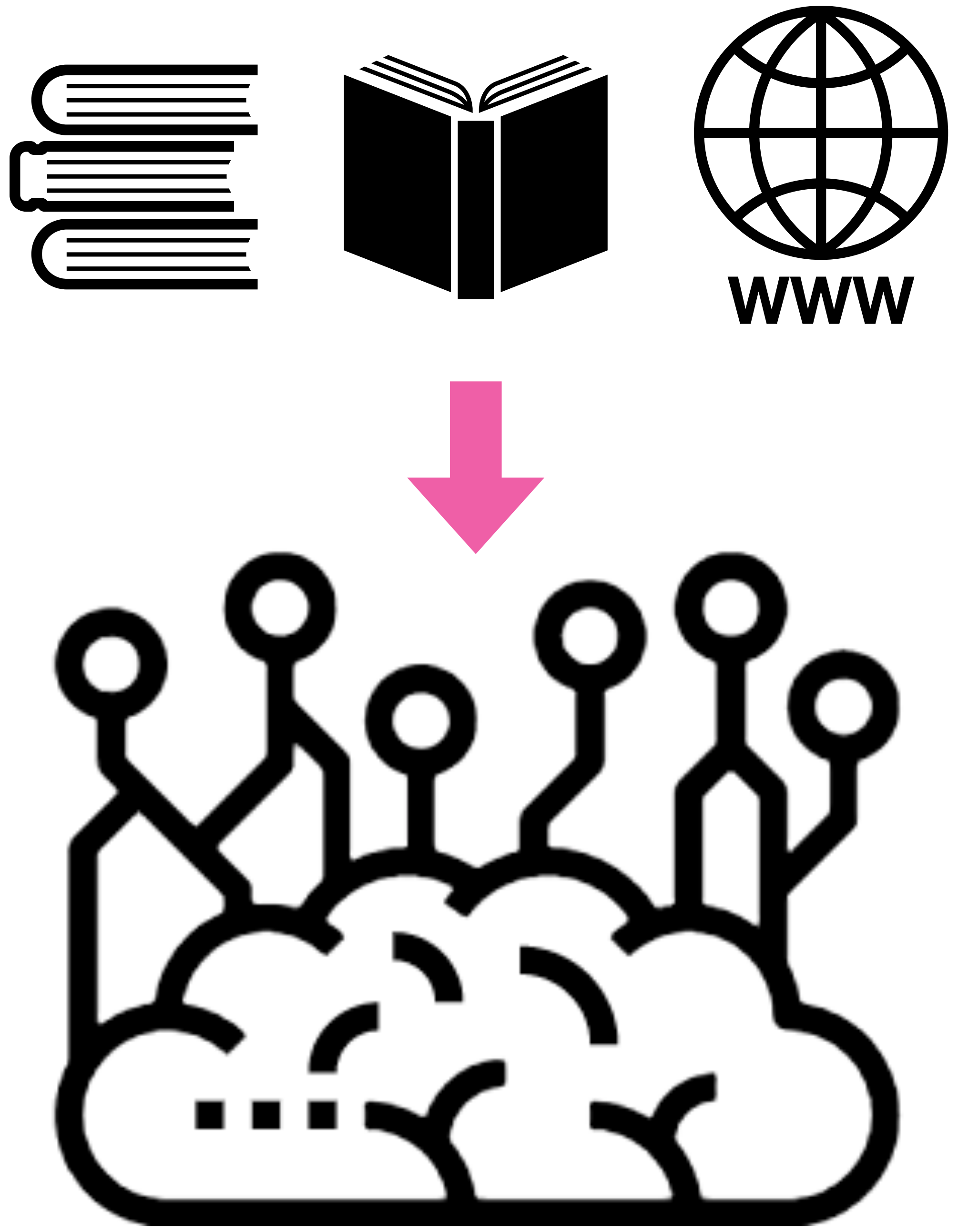
We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restrictions are sub-optimal for sentence-level tasks, and could be very harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions.

In this paper, we improve the fine-tuning based approaches by proposing BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. BERT alleviates the previously mentioned unidirectionality constraint by using a "masked language model" (MLM) pre-training objective, inspired by the Cloze task (Taylor, 1953). The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked

2018

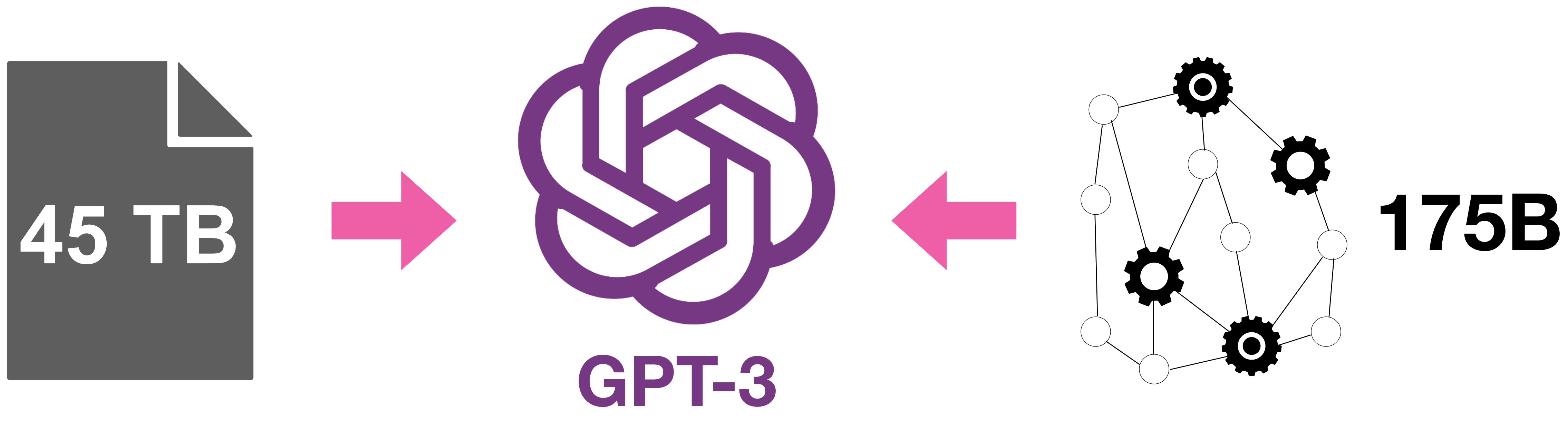
PRE-TRAINED NATURAL LANGUAGE MODELS (PTM)

- BERT, ELMo and ERNIE
- Neural architectures optimized for **language understanding**
- Trained on a large corpus of text



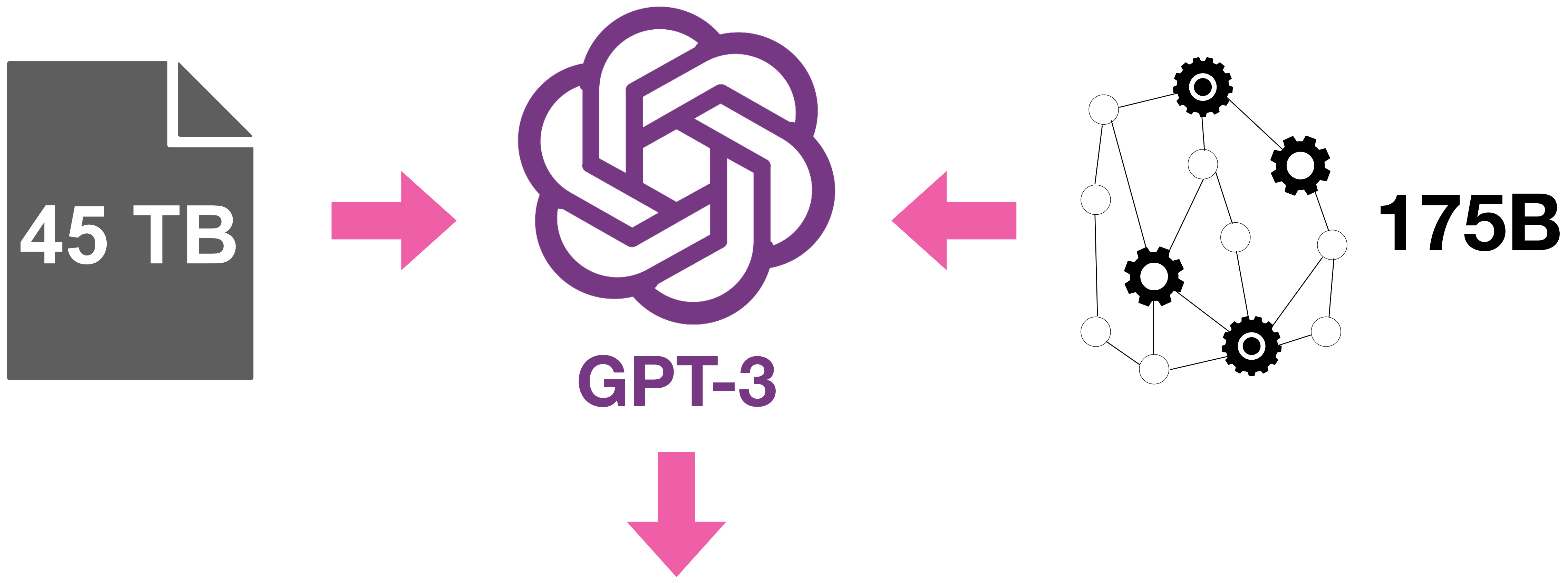
GPT-3 FROM OPEN AI

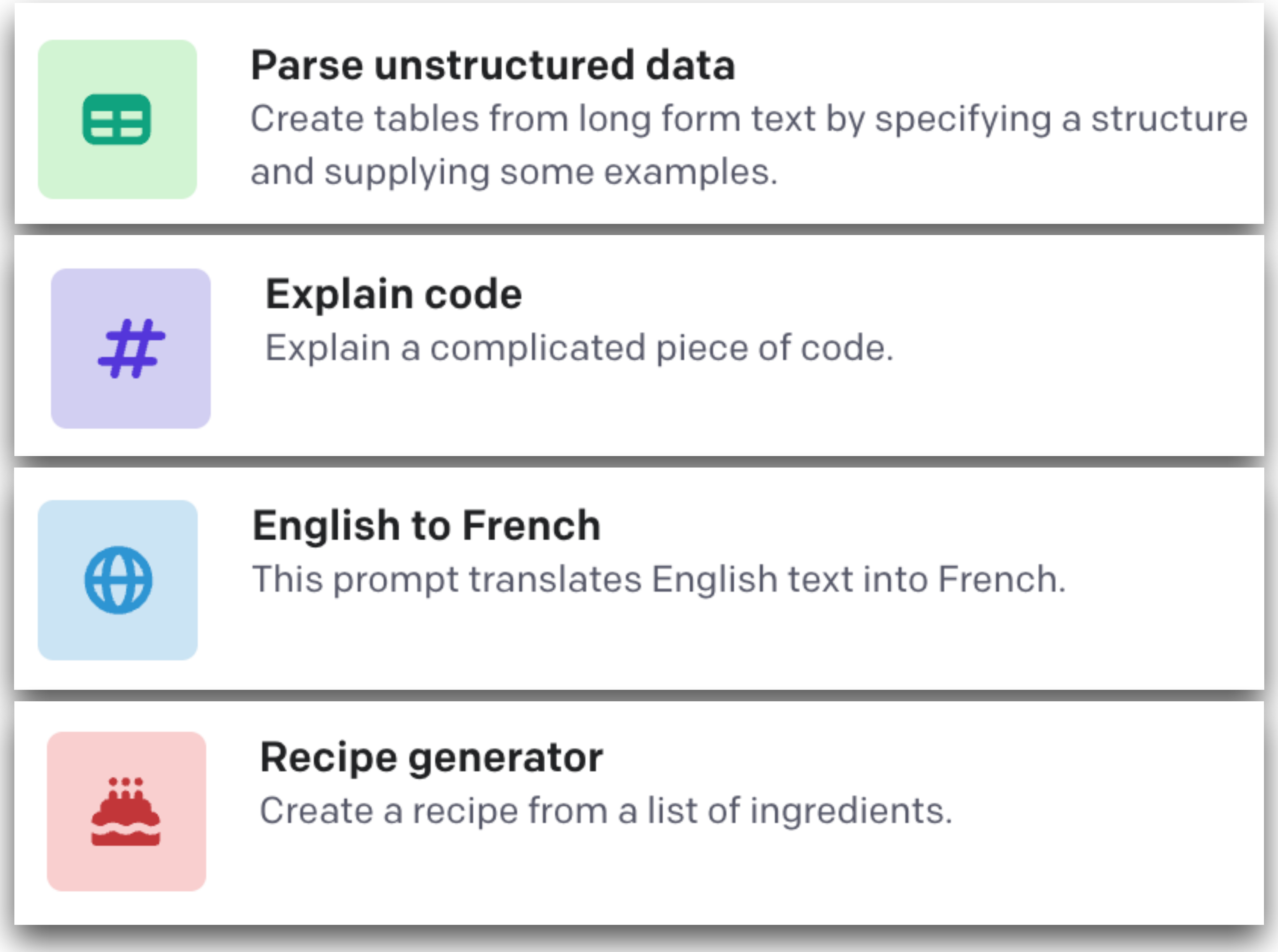
- Latest model from GPT-n series
- Deployed in 300 applications
  - Generates 4.5B words per day



GPT-3 FROM OPEN AI

- Latest model from GPT-n series
- Deployed in 300 applications
  - Generates 4.5B words per day
- Can be “taught” by showing a few examples of the tasks
- **Few-shot Learning**
- (Very!) diverse use-cases




Parse unstructured data
Create tables from long form text by specifying a structure and supplying some examples.

Explain code
Explain a complicated piece of code.

English to French
This prompt translates English text into French.

Recipe generator
Create a recipe from a list of ingredients.

GPT-3 FOR CODE GENERATION

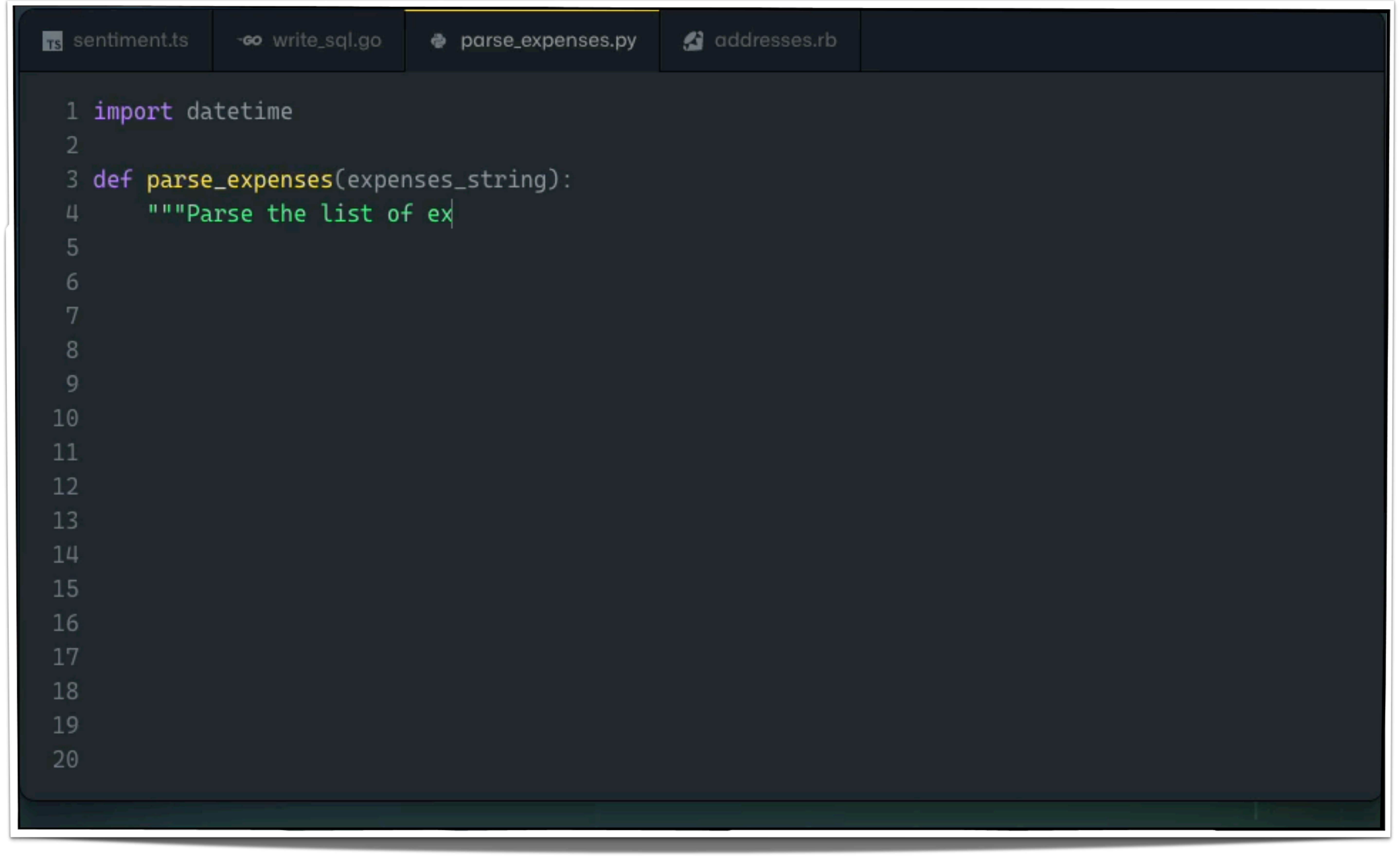
- “*Rise of AI language models in programming automation*”

GPT-3 FOR CODE GENERATION

- *“Rise of AI language models in programming automation”*
- Github Copilot
  - A dozen programming languages

GitHub Copilot

Powered by OpenAI Codex

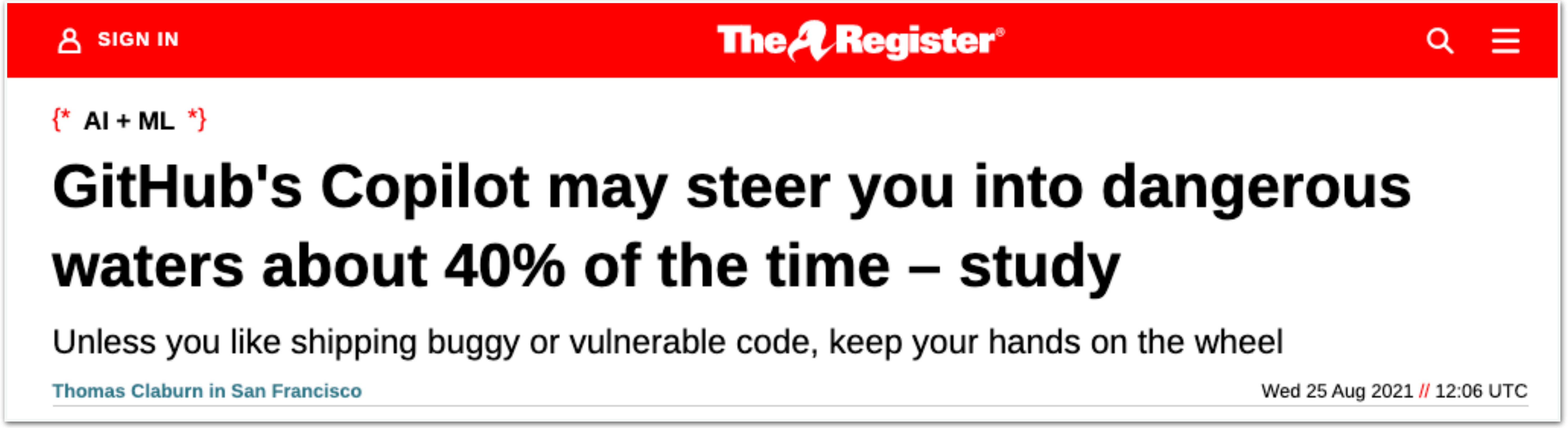


```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of ex
```

GPT-3 FOR CODE GENERATION (CONT'D)

- *"Rise of AI language models in programming automation"*
- Github Copilot
  - A dozen programming languages
- Limited Precision



SIGN IN

The Register

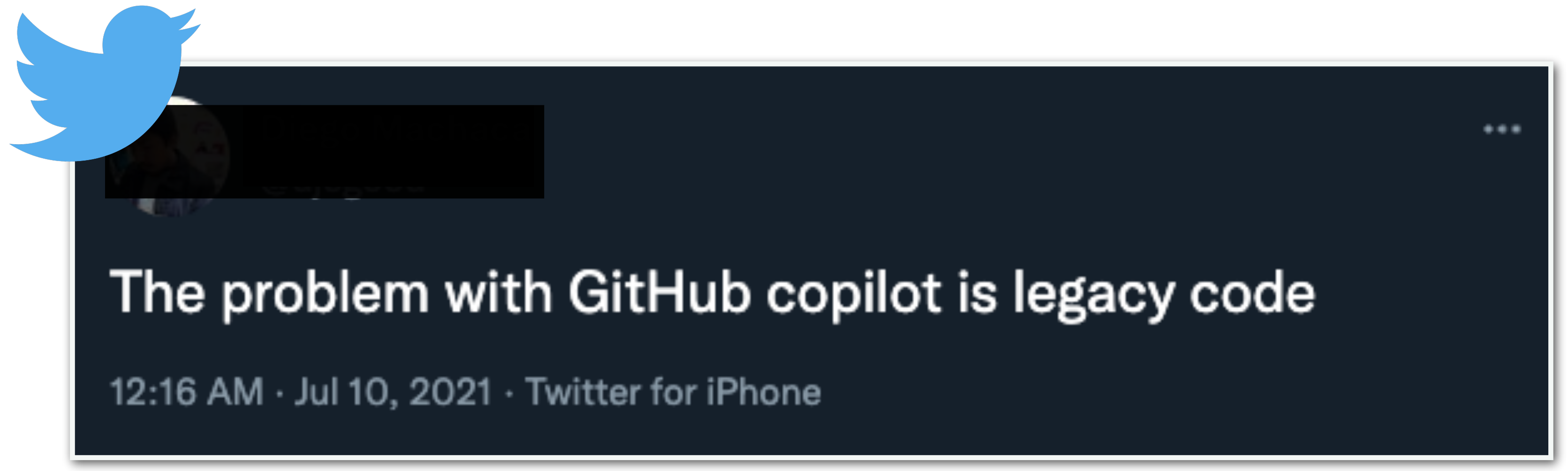
{* AI + ML *}

GitHub's Copilot may steer you into dangerous waters about 40% of the time – study

Unless you like shipping buggy or vulnerable code, keep your hands on the wheel

Thomas Claburn in San Francisco

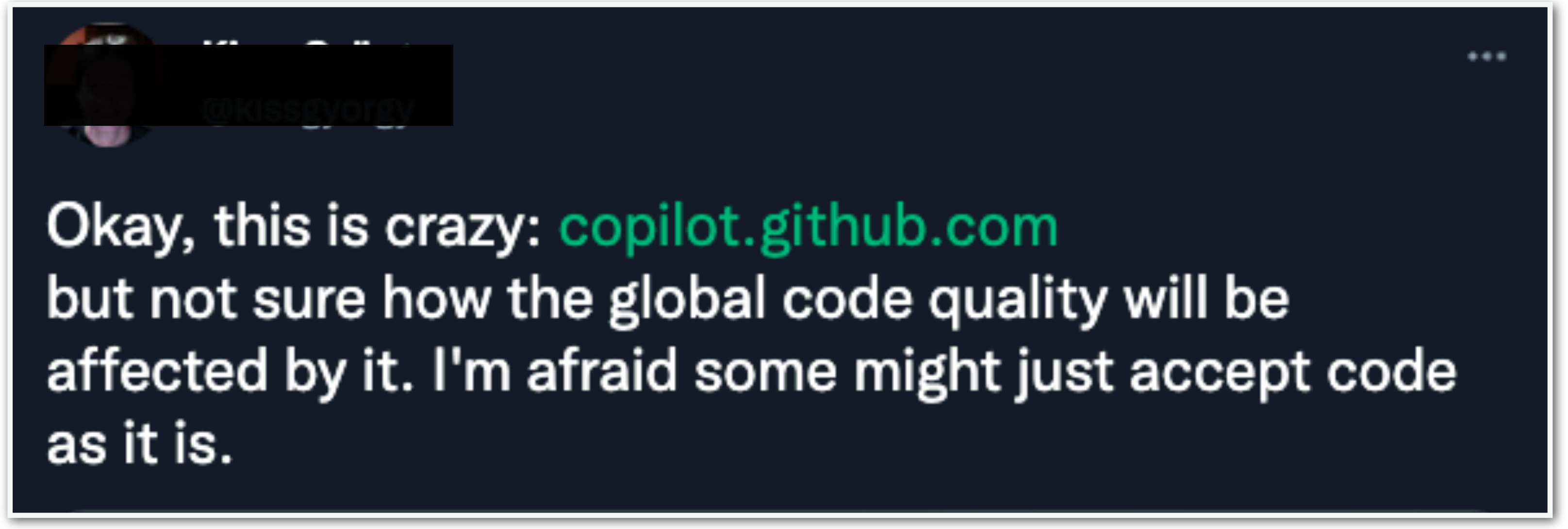
Wed 25 Aug 2021 // 12:06 UTC



[redacted]

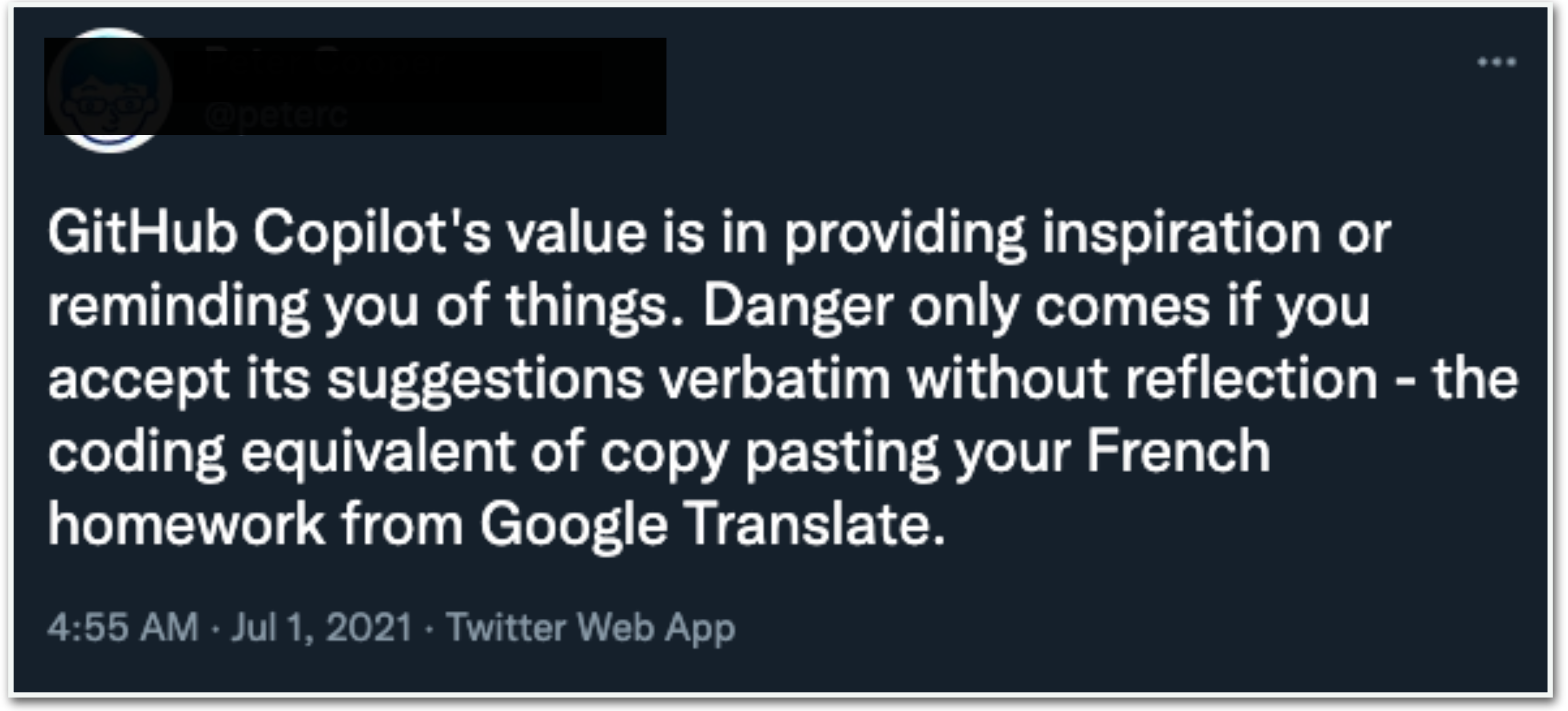
The problem with GitHub copilot is legacy code

12:16 AM · Jul 10, 2021 · Twitter for iPhone



[redacted]

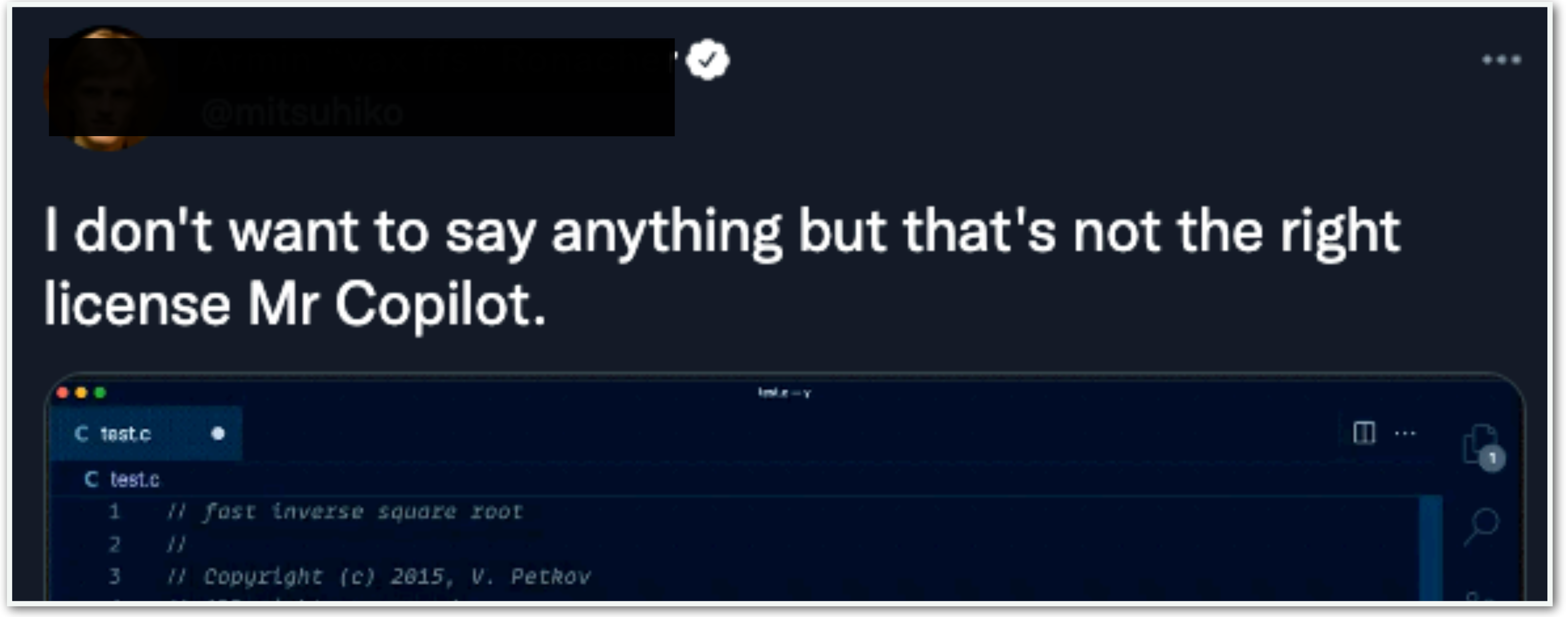
Okay, this is crazy: copilot.github.com
but not sure how the global code quality will be affected by it. I'm afraid some might just accept code as it is.



[redacted]

GitHub Copilot's value is in providing inspiration or reminding you of things. Danger only comes if you accept its suggestions verbatim without reflection - the coding equivalent of copy pasting your French homework from Google Translate.

4:55 AM · Jul 1, 2021 · Twitter Web App



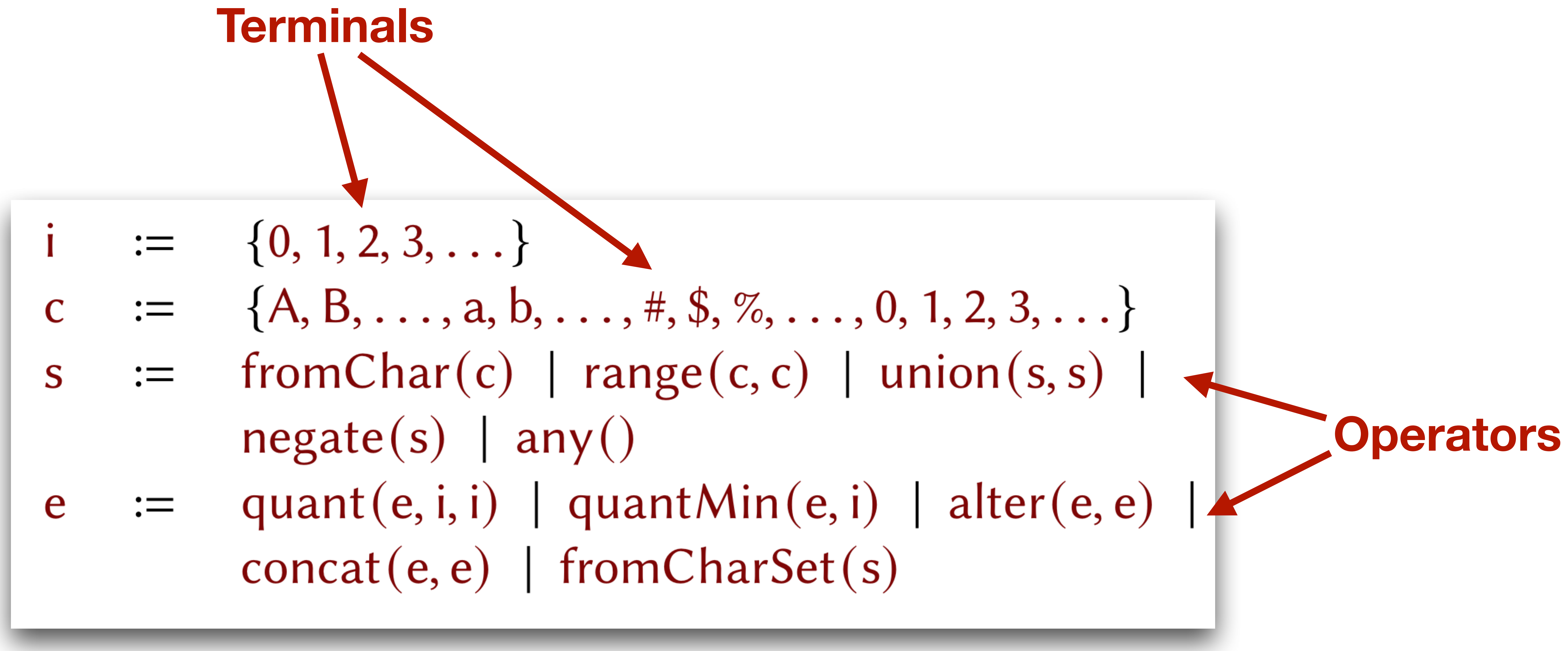
[redacted]

I don't want to say anything but that's not the right license Mr Copilot.

```
// fast inverse square root
//
// Copyright (c) 2015, V. Petkov
```

FIRST HAND EXPERIMENTS WITH (NL → CODE)

- Domain of Regular Expressions (REGEX)
  - concise search patterns
  - terminals and operators



Terminals

$$
\begin{aligned}
\mathrm{i} &:= \{0, 1, 2, 3, \ldots\} \\
\mathrm{c} &:= \{A, B, \ldots, a, b, \ldots, \#, \$, \%, \ldots, 0, 1, 2, 3, \ldots\} \\
\mathrm{s} &:= \mathrm{fromChar(c)} \mid \mathrm{range(c, c)} \mid \mathrm{union(s, s)} \mid \\
&\quad\ \mathrm{negate(s)} \mid \mathrm{any()} \\
\mathrm{e} &:= \mathrm{quant(e, i, i)} \mid \mathrm{quantMin(e, i)} \mid \mathrm{alter(e, e)} \mid \\
&\quad\ \mathrm{concat(e, e)} \mid \mathrm{fromCharSet(s)}
\end{aligned}
$$

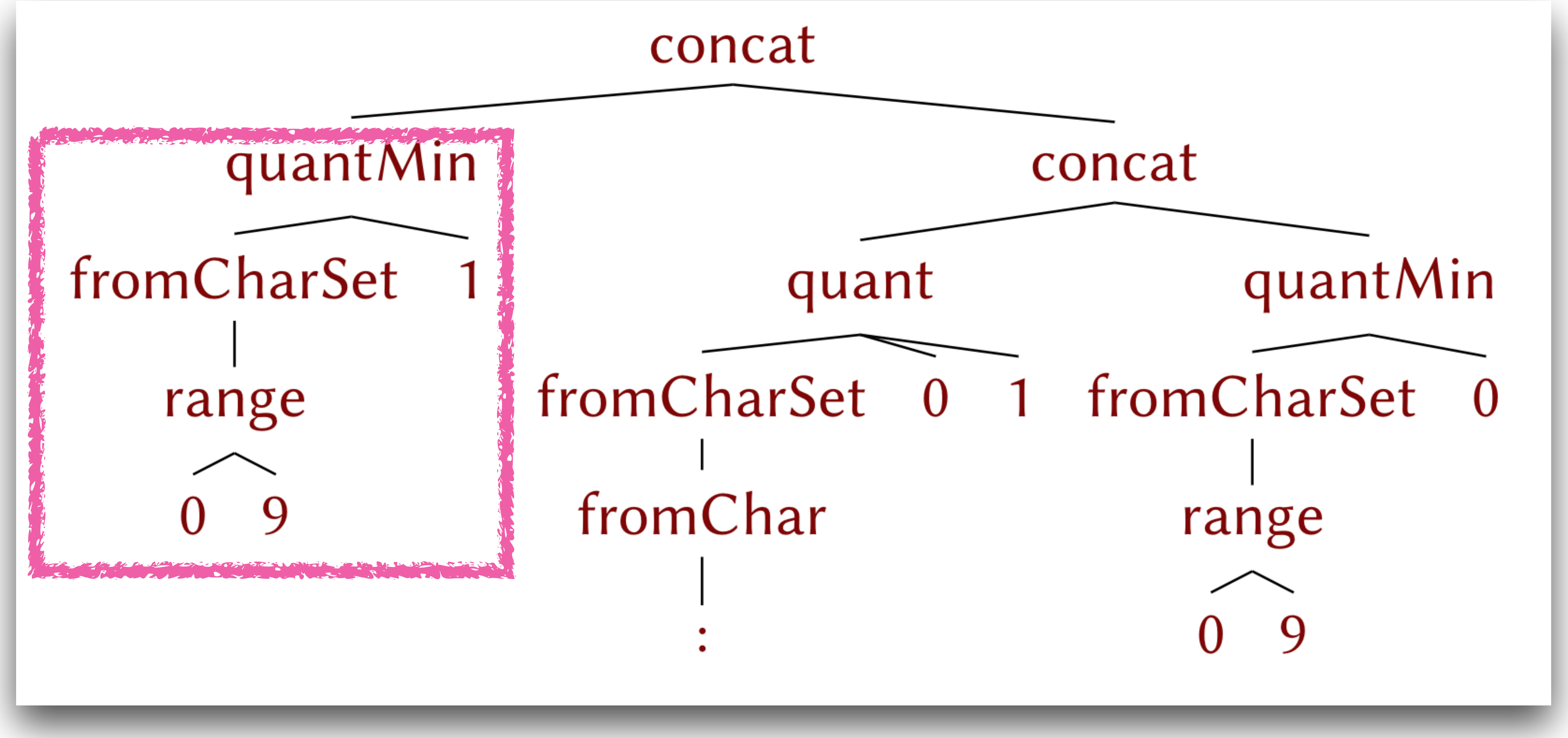
Operators

FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Domain of Regular Expressions (REGEX)
  - concise search patterns
  - terminals and operators

At least one digit,
followed by ‘:’ at most once,
followed by a digit at least zero times

[0–9]+:?[0–9]*

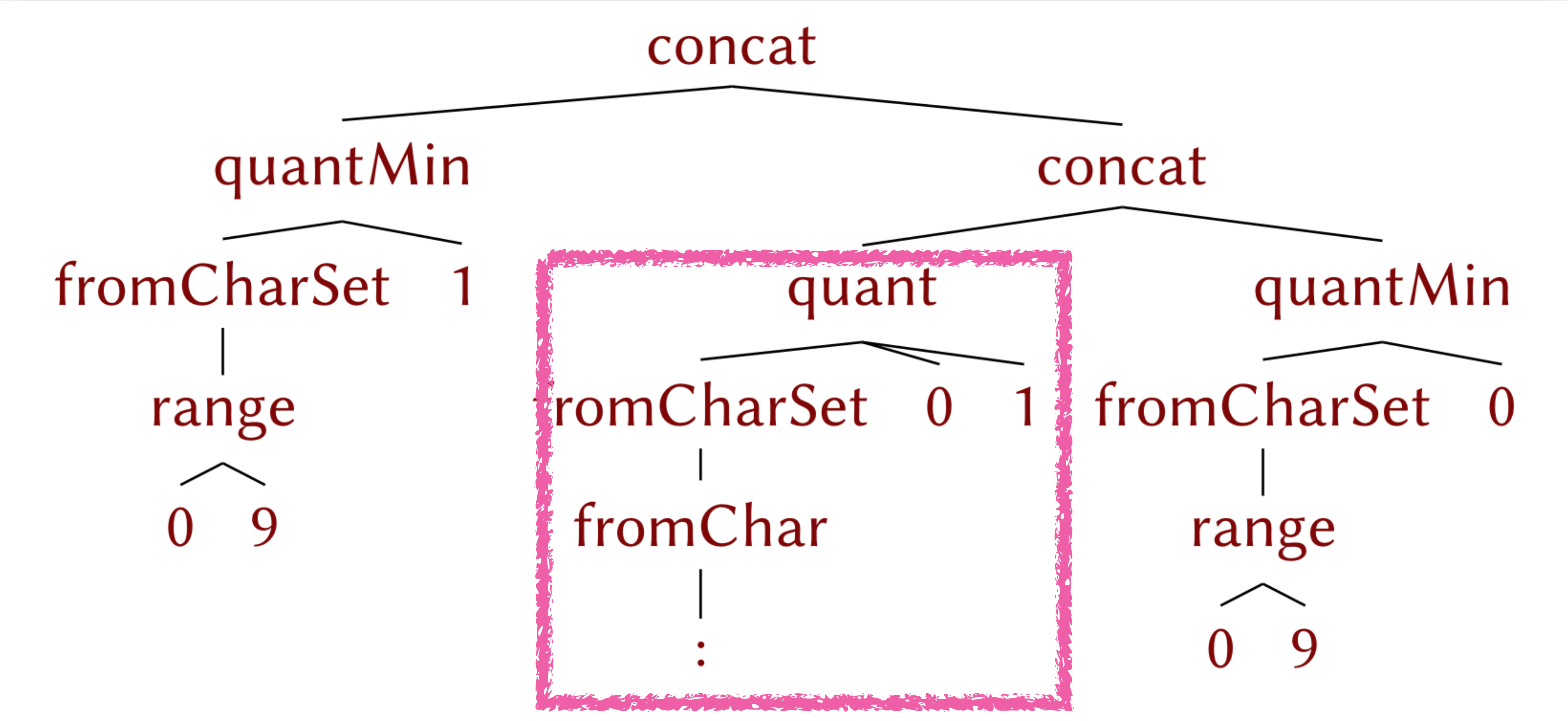


FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Domain of Regular Expressions (REGEX)
  - concise search patterns
  - terminals and operators

At least one digit,
followed by ‘:’ at most once,
followed by a digit at least zero times

[0–9]+:?[0–9]*

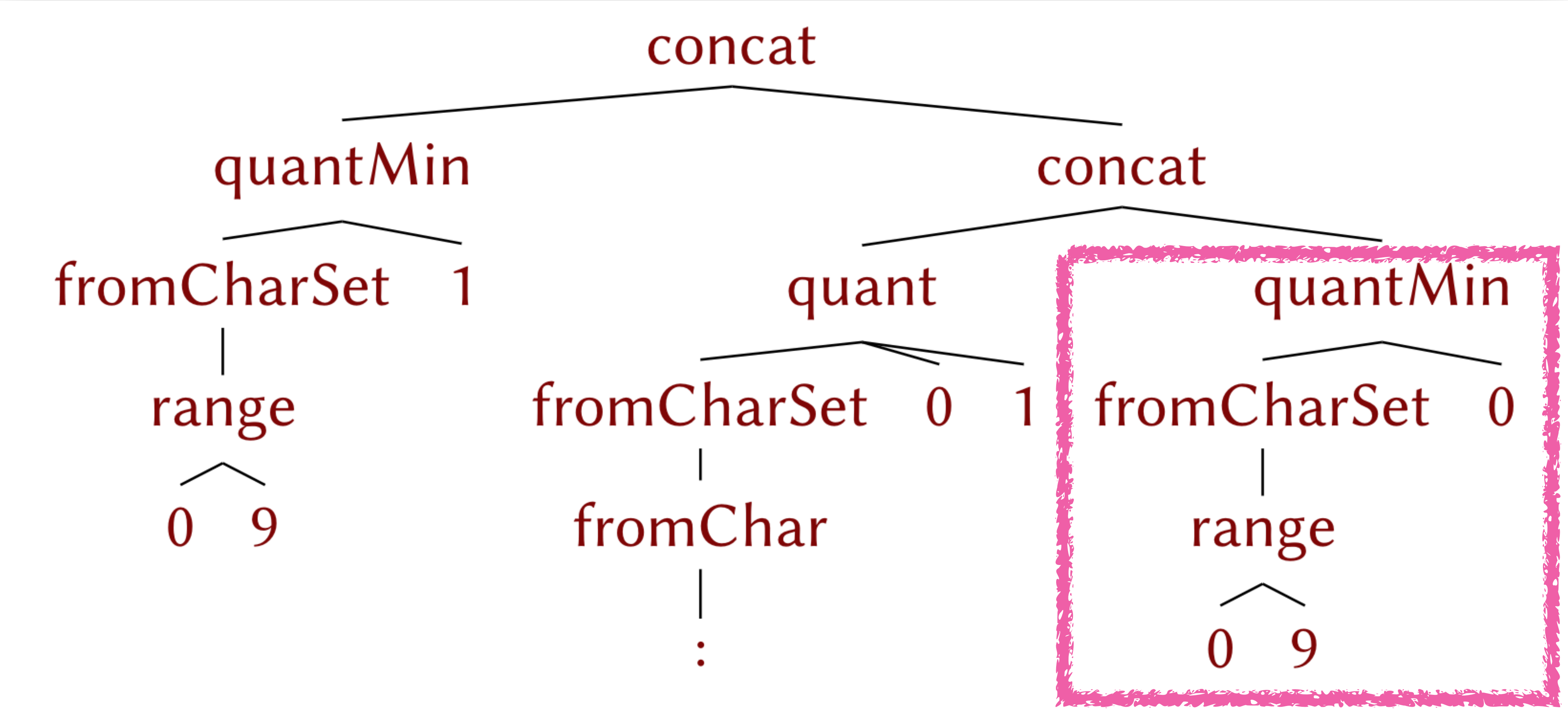


FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Domain of Regular Expressions (REGEX)
  - concise search patterns
  - terminals and operators

At least one digit,
followed by ‘:’ at most once,
followed by a digit at least zero times

[0–9]+:?[0–9]*



FIRST HAND EXPERIMENTS WITH (NL→CODE)

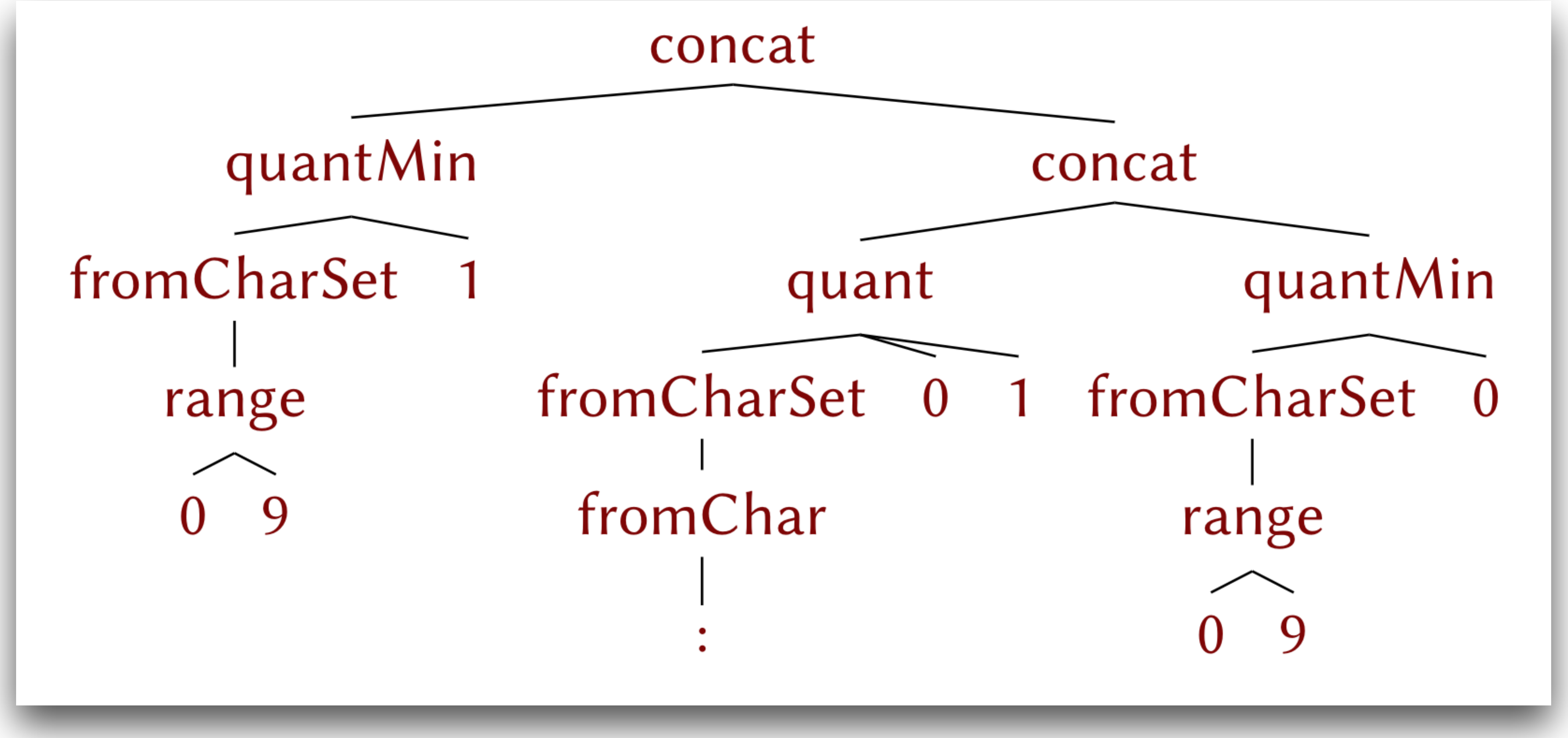
- Domain of Regular Expressions (REGEX)
  - concise search patterns
  - terminals and operators

At least one digit,
followed by ‘:’ at most once,
followed by a digit at least zero times

[0–9]+:?[0–9]*

✗ 12 , Abc

✓ 2345:6789 , 123



FIRST HAND EXPERIMENTS WITH (NL → CODE)

- Domain of Regular Expressions (REGEX)
  - concise search patterns
  - terminals and operators

At least one digit,
followed by ‘:’ at most once,
followed by a digit at least zero times

↓

GPT-3

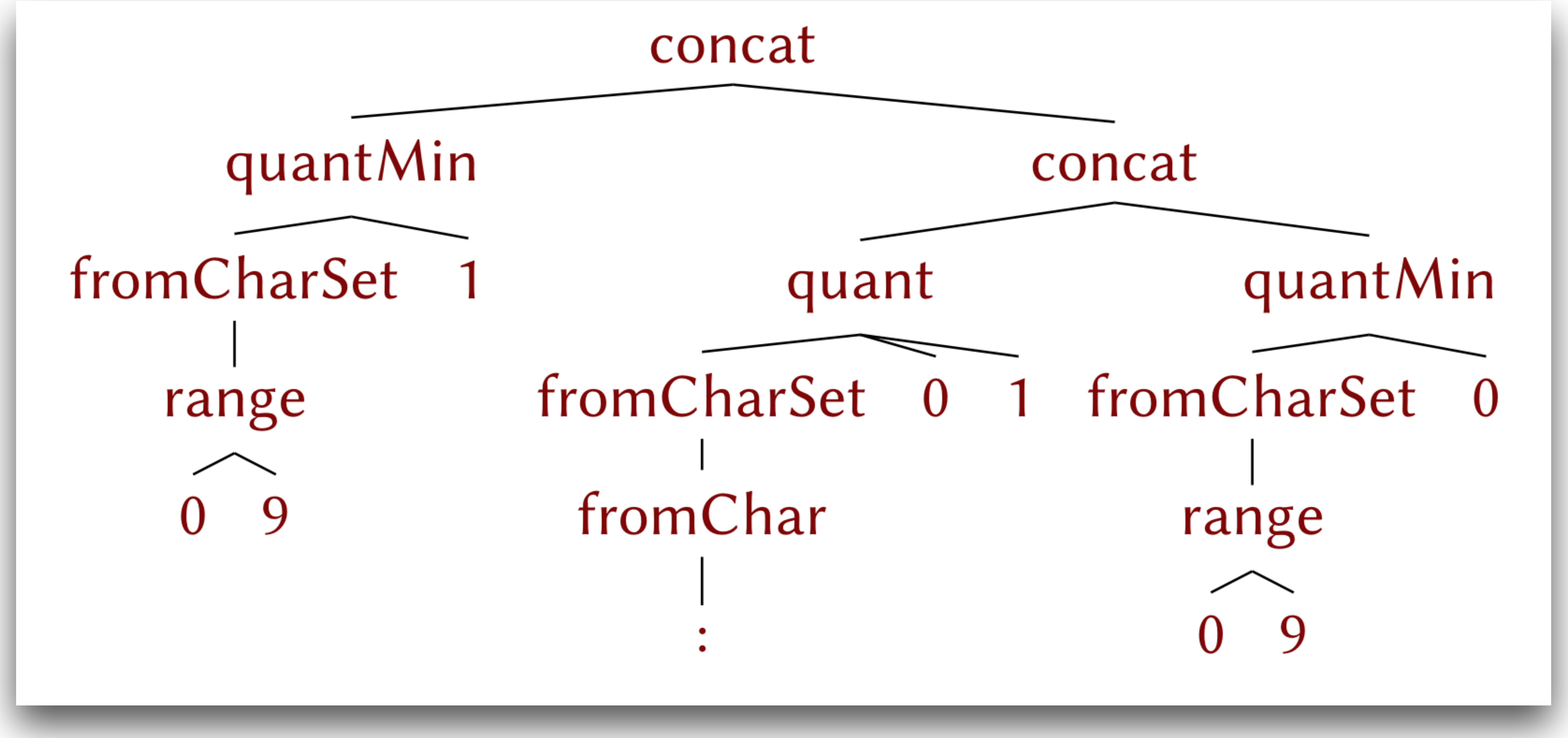
↓

([0–9]*.. :([0–9]*)?)+

[0–9]+:?[0–9]*

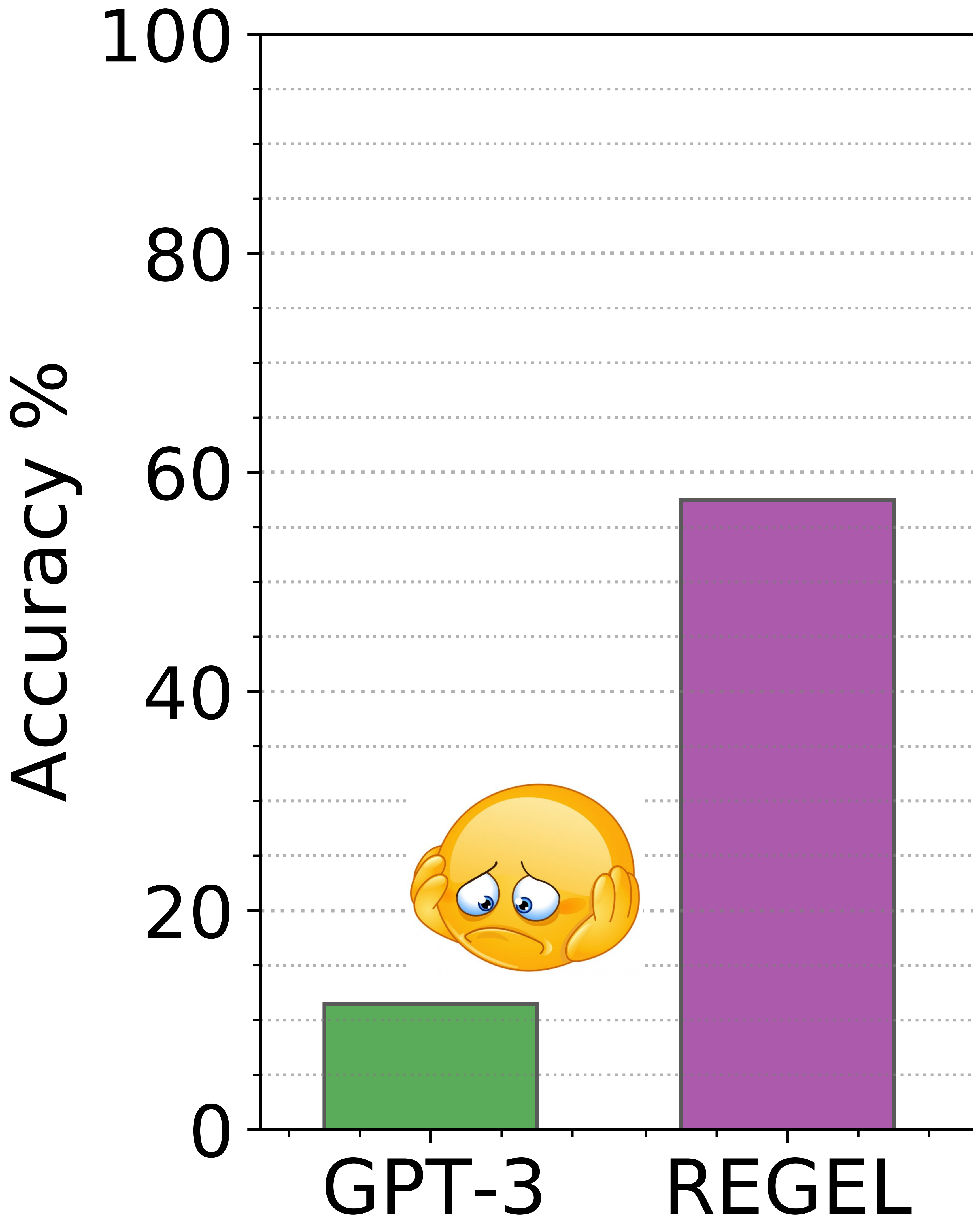
✗ 12 , Abc

✓ 2345:6789 , 123



FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Evaluated on 2 standard benchmark sets
- Less than **15%** overall success rate
- Compared to almost **60%** success rate of the state-of-the-art [2]



END OF THE STORY?

NOT THE END OF THE STORY!

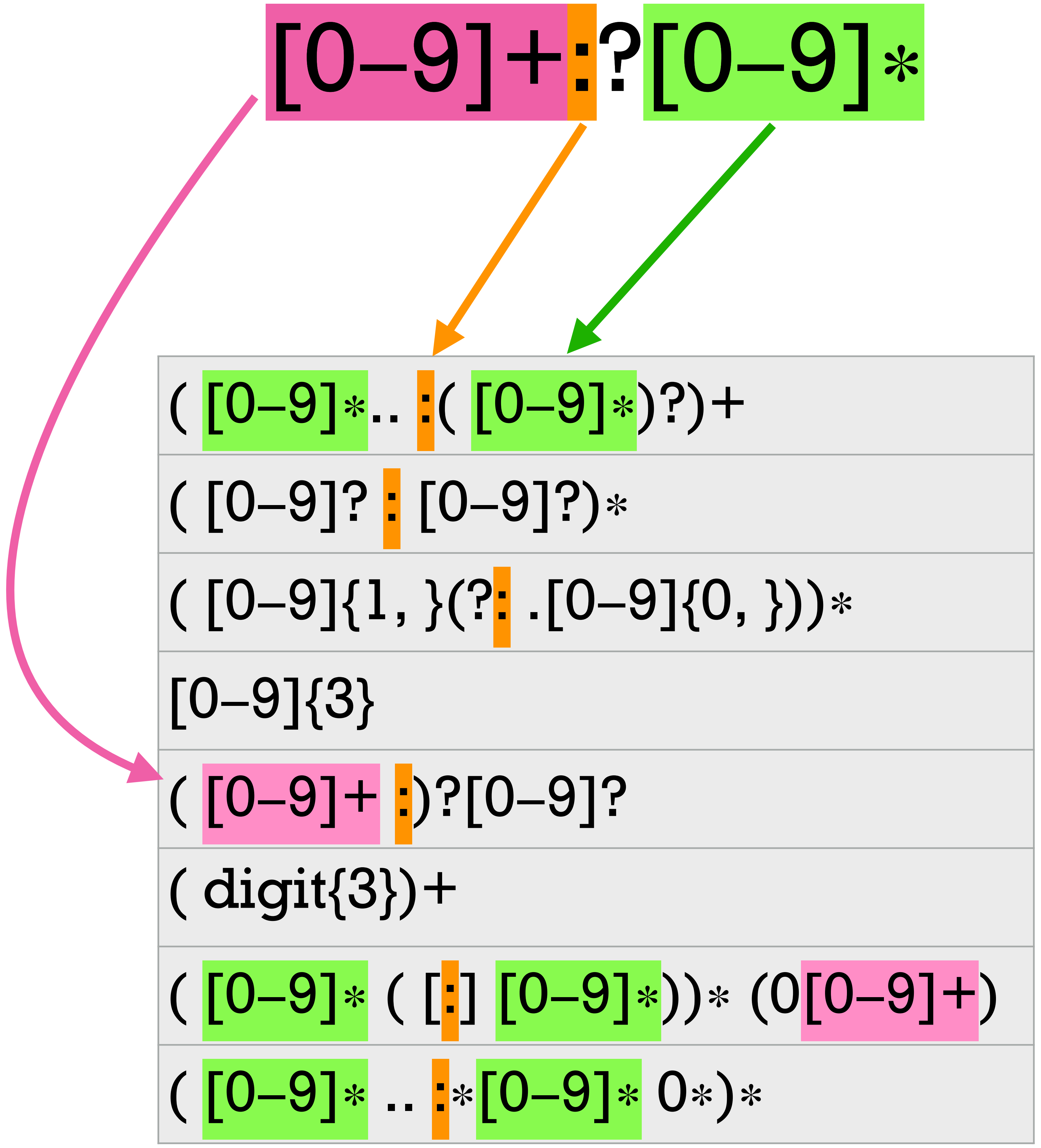
- Similarities between target and candidates:

[0–9]+:?[0–9]*

([0–9]*.. :([0–9]*)?)+
([0–9]? : [0–9]?)*
([0–9]{1, }(?: .[0–9]{0, }))*
[0–9]{3}
([0–9]+ :)?[0–9]?
(digit{3})+
([0–9]* ([:] [0–9]*))* (0[0–9]+)
([0–9]* .. :*[0–9]* 0*)*

NOT THE END OF THE STORY!

- Similarities between target and candidates:
  - Components of target are present



[0–9]+:?[0–9]*

([0–9]*.. :([0–9]*)?)+

([0–9]? : [0–9]?)*

([0–9]{1, }(?: .[0–9]{0, }))*

[0–9]{3}

([0–9]+ :)?[0–9]?

(digit{3})+

([0–9]* ([:] [0–9]*))* (0[0–9]+)

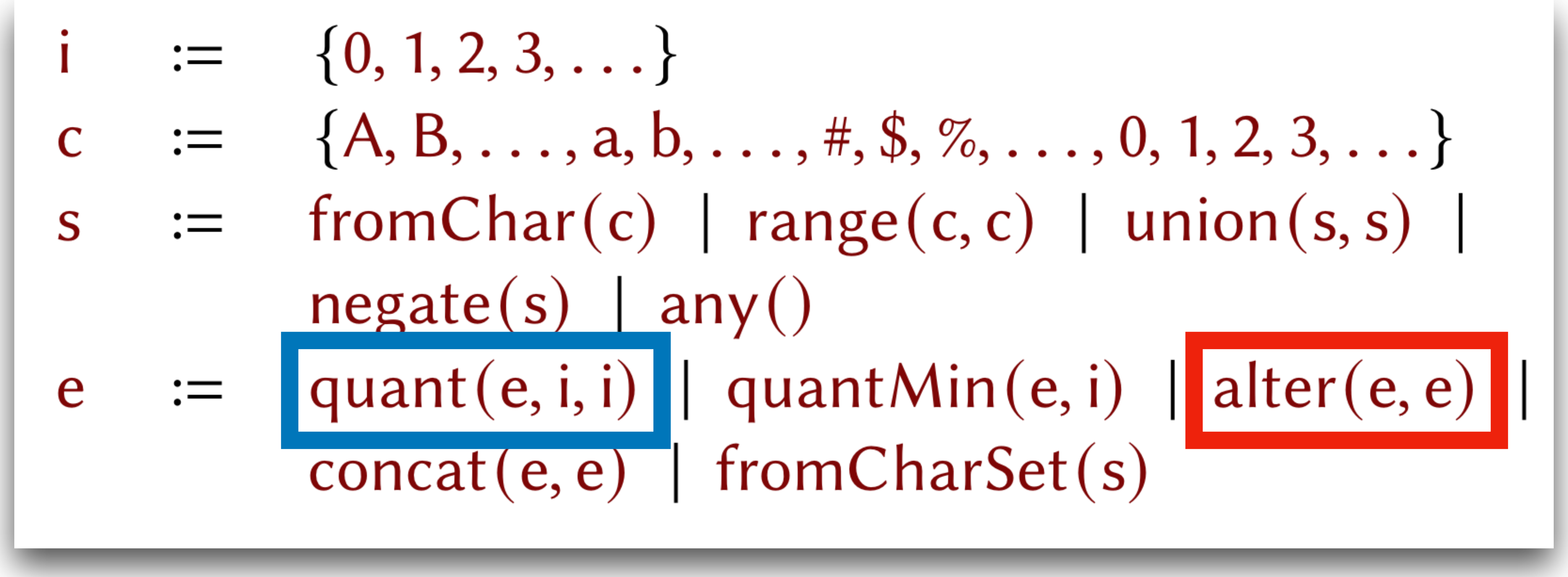
([0–9]* .. :*[0–9]* 0*)*

NOT THE END OF THE STORY!

- Similarities between target and candidates:
  - Components of target are present
  - Similar *shape* (operator types) to the target

[0–9]+ ? [0–9]∗

([0–9]∗.. :([0–9]∗ ?)+
([0–9] ? : [0–9] ? ∗
([0–9]{1, } ? .[0–9]{0, }))∗
[0–9]{3}
([0–9]+ : ? [0–9] ?
(digit{3})+
([0–9]∗ ([:] [0–9]∗))∗ (0[0–9]+)
([0–9]∗ .. :∗[0–9]∗ 0∗)∗



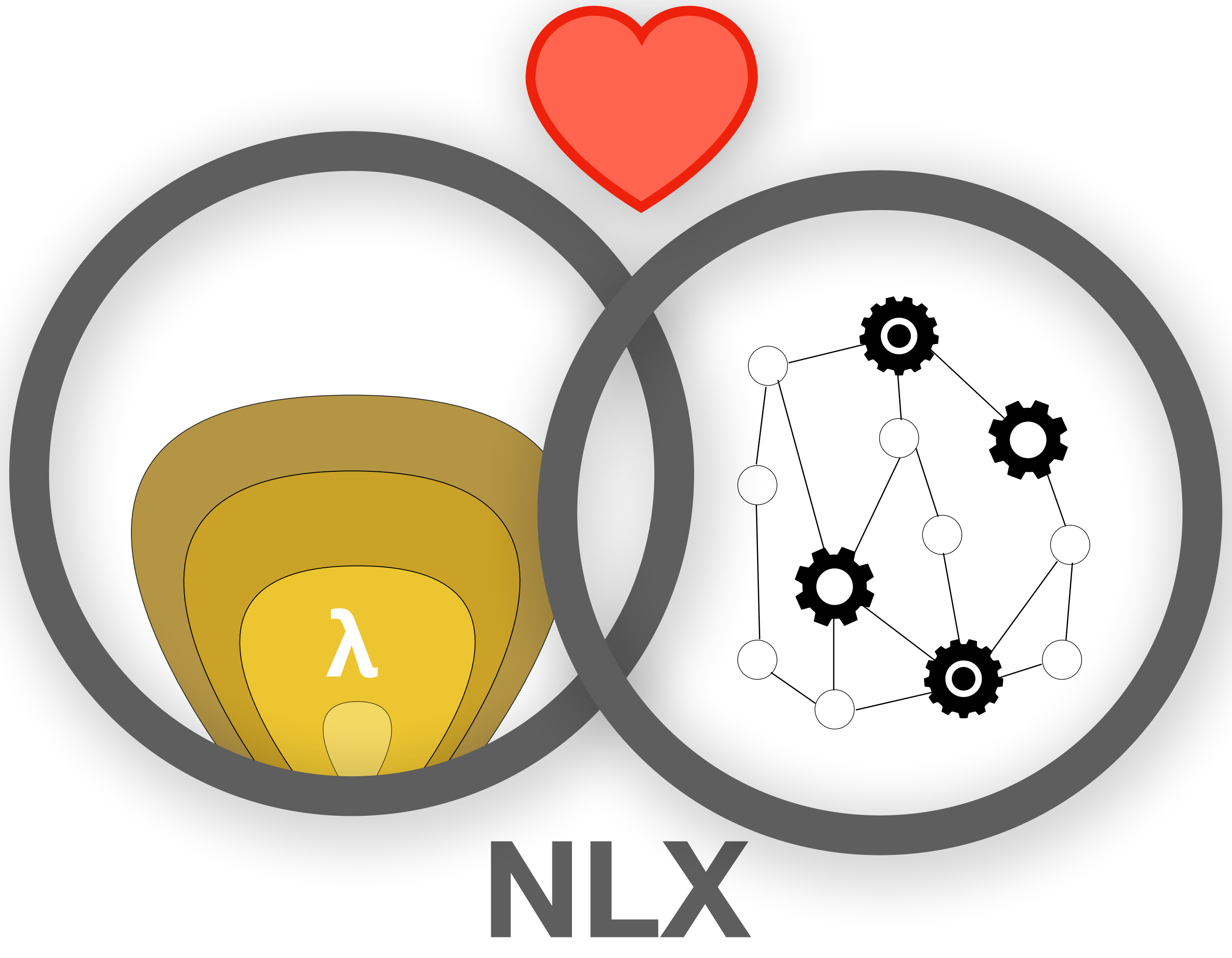
```
i  :=  {0, 1, 2, 3, . . .}
c  :=  {A, B, . . . , a, b, . . . , #, $, %, . . . , 0, 1, 2, 3, . . .}
s  :=  fromChar(c) | range(c, c) | union(s, s) |
       negate(s) | any()
e  :=  quant(e, i, i) | quantMin(e, i) | alter(e, e) |
       concat(e, e) | fromCharSet(s)
```

NLX PROGRAM SYNTHESIS FRAMEWORK

- Similarities between target and candidates:
  - Components of target are present
  - Similar *shape* (operator types) to the target
- NLX framework
  - Combine PTM with program synthesis

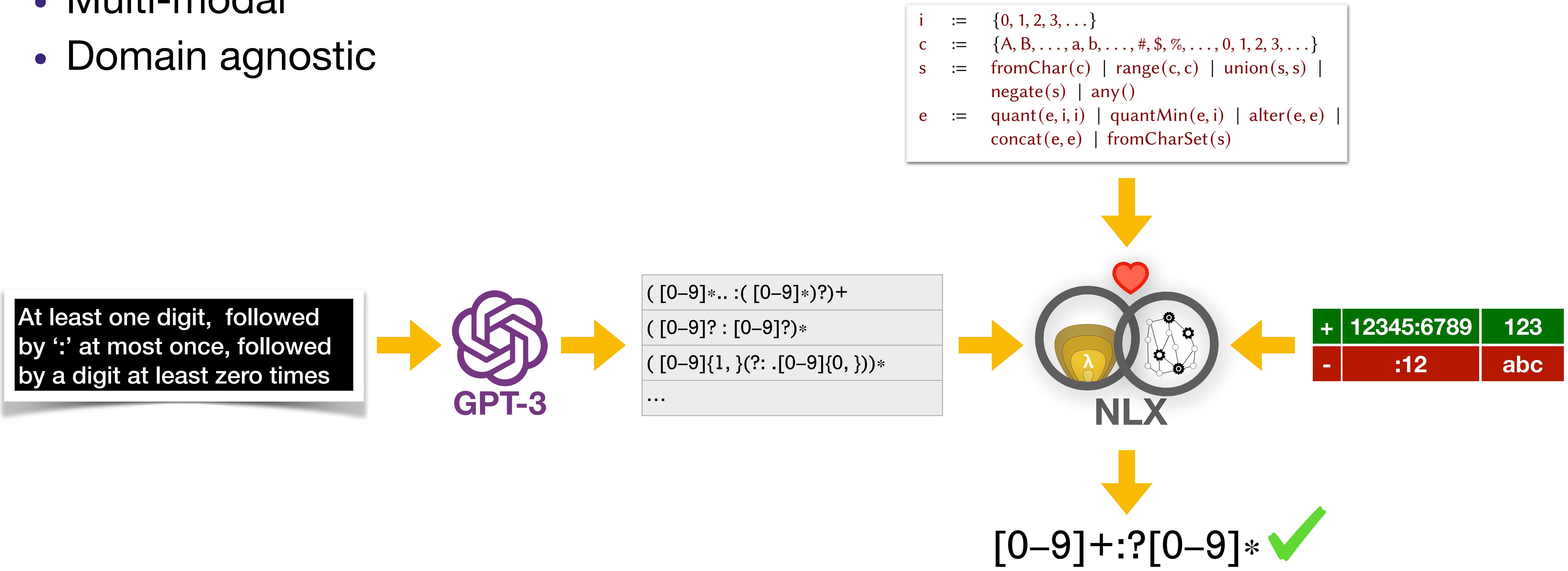
PTM ↓ Handle Ambiguous Natural Language

program synthesis ↓ Syntactically and Semantically Precise Code



NLX PROGRAM SYNTHESIS FRAMEWORK

- NLX framework
  - Multi-modal
  - Domain agnostic



```
i  :=  {0, 1, 2, 3, . . .}
c  :=  {A, B, . . . , a, b, . . . , #, $, %, . . . , 0, 1, 2, 3, . . .}
s  :=  fromChar(c) | range(c, c) | union(s, s) |
       negate(s) | any()
e  :=  quant(e, i, i) | quantMin(e, i) | alter(e, e) |
       concat(e, e) | fromCharSet(s)
```

At least one digit, followed by ':' at most once, followed by a digit at least zero times

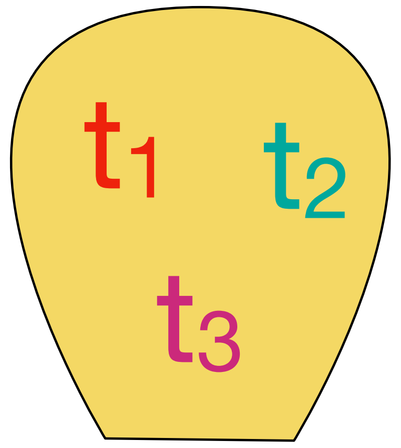
```
( [0–9]∗.. :( [0–9]∗)?)+
( [0–9]? : [0–9]?)∗
( [0–9]{1, }(?: .[0–9]{0, }))∗
...
```

+	12345:6789	123
-	:12	abc

[0–9]+:?[0–9]∗ ✓

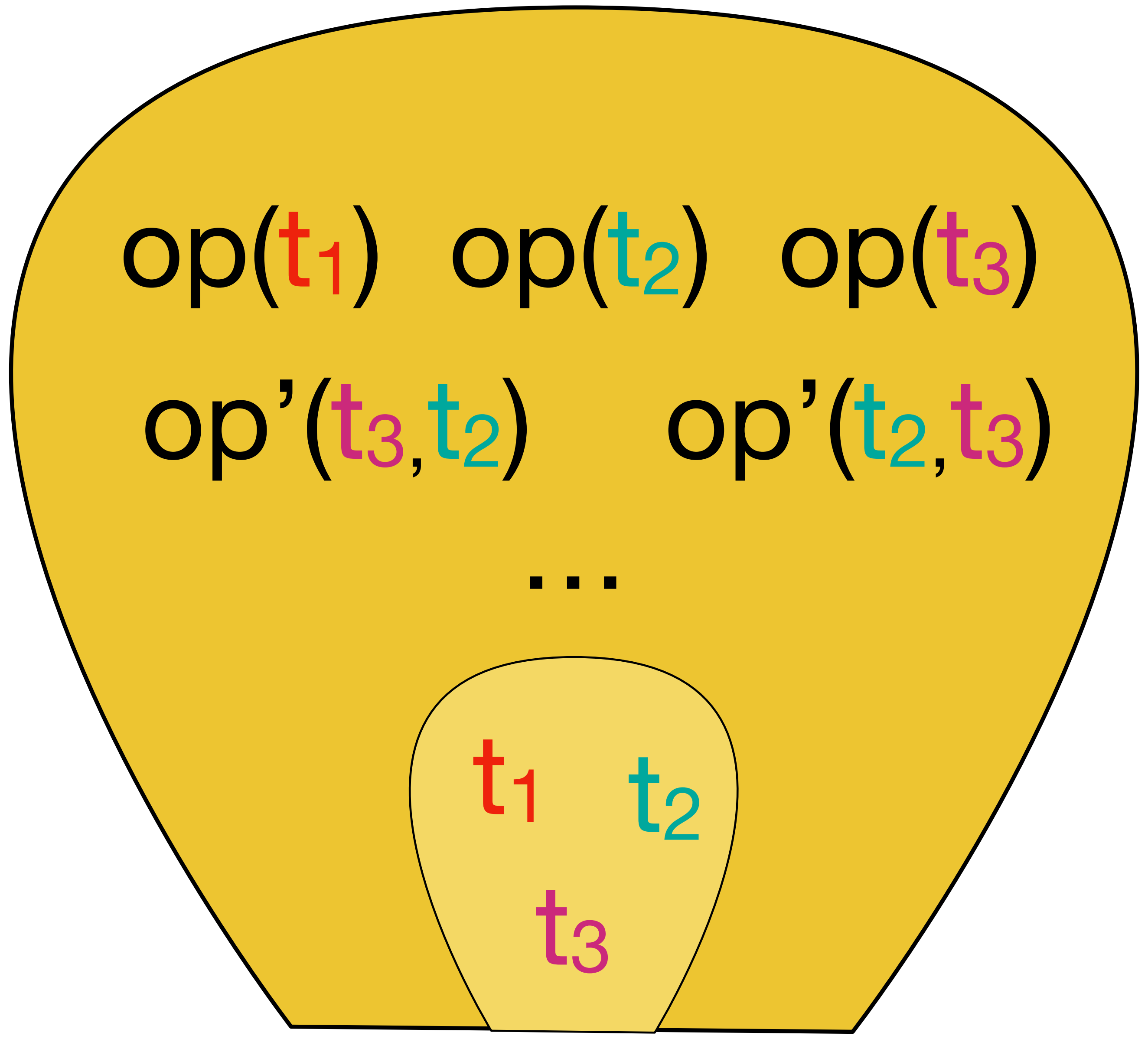
COMPONENT-BASED SYNTHESIS (CBS)

- Search based approach
  - Seed terms



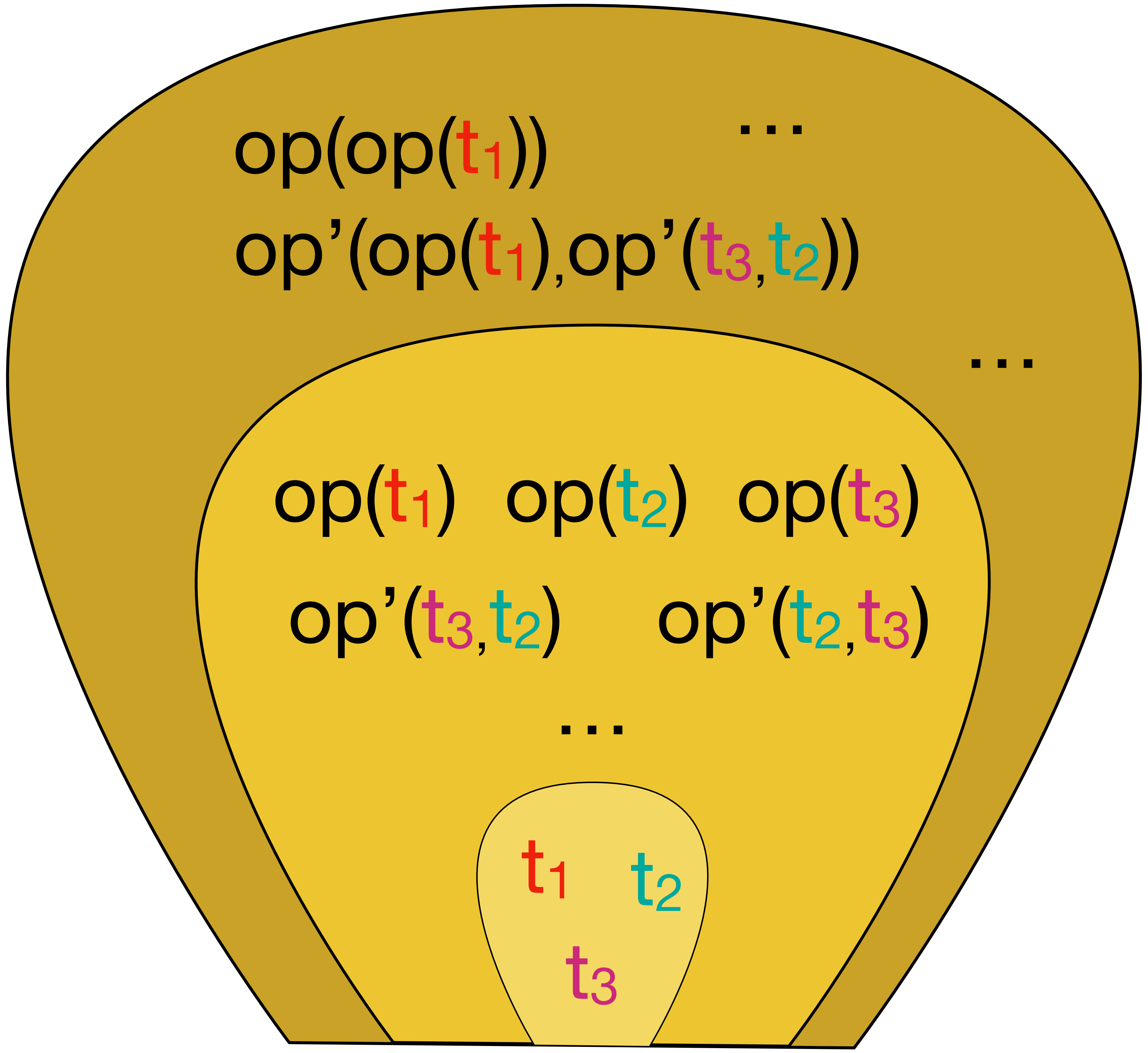
COMPONENT-BASED SYNTHESIS (CBS)

- Search based approach
  - Seed terms
  - Iterative expansion


op(t_1) op(t_2) op(t_3)
op'(t_3,t_2) op'(t_2,t_3)
...
t_1 t_2
t_3

COMPONENT-BASED SYNTHESIS (CBS)

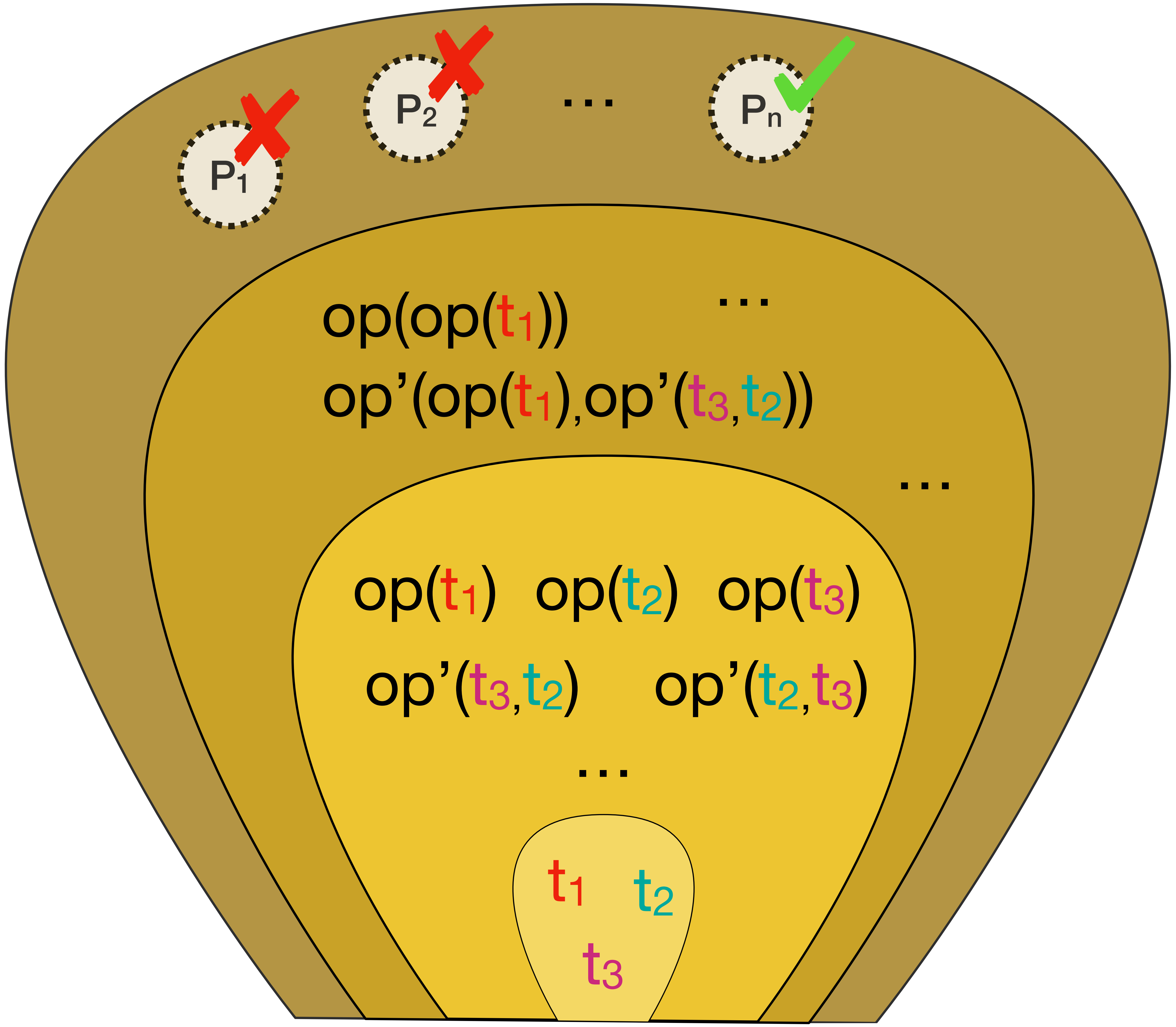
- Search based approach
  - Seed terms
  - Iterative expansion


op(op(t_1))
…
op'(op(t_1),op'(t_3,t_2))
…
op(t_1) op(t_2) op(t_3)
op'(t_3,t_2) op'(t_2,t_3)
…
t_1 t_2
t_3

CHICAGO SPLASH2021

COMPONENT-BASED SYNTHESIS (CBS)

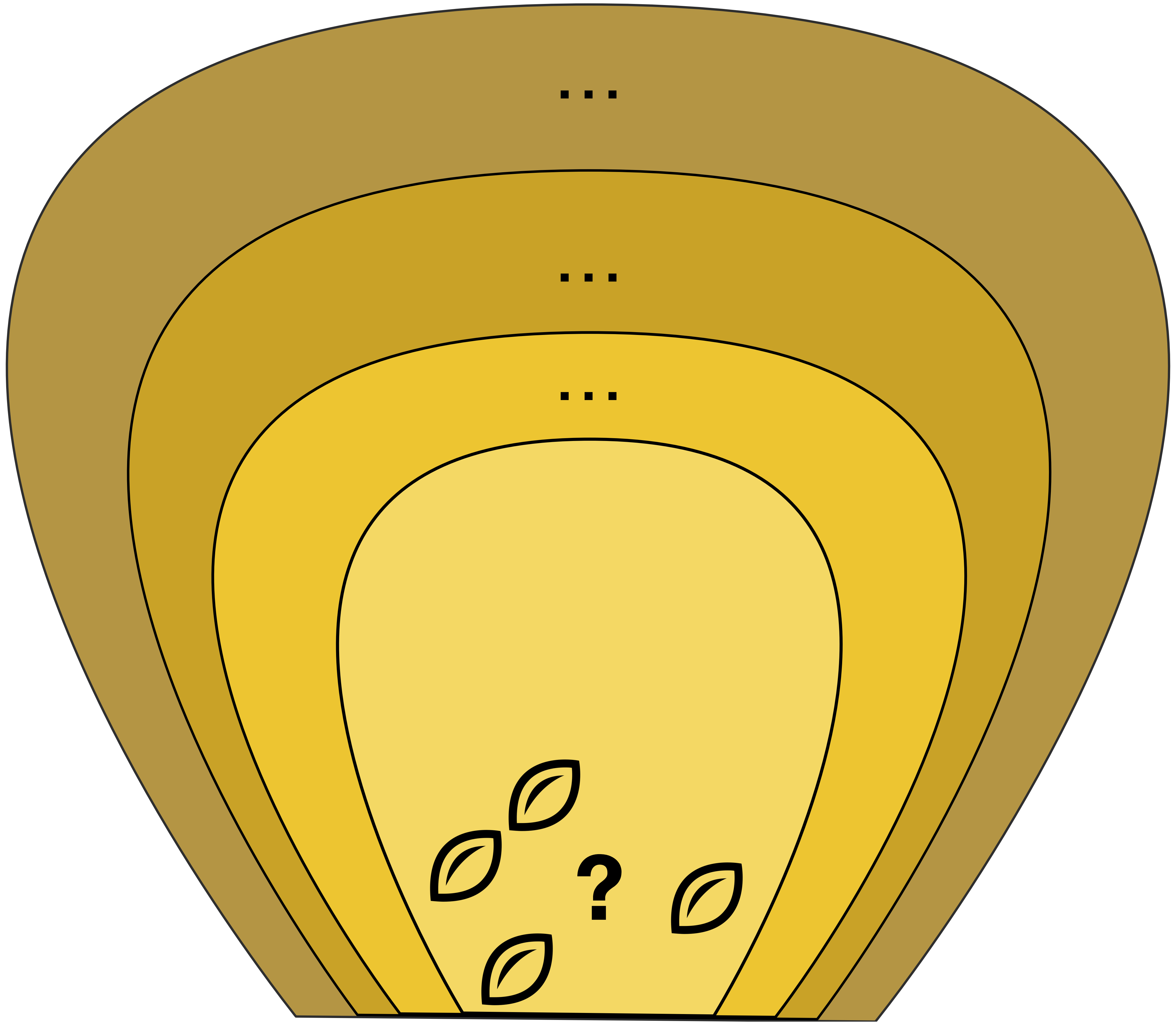
- Search based approach
  - Seed terms
  - Iterative expansion
  - Find consistent programs



P_1 ✗
P_2 ✗
…
P_n ✓
op(op(t_1))
…
op’(op(t_1),op’(t_3,t_2))
…
op(t_1) op(t_2) op(t_3)
op’(t_3,t_2) op’(t_2,t_3)
…
t_1 t_2
t_3

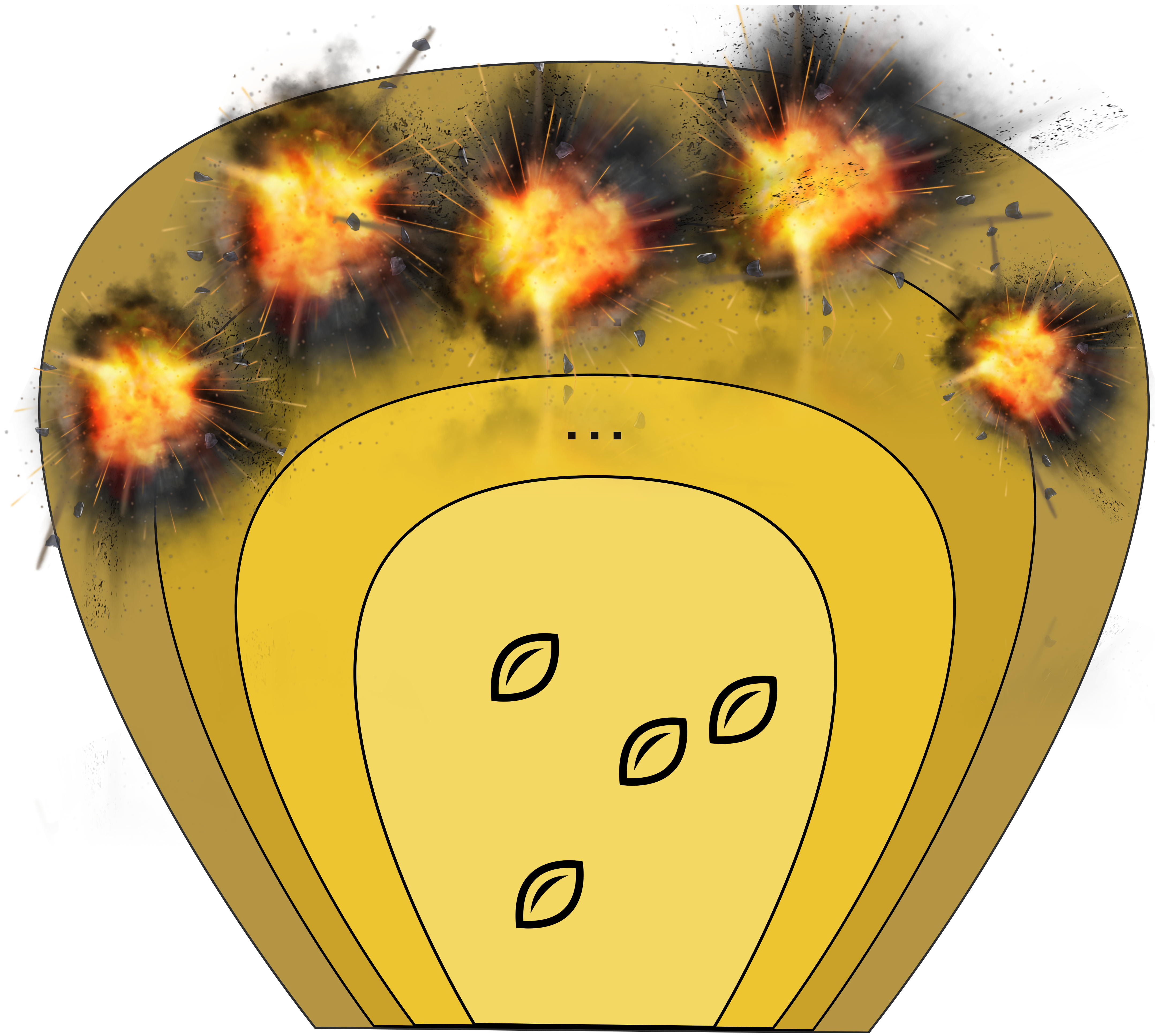
CHALLENGES WITH CBS

- Search based approach
  - Seed terms
  - Iterative expansion
  - Find consistent programs
- Challenges:
  - Useful + concise seeds



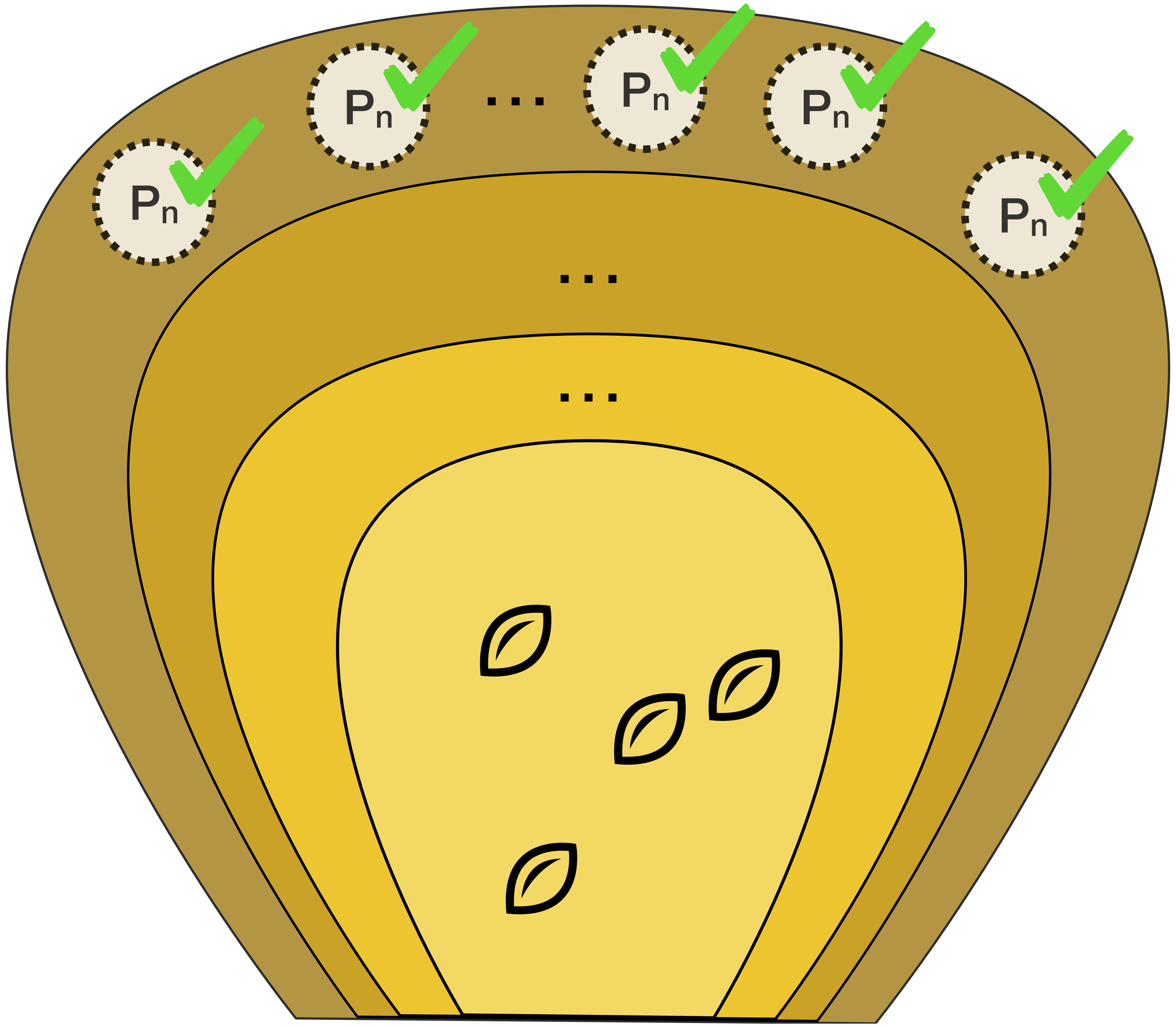
CHALLENGES WITH SEARCH

- Search based approach
  - Seed terms
  - Iterative expansion
  - Find consistent programs
- Challenges:
  - Useful + concise seeds
  - State-space explosion



CHALLENGES WITH SEARCH

- Search based approach
  - Seed terms
  - Iterative expansion
  - Find consistent programs
- Challenges:
  - Useful + concise seeds
  - State-space explosion
  - Final ranking



NLX Solution

SEED COMPONENTS

- Extract components from PTM's candidates

([0–9]*.. :([0–9]*)?)+

[0–9]*.. :([0–9]*)?
[0–9]*.. :
([0–9]*)?
[0–9]*..
[0–9]*.
[0–9]*
[0–9]
0
9
⋮

Can become prohibitively large!

SEED COMPONENTS

- Extract components from PTM's candidates
  - Eliminate *infrequent* components

([0–9]∗.. :([0–9]∗)?)+	([0–9]+ :)?[0–9]?
([0–9]? : [0–9]?)∗	([0–9]{3})+
([0–9]{1, }(?: .[0–9]{0, }))∗	([0–9]∗ ([:] [0–9]∗))∗ (0[0–9]+)
(digit){3}	([0–9]∗ .. :∗[0–9]∗ 0∗)∗

SEED COMPONENTS

- Extract components from PTM's candidates
    - Eliminate *infrequent* components

([0–9]*.. :([0–9]*)?)+	([0–9]+ :)?[0–9]?
([0–9]? : [0–9]?)*	([0–9]{3})+
([0–9]{1, }(?: .[0–9]{0, }))*	([0–9]* ([:] [0–9]*))* (0[0–9]+)
~~(digit){3}~~	([0–9]* .. :*[0–9]* 0*)*

SEED COMPONENTS

- Extract components from PTM’s candidates
  - Eliminate *infrequent* components
  - Eliminate *redundant* components

([0–9]∗.. :([0–9]∗)?)+

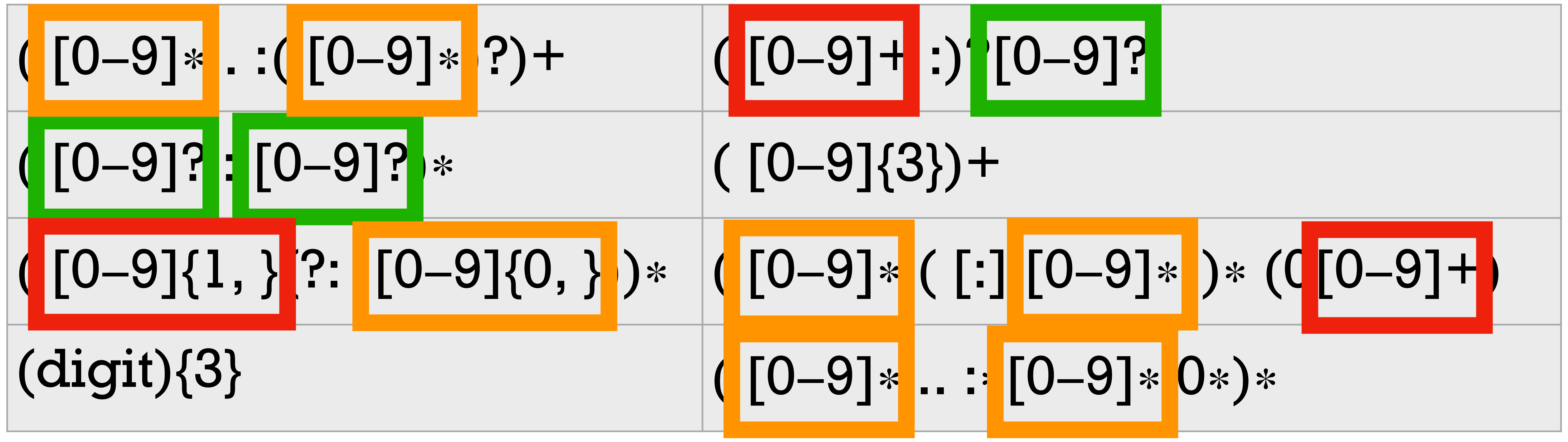
⬇

[0–9]∗.. :([0–9]∗)?
[0–9]∗.. :
([0–9]∗)?
[0–9]∗..
[0–9]∗.
[0–9]∗
[0–9]
0
9
:
.

([0–9]∗.. :([0–9]∗)?)+	([0–9]+ :)?[0–9]?
([0–9]? : [0–9]?)∗	([0–9]{3})+
([0–9]{1, }(?: .[0–9]{0, }))∗	([0–9]∗ ([:] [0–9]∗))∗ (0[0–9]+)
(digit){3}	([0–9]∗ .. :∗[0–9]∗ 0∗)∗

SEED COMPONENTS

- Extract components from PTM’s candidates
  - Eliminate *infrequent* components
  - Eliminate *redundant* components
    - **Non-Maximal component: 0, 9**
    - **Maximal component: [0-9]**



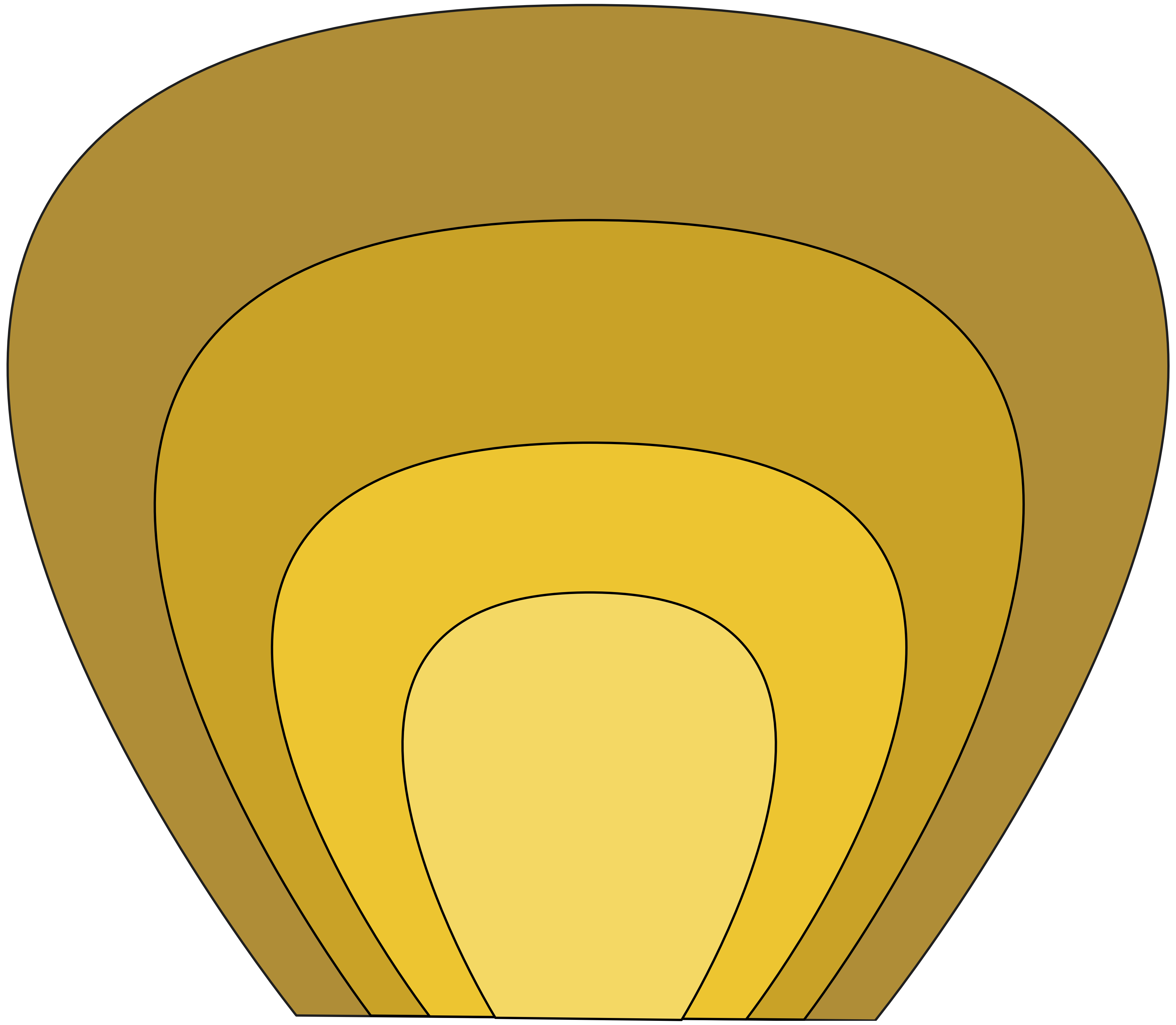
([0–9]∗ . :([0–9]∗ ?)+	([0–9]+ :)? [0–9]?
([0–9]? : [0–9]?)∗	([0–9]{3})+
([0–9]{1, } (?: [0–9]{0, })∗	([0–9]∗ ([:] [0–9]∗)∗ (0 [0–9]+)
(digit){3}	([0–9]∗ .. :∗ [0–9]∗ 0∗)∗

([0–9]∗.. :([0–9]∗)?)+

[0–9]∗.. :([0–9]∗)?
[0–9]∗.. :
([0–9]∗)?
[0–9]∗..
[0–9]∗.
[0–9]∗
✓ [0–9]
✗ 0
✗ 9
⋮

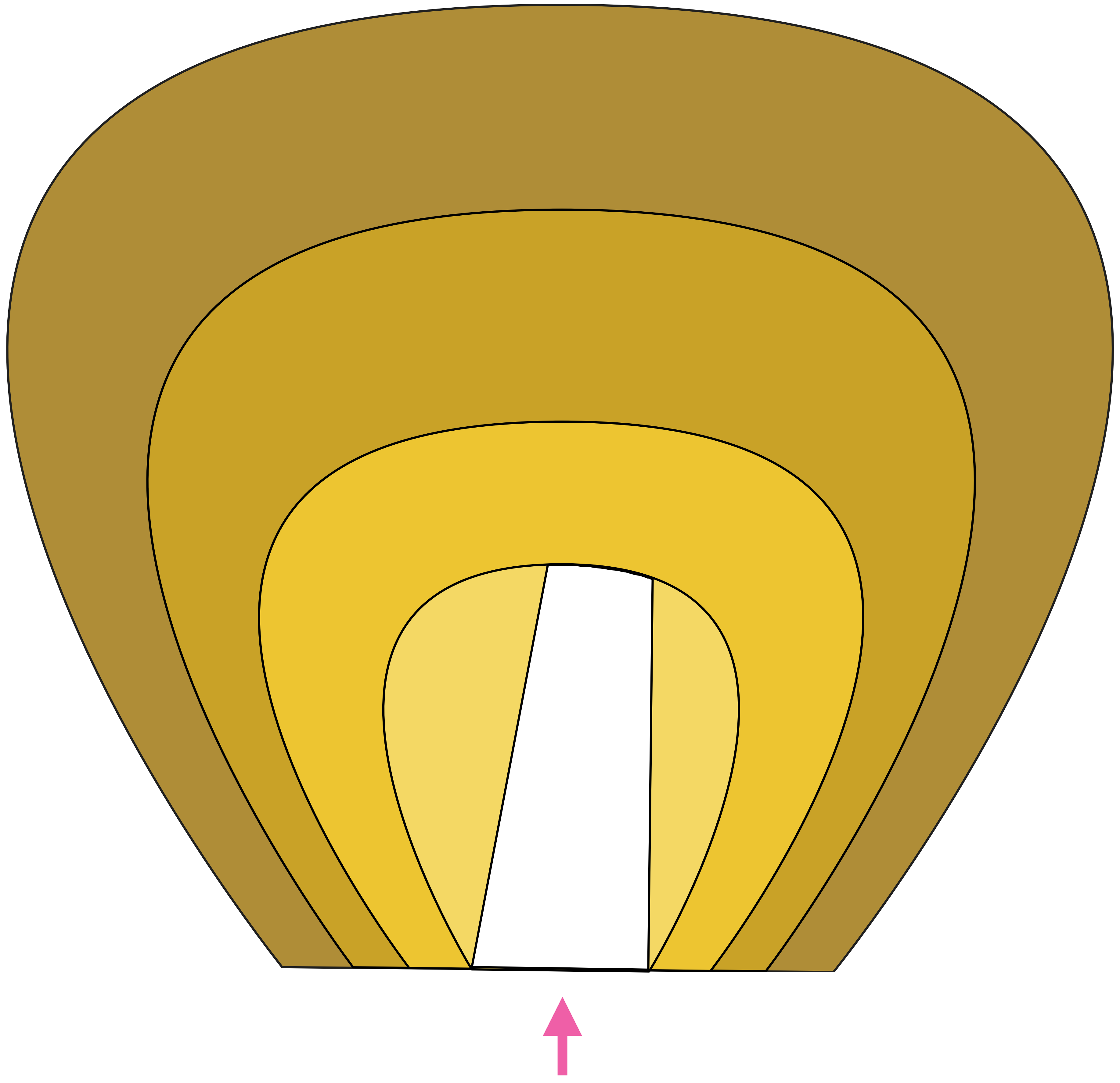
ITERATIVE EXPANSION

- Beam search



ITERATIVE EXPANSION

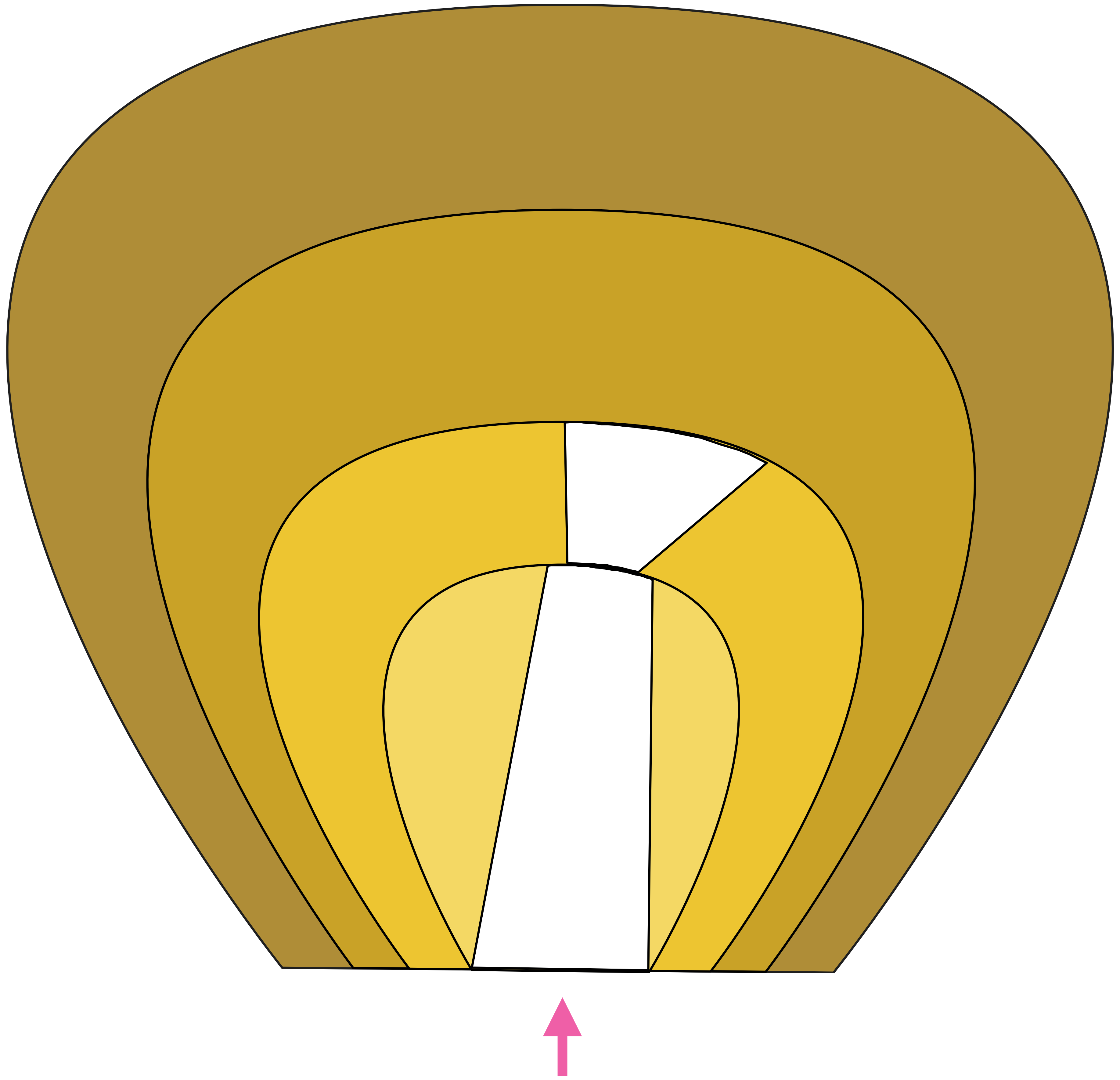
- Beam search



Only a subset of terms are considered

ITERATIVE EXPANSION

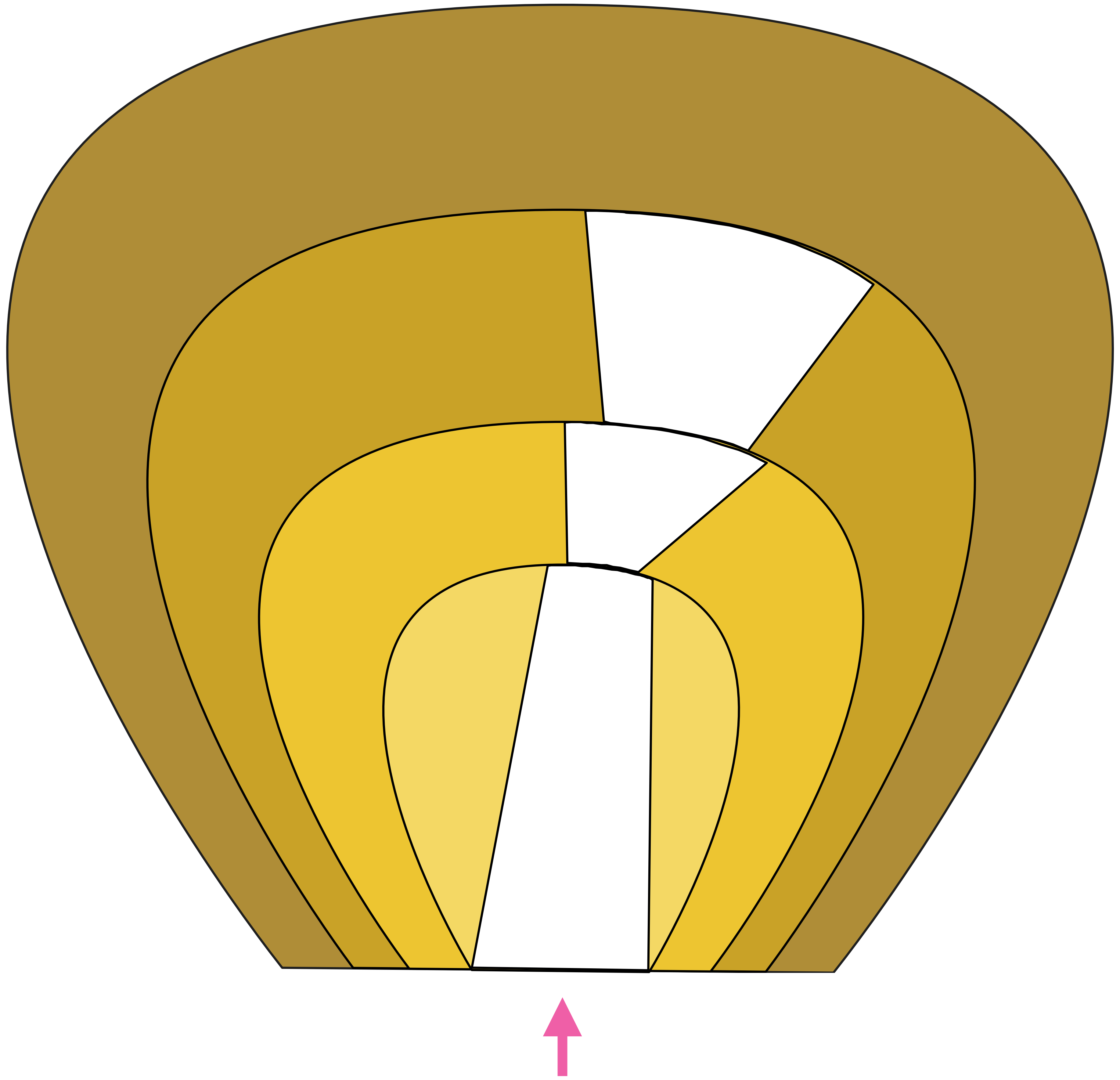
- Beam search



Only a subset of terms are considered

ITERATIVE EXPANSION

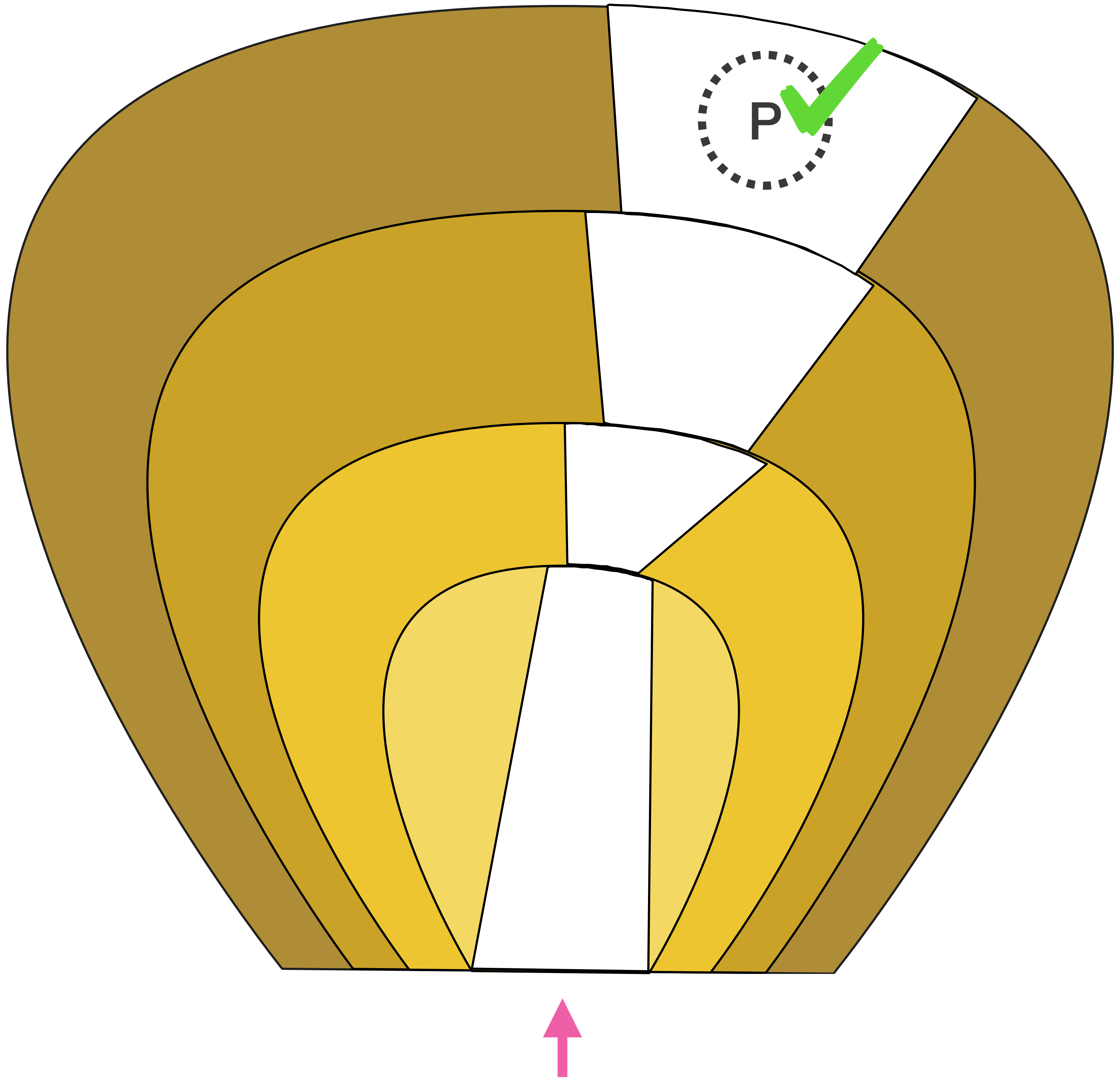
- Beam search



Only a subset of terms are considered

ITERATIVE EXPANSION

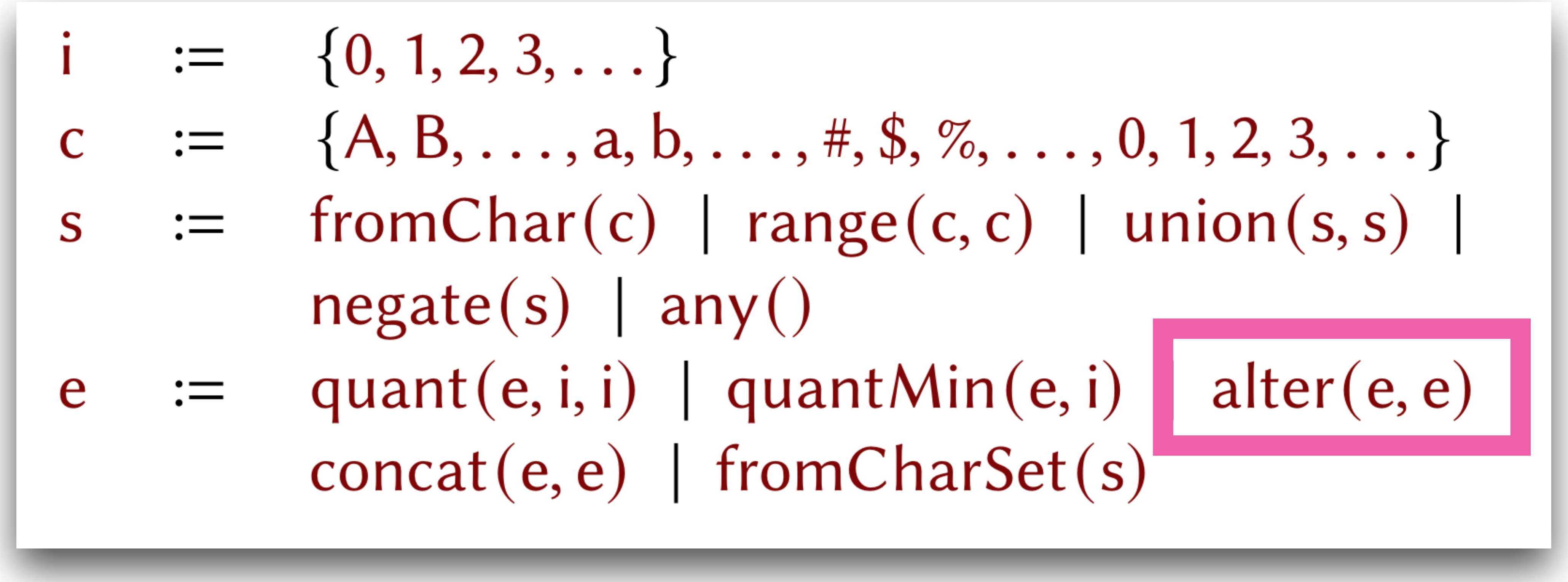
- Beam search



Only a subset of terms are considered

ITERATIVE EXPANSION

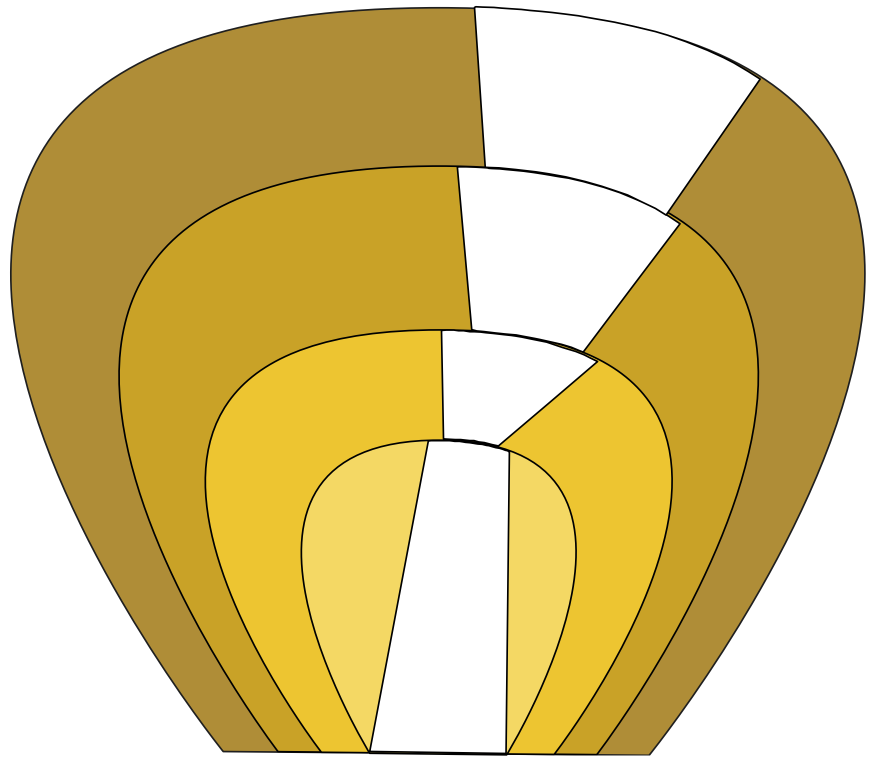
- Beam search
  - Bias the search w.r.t. operator distribution
  - Eliminate low-frequency operators

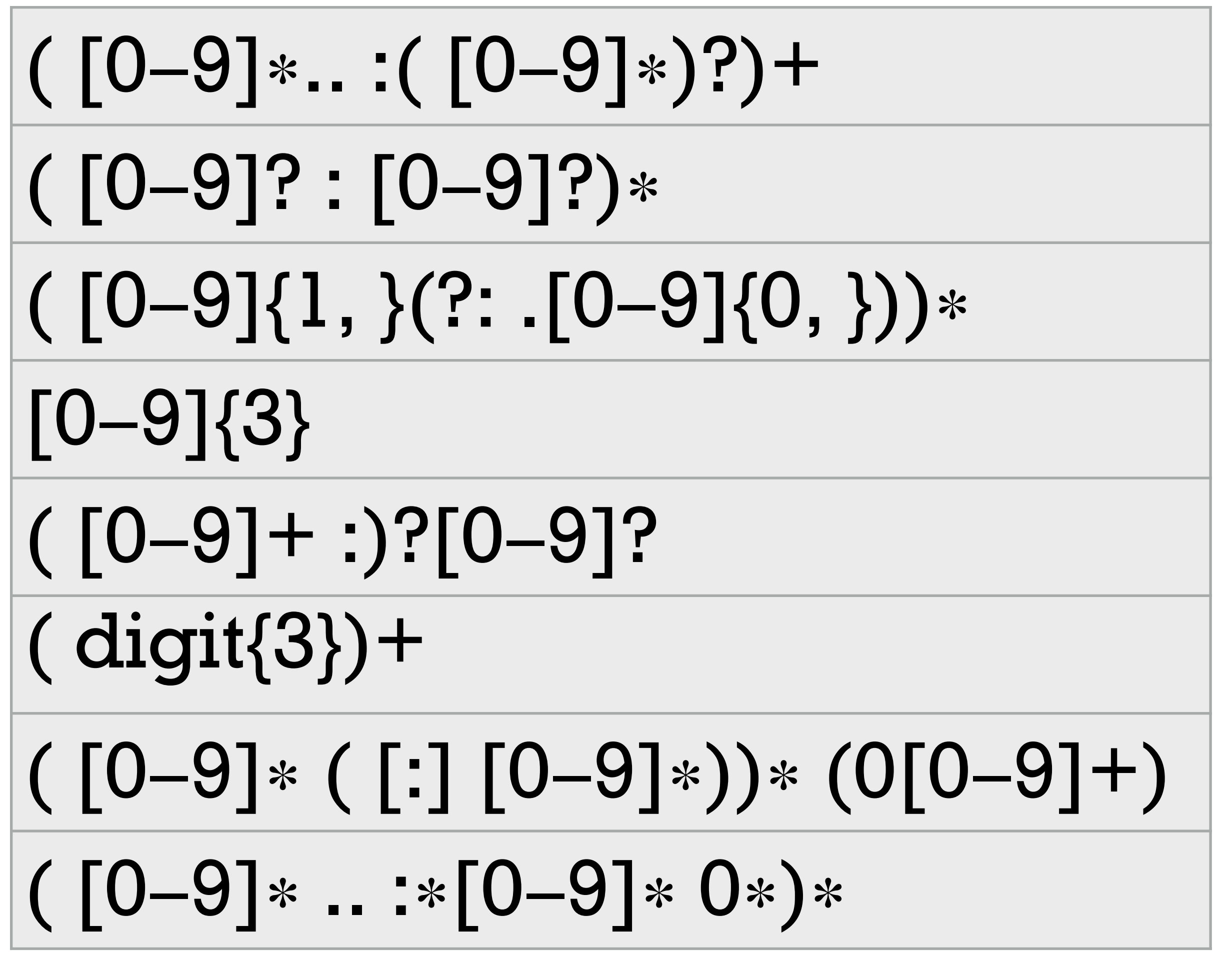


```
i  :=  {0, 1, 2, 3, . . .}
c  :=  {A, B, . . . , a, b, . . . , #, $, %, . . . , 0, 1, 2, 3, . . .}
s  :=  fromChar(c) | range(c, c) | union(s, s) |
       negate(s) | any()
e  :=  quant(e, i, i) | quantMin(e, i) | alter(e, e)
       concat(e, e) | fromCharSet(s)
```

Alter operator is NOT used

No need to apply *Alter* at expansions

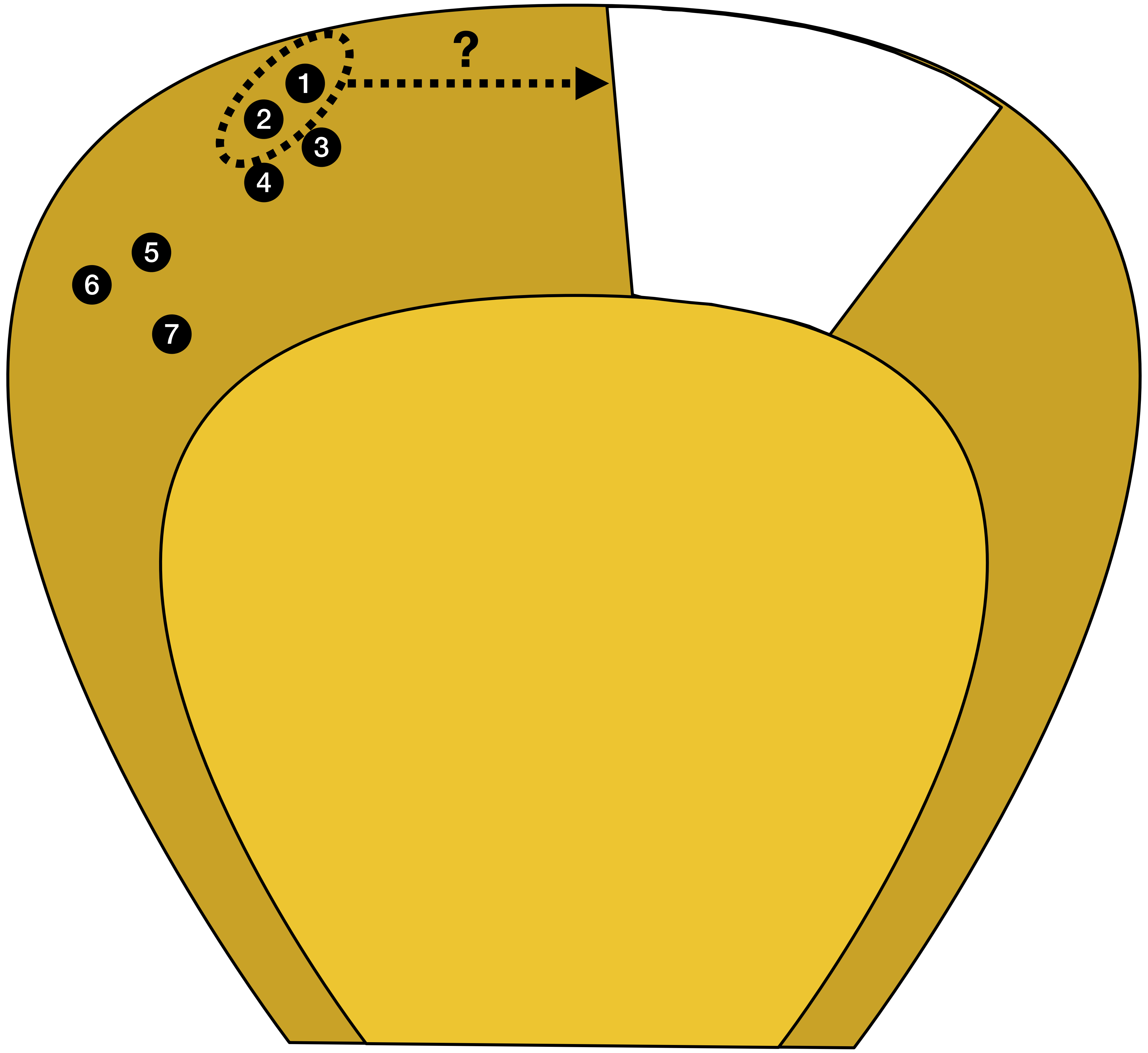




([0–9]*.. :([0–9]*)?)+
([0–9]? : [0–9]?)*
([0–9]{1, }(?: .[0–9]{0, }))*
[0–9]{3}
([0–9]+ :)?[0–9]?
(digit{3})+
([0–9]* ([:] [0–9]*))* (0[0–9]+)
([0–9]* .. :*[0–9]* 0*)*

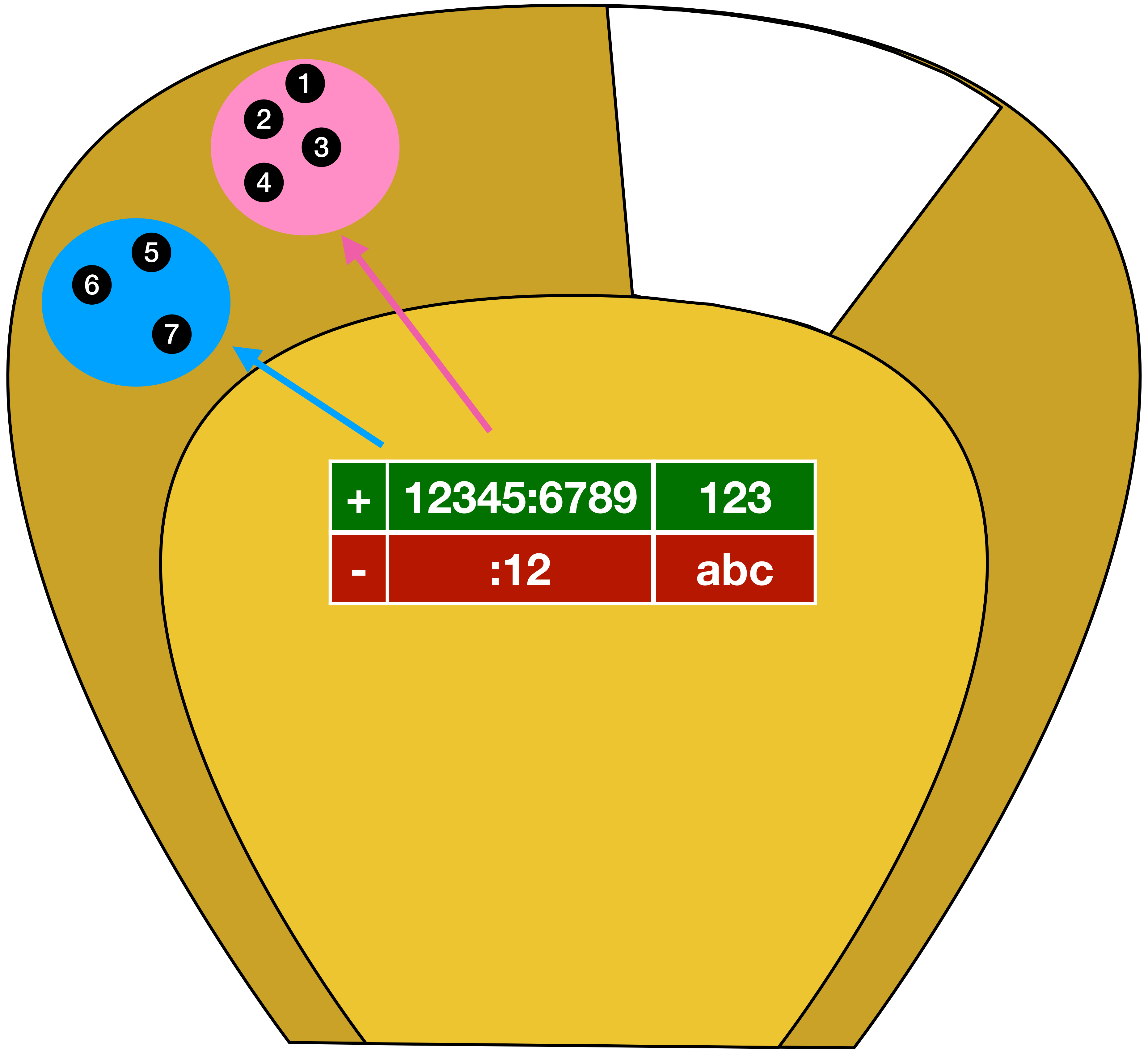
ITERATIVE EXPANSION

- Beam search
  - Bias the search w.r.t. operator distribution
  - Eliminate low-frequency operators
- How to define the beam?



ITERATIVE EXPANSION

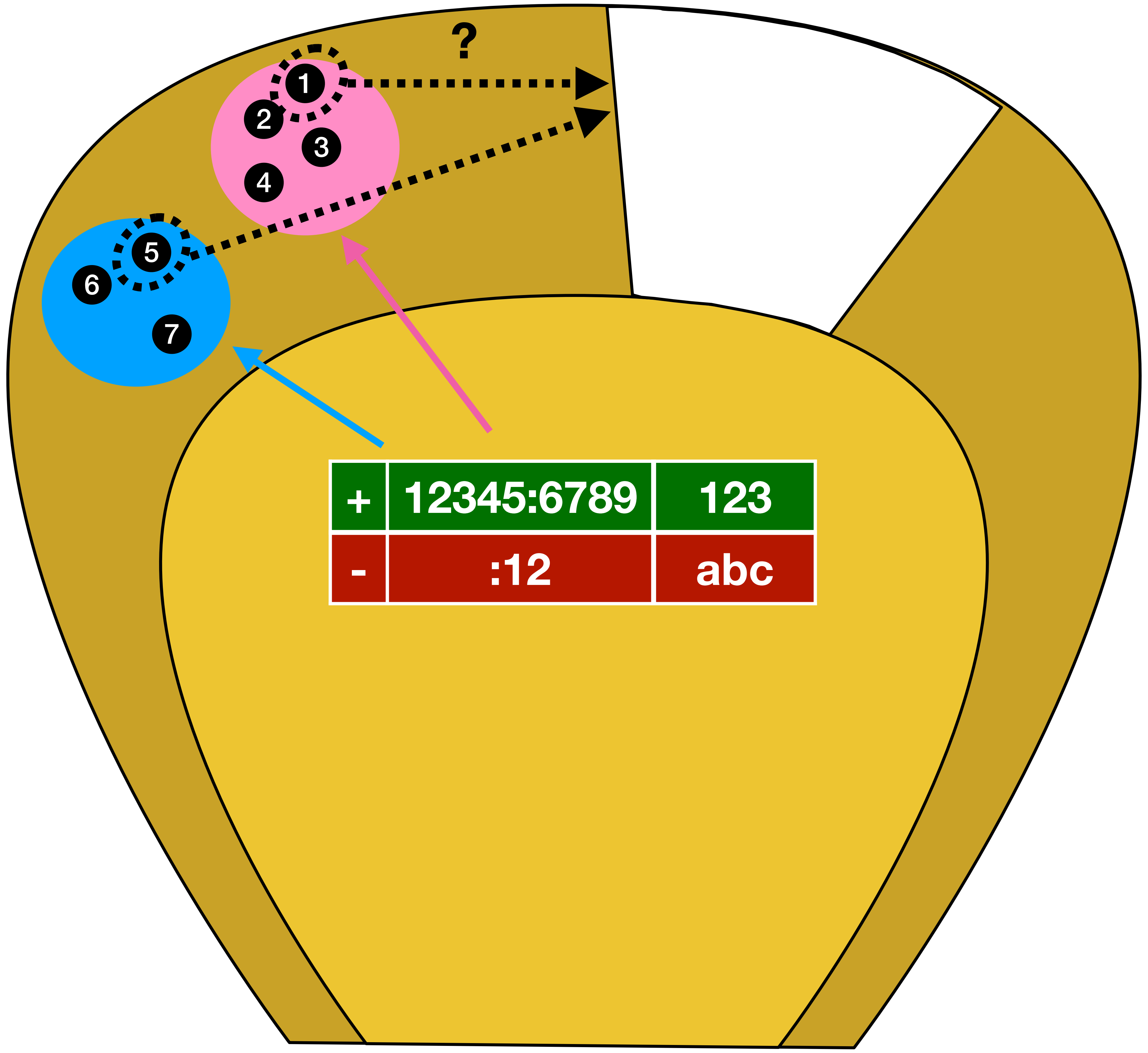
- Beam search
  - Bias the search w.r.t. operator distribution
  - Eliminate low-frequency operators
- How to define the beam?
- *Semantic condensation*
  - Classify candidates using examples



+	12345:6789	123
-	:12	abc

ITERATIVE EXPANSION

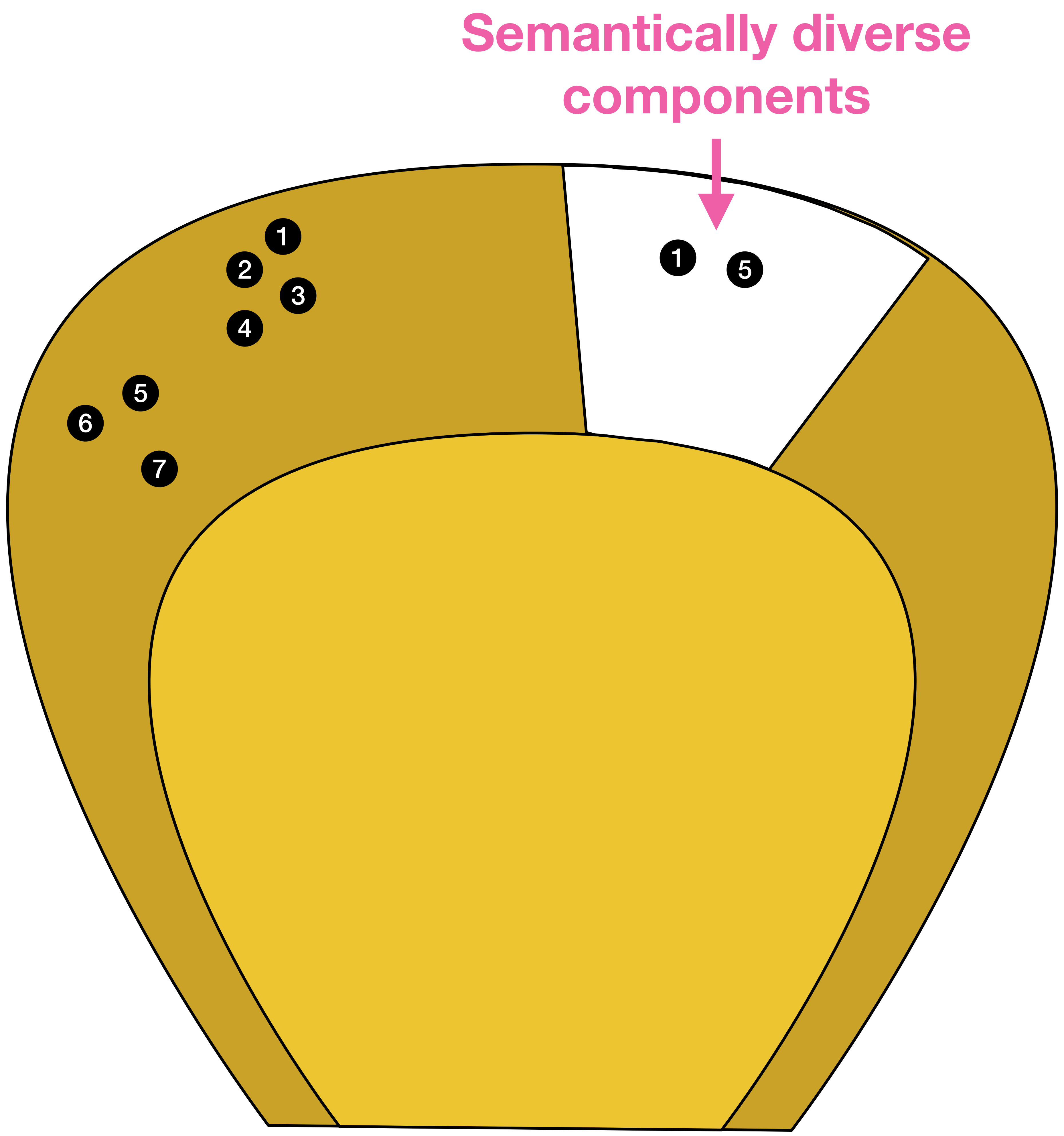
- Beam search
  - Bias the search w.r.t. operator distribution
  - Eliminate low-frequency operators
- How to define the beam?
- *Semantic condensation*
  - Classify candidates using examples
  - Pick top candidates from each class



+	12345:6789	123
-	:12	abc

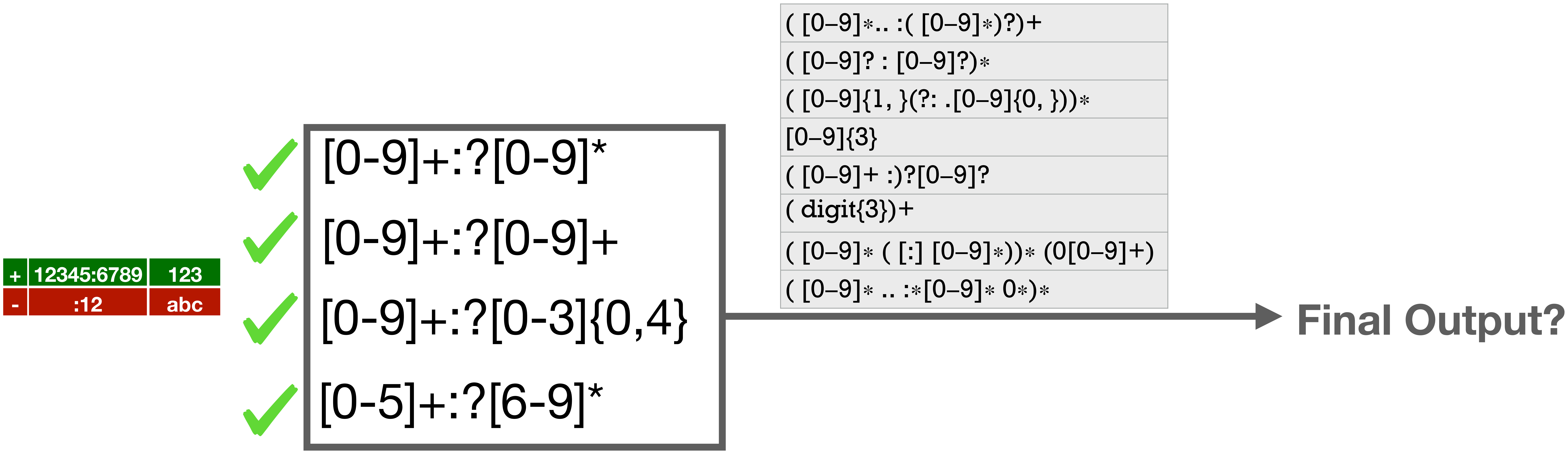
ITERATIVE EXPANSION

- Beam search
  - Bias the search w.r.t. operator distribution
  - Eliminate low-frequency operators
- How to define the beam?
- *Semantic condensation*
  - Classify candidates using examples
  - Pick top candidates from each class



FINAL RANKING

- A large number of programs which satisfy the examples



+	12345:6789	123
-	:12	abc

✓ [0-9]+:?[0-9]*

✓ [0-9]+:?[0-9]+

✓ [0-9]+:?[0-3]{0,4}

✓ [0-5]+:?[6-9]*

([0–9]∗.. :([0–9]∗)?)+

([0–9]? : [0–9]?)∗

([0–9]{1, }(?: .[0–9]{0, }))∗

[0–9]{3}

([0–9]+ :)?[0–9]?

(digit{3})+

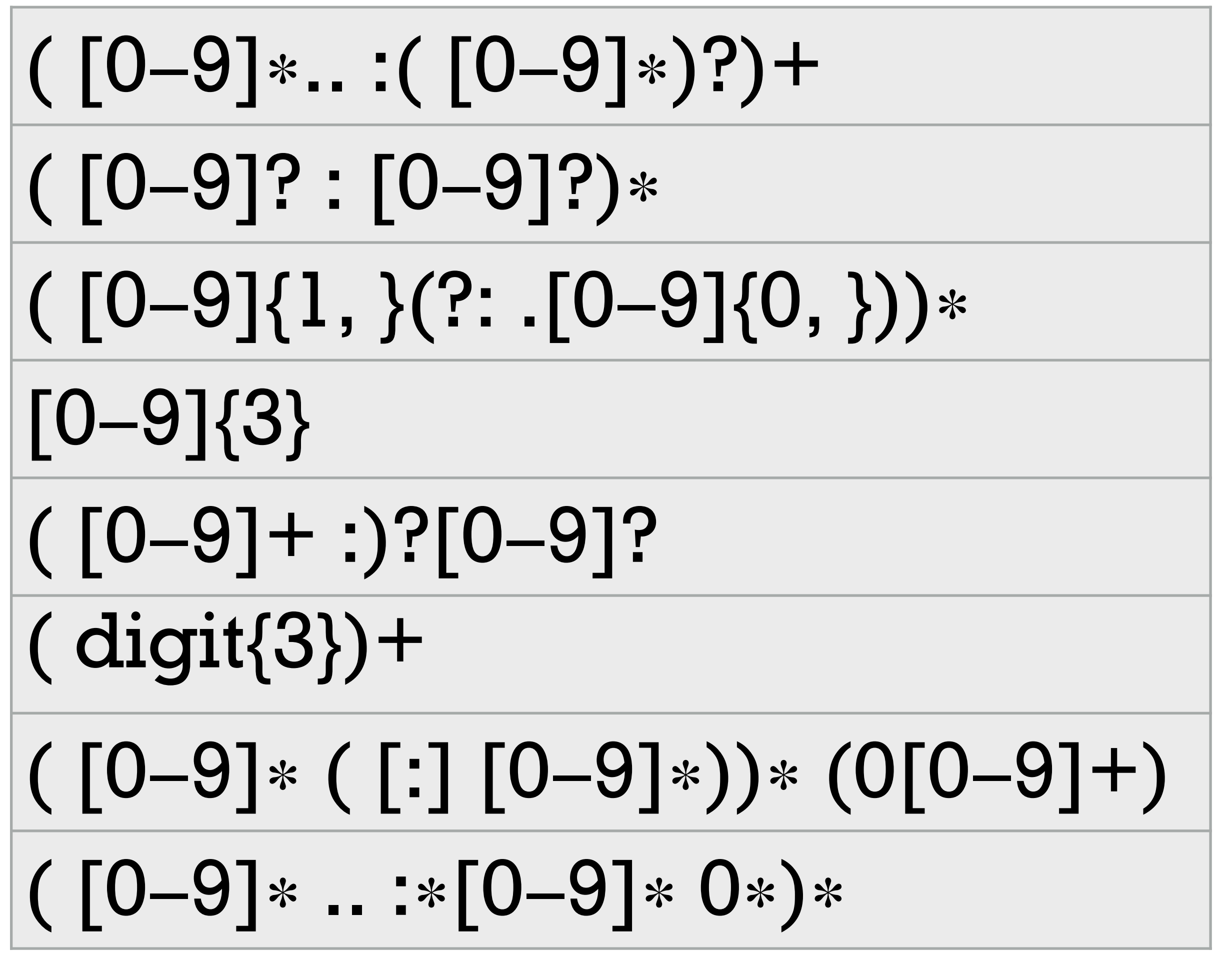
([0–9]∗ ([:] [0–9]∗))∗ (0[0–9]+)

([0–9]∗ .. :∗[0–9]∗ 0∗)∗

Final Output?

FINAL RANKING

- A large number of programs which satisfy the examples
  - Euclidean distance
  - Levenshtein distance



([0–9]*.. :([0–9]*)?)+
([0–9]? : [0–9]?)*
([0–9]{1, }(?: .[0–9]{0, }))*
[0–9]{3}
([0–9]+ :)?[0–9]?
(digit{3})+
([0–9]* ([:] [0–9]*))* (0[0–9]+)
([0–9]* .. :*[0–9]* 0*)*

[0-9]+:?[0-9]*

[0-9]+:?[0-9]+

[0-9]+:?[0-3]{0,4}

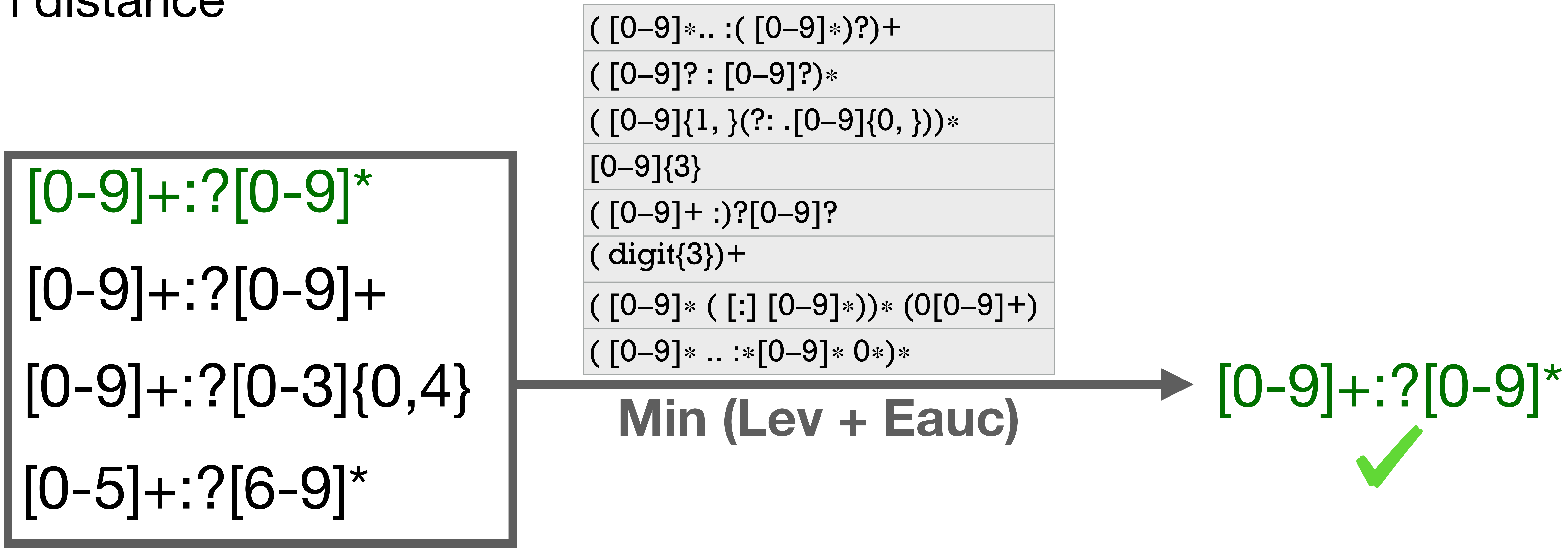
[0-5]+:?[6-9]*

Min (Lev + Eauc)

Final Output?

FINAL RANKING

- A large number of programs which satisfy the examples
  - Euclidean distance
  - Levenshtein distance



([0–9]∗.. :([0–9]∗)?)+

([0–9]? : [0–9]?)∗

([0–9]{1, }(?: .[0–9]{0, }))∗

[0–9]{3}

([0–9]+ :)?[0–9]?

(digit{3})+

([0–9]∗ ([:] [0–9]∗))∗ (0[0–9]+)

([0–9]∗ .. :∗[0–9]∗ 0∗)∗

[0-9]+:?[0-9]*

[0-9]+:?[0-9]+

[0-9]+:?[0-3]{0,4}

[0-5]+:?[6-9]*

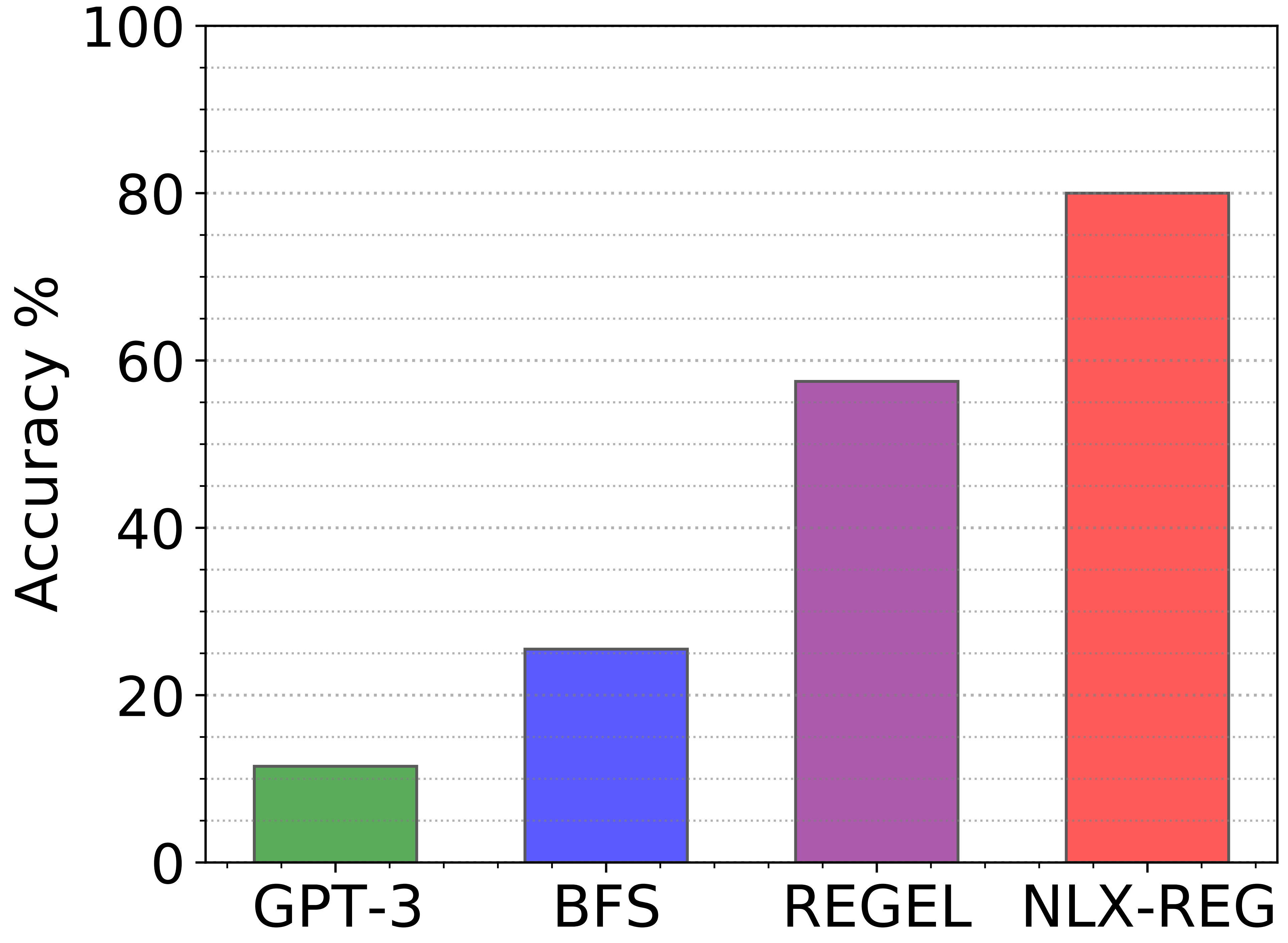
Min (Lev + Eauc)

[0-9]+:?[0-9]* ✓

EMPIRICAL RESULTS

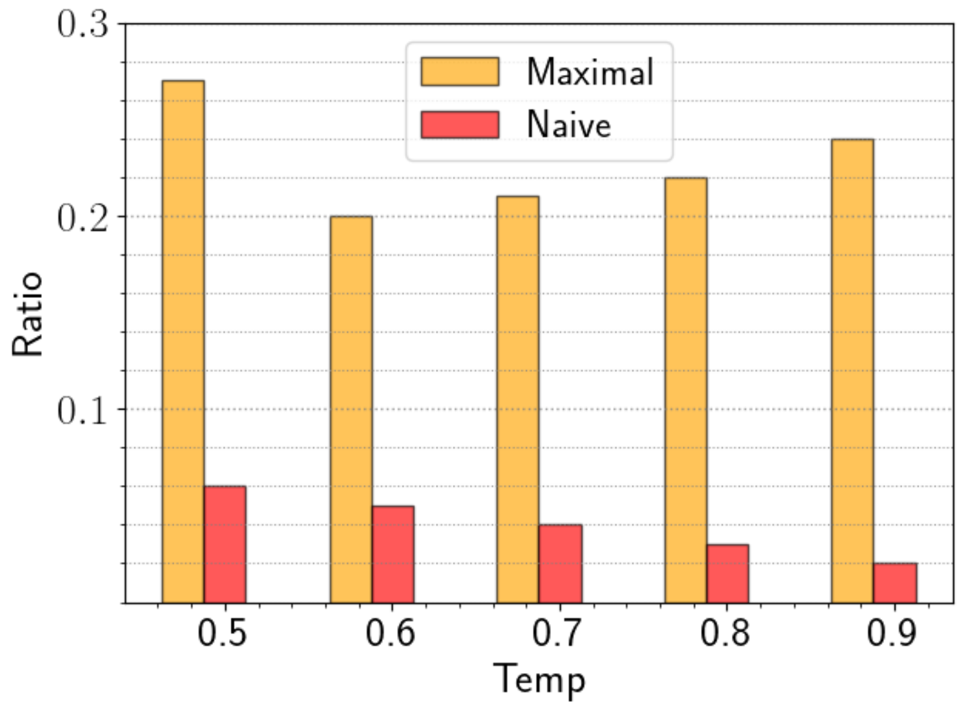
EXPERIMENTAL EVALUATION

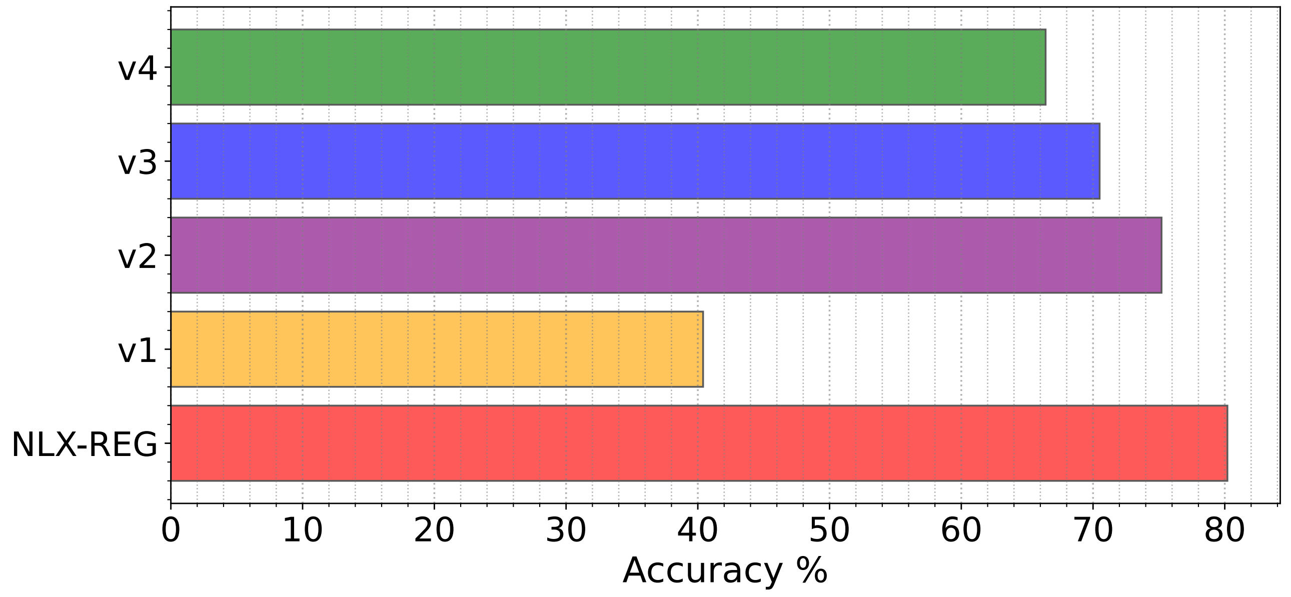
- Two Data sets
  - StackOverflow: 25 tasks
  - Previous work: 125 tasks
- NLX-REG outperforms the state-of-the-art

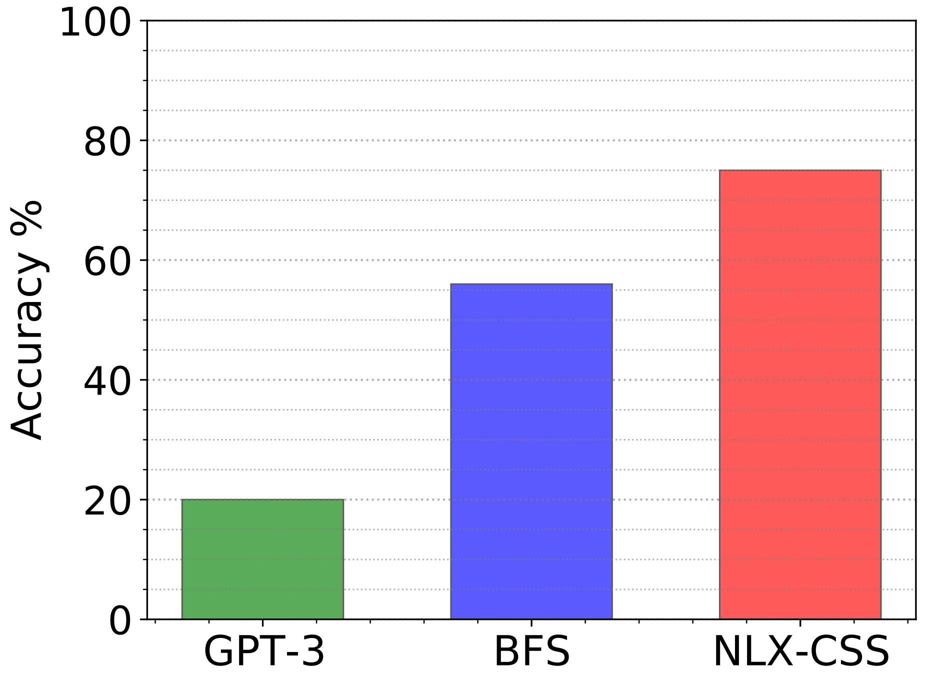


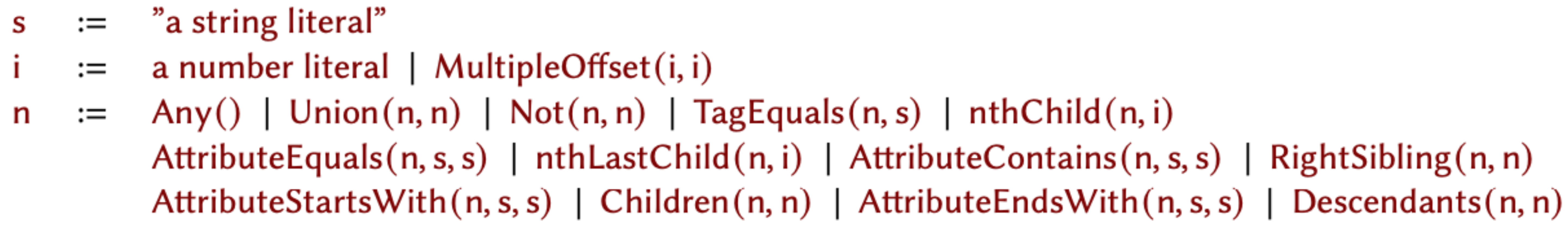
READ MORE!

- Ablation Study
- Domain of CSS selector
- Optimized use of the PTM



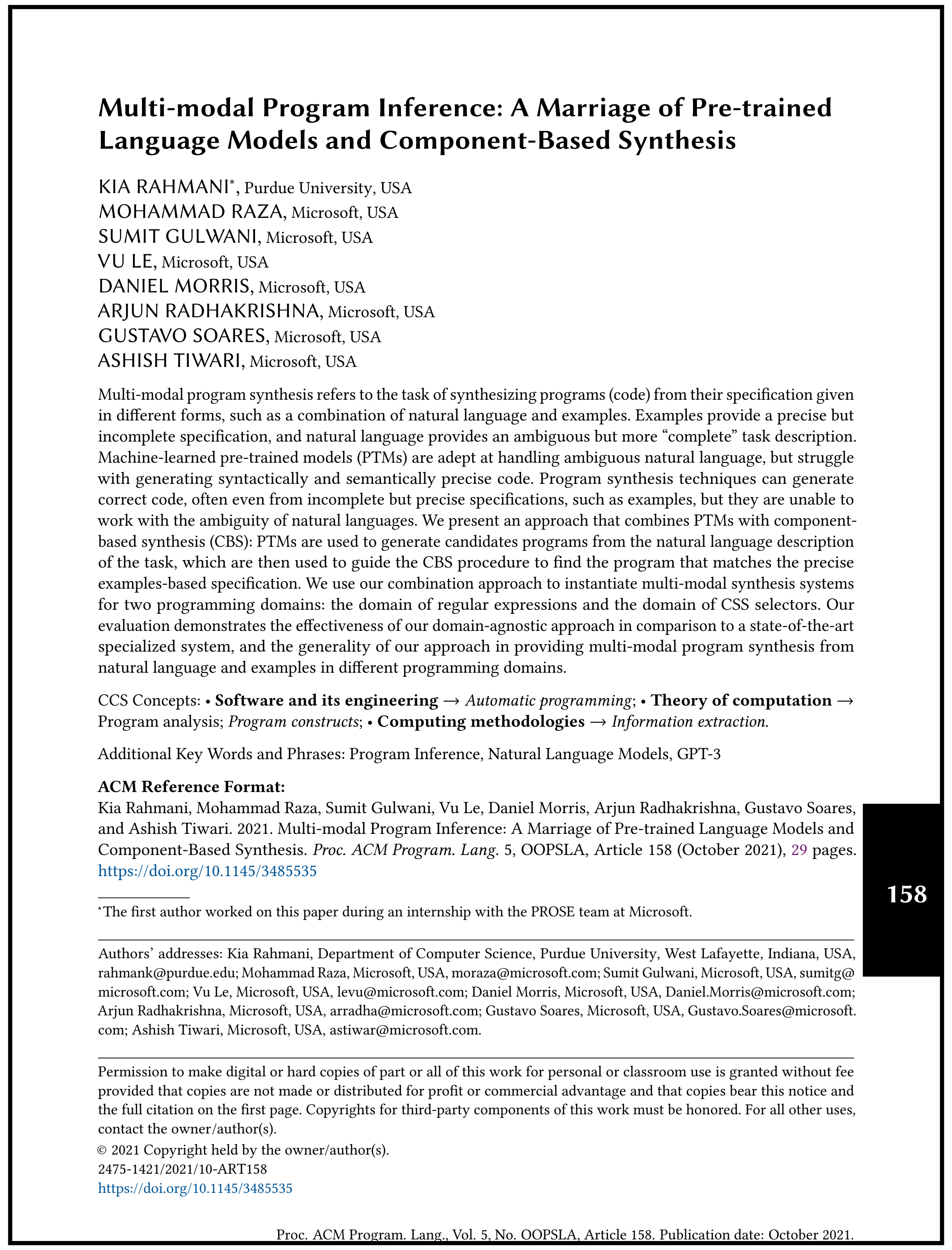






```
s  :=  "a string literal"
i  :=  a number literal | MultipleOffset(i, i)
n  :=  Any() | Union(n, n) | Not(n, n) | TagEquals(n, s) | nthChild(n, i)
       AttributeEquals(n, s, s) | nthLastChild(n, i) | AttributeContains(n, s, s) | RightSibling(n, n)
       AttributeStartsWith(n, s, s) | Children(n, n) | AttributeEndsWith(n, s, s) | Descendants(n, n)
```

Fig. 3. The DSL $\mathcal{L}_{\text{css}}$ of CSS expressions.



Multi-modal Program Inference: A Marriage of Pre-trained Language Models and Component-Based Synthesis

KIA RAHMANI[*], Purdue University, USA
MOHAMMAD RAZA, Microsoft, USA
SUMIT GULWANI, Microsoft, USA
VU LE, Microsoft, USA
DANIEL MORRIS, Microsoft, USA
ARJUN RADHAKRISHNA, Microsoft, USA
GUSTAVO SOARES, Microsoft, USA
ASHISH TIWARI, Microsoft, USA

Multi-modal program synthesis refers to the task of synthesizing programs (code) from their specification given in different forms, such as a combination of natural language and examples. Examples provide a precise but incomplete specification, and natural language provides an ambiguous but more "complete" task description. Machine-learned pre-trained models (PTMs) are adept at handling ambiguous natural language, but struggle with generating syntactically and semantically precise code. Program synthesis techniques can generate correct code, often even from incomplete but precise specifications, such as examples, but they are unable to work with the ambiguity of natural languages. We present an approach that combines PTMs with component-based synthesis (CBS): PTMs are used to generate candidates programs from the natural language description of the task, which are then used to guide the CBS procedure to find the program that matches the precise examples-based specification. We use our combination approach to instantiate multi-modal synthesis systems for two programming domains: the domain of regular expressions and the domain of CSS selectors. Our evaluation demonstrates the effectiveness of our domain-agnostic approach in comparison to a state-of-the-art specialized system, and the generality of our approach in providing multi-modal program synthesis from natural language and examples in different programming domains.

CCS Concepts: • **Software and its engineering** → *Automatic programming*; • **Theory of computation** → Program analysis; *Program constructs*; • **Computing methodologies** → *Information extraction.*

Additional Key Words and Phrases: Program Inference, Natural Language Models, GPT-3

ACM Reference Format:
Kia Rahmani, Mohammad Raza, Sumit Gulwani, Vu Le, Daniel Morris, Arjun Radhakrishna, Gustavo Soares, and Ashish Tiwari. 2021. Multi-modal Program Inference: A Marriage of Pre-trained Language Models and Component-Based Synthesis. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 158 (October 2021), 29 pages. https://doi.org/10.1145/3485535

158

[*] The first author worked on this paper during an internship with the PROSE team at Microsoft.

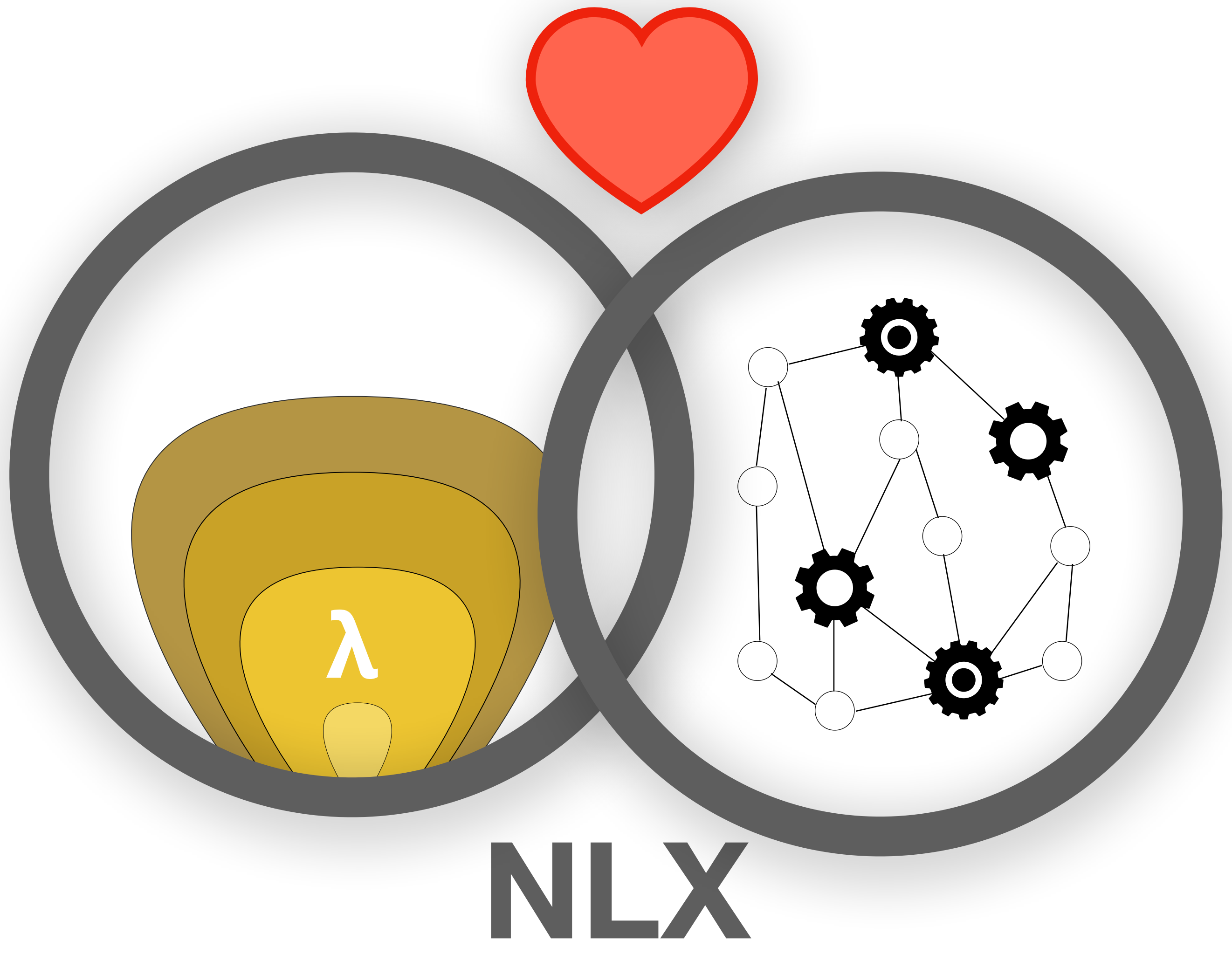
Authors' addresses: Kia Rahmani, Department of Computer Science, Purdue University, West Lafayette, Indiana, USA, rahmank@purdue.edu; Mohammad Raza, Microsoft, USA, moraza@microsoft.com; Sumit Gulwani, Microsoft, USA, sumitg@microsoft.com; Vu Le, Microsoft, USA, levu@microsoft.com; Daniel Morris, Microsoft, USA, Daniel.Morris@microsoft.com; Arjun Radhakrishna, Microsoft, USA, arradha@microsoft.com; Gustavo Soares, Microsoft, USA, Gustavo.Soares@microsoft.com; Ashish Tiwari, Microsoft, USA, astiwar@microsoft.com.

2475-1421/2021/10-ART158
https://doi.org/10.1145/3485535

https://doi.org/10.1145/3485535

RECAP

- PTM: “Rise of AI Language Models in Programming Automation”
- Multi-modal -> precision
- NLX: component-based synthesis based on results generated from a PTM
    - Domain Agnostic (REGEX and CSS selectors)
- Other domains + general purpose programming



ACKNOWLEDGMENT

microsoft.com/research/group/prose

	A	B	C
1	DEC	December	
2	NOV	November	
3	OCT	Octember	
4	APR	Aprember	
5	AUG	Augember	
6	FEB	Febember	
7	JAN	Janember	
8	JUL	Julember	
9	JUN	Junember	
10	MAR	Marember	
11	MAY	Mayember	
12	SEP	Sepember	
13			

"AI is going to take over the world..."

Microsoft

PROSE TEAM

Mohammad Raza	(moraza@microsoft.com)
Sumit Gulwani	(sumitg@microsoft.com)
Ashish Tiwari	(astiwar@microsoft.com)
Gustavo Soares	(Gustavo.Soares@microsoft.com)
Arjun Radhakrishna	(arradha@microsoft.com)
Daniel Morris	(Daniel.Morris@microsoft.com)
Vu Le	(levu@microsoft.com)

REFERENCES (DOI):

[1] 10.1145/3385412.3385988
[2] 10.18653/v1/D16-1197

THANKS FOR YOUR ATTENTION!