Grammars for Free

Toward Grammar Inference for Ad Hoc Parsers

https://arxiv.org/abs/2202.01021

Michael Schröder
TU Wien
Vienna, Austria
michael.schroeder@tuwien.ac.at

Jürgen Cito
TU Wien and Meta Platforms, Inc.
Vienna, Austria
juergen.cito@tuwien.ac.at

New Ideas and Emerging Results, ICSE 2022
Pittsburgh, PA, USA

TU WIEN Informatics

psf/requests > requests/utils.py 47 matches Python main

```python
    """
    if string_network.count('/') == 1:
        try:
            mask = int(string_network.split('/')[1])
        except ValueError:
            return False

```

cortinico/react-native > Libraries/Image/RCTImageCache.m 2 matches Objective-C main

```objc
    } else {
      NSRange range = [component rangeOfString:@"max-age="];
      if (range.location != NSNotFound) {
        NSInteger seconds = [[component substringFromIndex:range.location + range.length] integerValue];
        staleTime = [originalDate dateByAddingTimeInterval:(NSTimeInterval)seconds];
      }
    }
```

vuejs/vuex > src/plugins/devtool.js 6 matches JavaScript main

```javascript
 * @param {string} path
 */
function extractNameFromPath (path) {
  return path && path !== 'root' ? path.split('/').slice(-2, -1)[0] : 'Root'
}

/**
```

x10an14/docs > javascripts/user-agent.ts 6 matches TypeScript main

```typescript

export default function parseUserAgent(ua = navigator.userAgent) {
  ua = ua.toLowerCase()
  const osRe = OS_REGEXPS.find((re) => re.test(ua))
  let [, os = 'other', os_version = '0'] = (osRe && ua.match(osRe)) || []
  if (os === 'iphone os' || os === 'ipad os') os = 'ios'
  const browserRe = BROWSER_REGEXPS.find((re) => re.test(ua))
```

Parsers are everywhere!

coolreader18/libqalculate > src/qalc.cc 86 matches C++ master

```cpp
			while(str[i] == '\033') {
				do {
					i++;
				} while(i < str.length() && str[i] != 'm');
				i++;
				if(i >= str.length()) break;
			}
```

django/django > django/utils/html.py 1 match Python main

```python
        if "@" not in value or value.startswith("@") or value.endswith("@"):
            return False
        try:
            p1, p2 = value.split("@")
        except ValueError:
            # value contains more than one @.
            return False
```

packages/GitPython/lib/git/actor.py 3 matches Python

```python
            Actor
        """
        if re.search(r'<.+>', string):
            m = re.search(r'(.*) <(.+?)>', string)
            name, email = m.groups()
            return Actor(name, email)
        else:
```

fixtures/dom/src/components/IssueList.js 1 match JavaScript

```javascript
const React = window.React;

function csv(string) {
  return string.split(/\s*,\s*/);
}

export default function IssueList({issues}) {
```

"1,2,3"

```
xs = map(int, s.split(","))
```

[1,2,3]

""

```
xs = map(int, s.split(","))
```

"+01_2,3_0_4 "

```
xs = map(int, s.split(","))
```

[12,304]

class **int**([*x*])
class **int**(*x*, *base=10*)

Return an integer object constructed from a number or string *x*, or return `0` if no argumen given. If *x* defines `__int__()`, `int(x)` returns `x.__int__()`. If *x* defines `__index__()`, it r `x.__index__()`. If *x* defines `__trunc__()`, it returns `x.__trunc__()`. For floating point nu this truncates towards zero.

If *x* is not a number or if *base* is given, then *x* must be a string, `bytes`, or `bytearray` insta resenting an integer literal in radix *base*. Optionally, the literal can be preceded by `+` or `-` space in between) and surrounded by whitespace. A base-n literal consists of the digits 0 with `a` to `z` (or `A` to `Z`) having values 10 to 35. The default *base* is 10. The allowed values and 2-36. Base-2, -8, and -16 literals can be optionally prefixed with `0b`/`0B`, `0o`/`0O`, or `0` with integer literals in code. Base 0 means to interpret exactly as a code literal, so that the

map(*function*, *iterable*, ...

Return an iterator that ap *iterable* arguments are passed, *function* must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted. For cases where the function inputs are already arranged into argument tuples, see `itertools.starmap()`.

`str.`**split**(*sep=None*, *maxsplit=-1*)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.

For example:

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
```

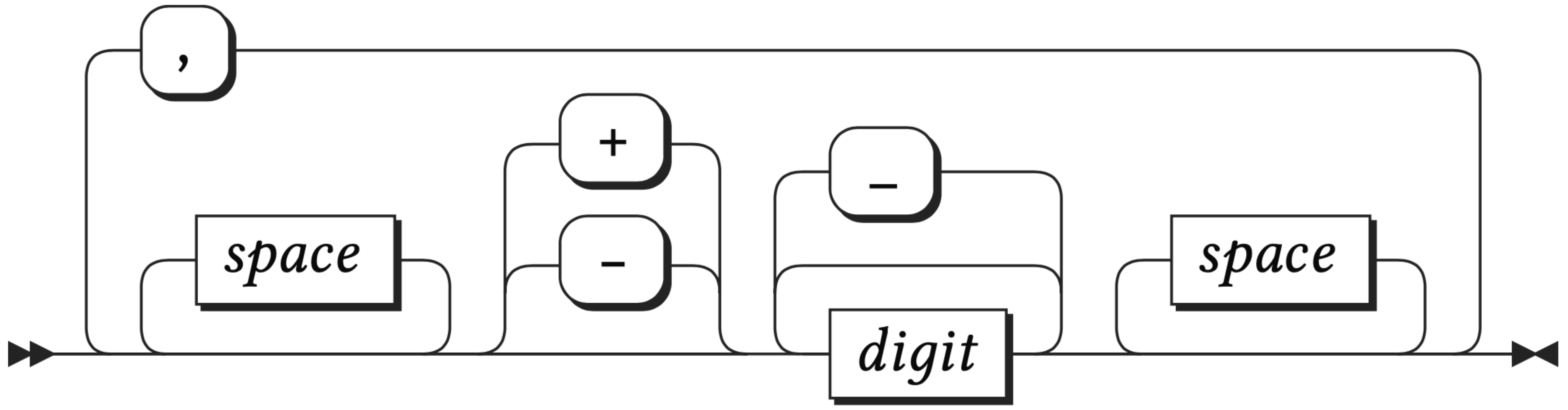
```
xs = map(int, s.split(","))
```

"1,2,3" → [1,2,3] ✅

"+01_2,3_0_4 " → [12,304] ✅

"" → ❌

⋮



$$
\begin{aligned}
s &\rightarrow \mathit{int} \mid \mathit{int}\ \texttt{,}\ s \\
\mathit{int} &\rightarrow \mathit{space}^{*}\ \mathit{sign}^{?}\ \mathit{digit}\ (\texttt{\_}^{?}\ \mathit{digit})^{*}\ \mathit{space}^{*} \\
\mathit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\mathit{sign} &\rightarrow \texttt{+} \mid \texttt{-} \\
\mathit{space} &\rightarrow \textvisiblespace \mid \texttt{\textbackslash t} \mid \texttt{\textbackslash n} \mid \texttt{\textbackslash v} \mid \texttt{\textbackslash f} \mid \texttt{\textbackslash r}
\end{aligned}
$$

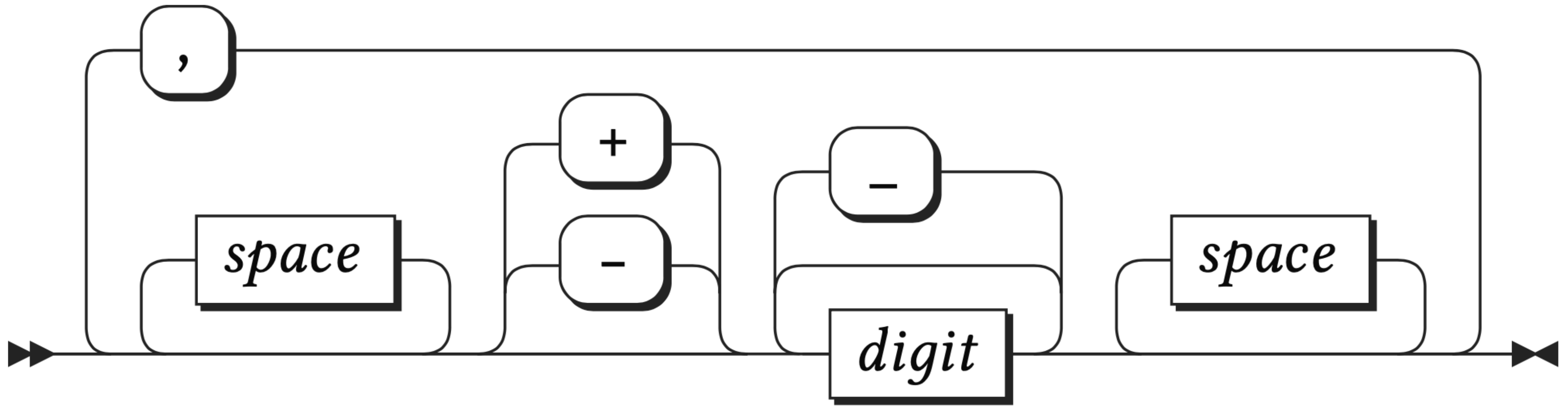
```
xs = map(int, s.split(","))
```

```
"1,2,3"          →   [1,2,3]      ✅
"+01_2,3_0_4 "   →   [12,304]     ✅
""               →   ❌
                 ⋮
```

The Need for Grammars

- Program Comprehension & Documentation
  - high-level data-centric perspective
  - diverse notations: regex, ABNF, PEG, state machine, railroad diagram,…
- Fuzzing
  - systematically generate random test inputs
  - grammar-based fuzzers can penetrate into deep program states (past syntactic checks)
- Language-Theoretic Reasoning
  - safety of parsers is connected to theoretic properties (e.g., computability bounds)
  - input languages should be minimally powerful
  - https://langsec.org

> [LANGSEC] regards the Internet insecurity epidemic as a consequence of *ad hoc* programming of input handling […]



$$\begin{aligned}
s &\rightarrow \mathit{int} \mid \mathit{int}\ \texttt{,}\ s \\
\mathit{int} &\rightarrow \mathit{space}^*\ \mathit{sign}^?\ \mathit{digit}\ (\texttt{_}^?\ \mathit{digit})^*\ \mathit{space}^* \\
\mathit{digit} &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\mathit{sign} &\rightarrow \texttt{+} \mid \texttt{-} \\
\mathit{space} &\rightarrow \textvisiblespace \mid \texttt{\textbackslash t} \mid \texttt{\textbackslash n} \mid \texttt{\textbackslash v} \mid \texttt{\textbackslash f} \mid \texttt{\textbackslash r}
\end{aligned}$$

```
xs = map(int, s.split(","))
```

Parser : Grammar ≈ Function : Type

Type Inference

String

```
xs = map(int, s.split(","))
```

[Int]

✨ Grammar Inference ✨

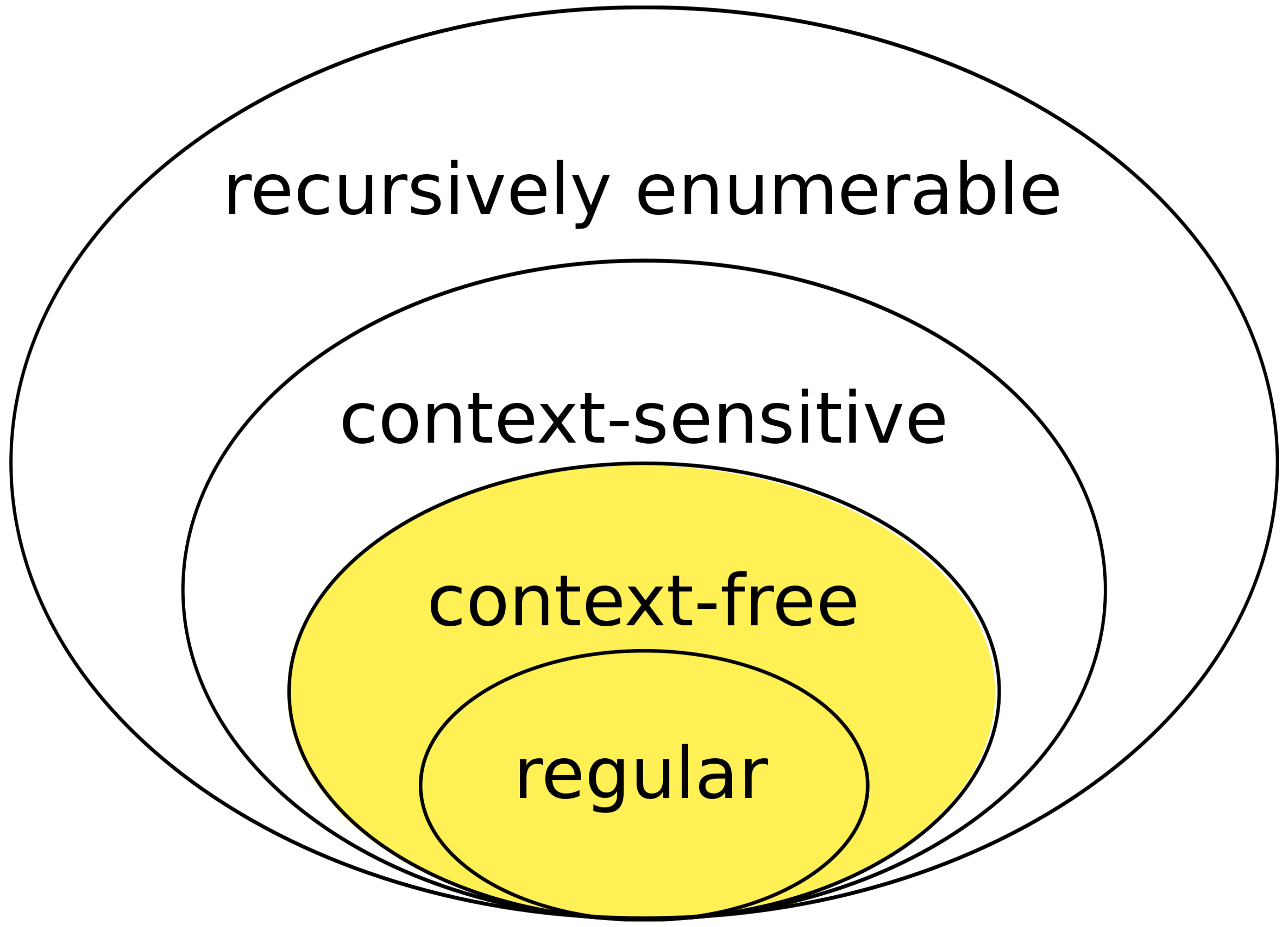
$$
\begin{aligned}
s &\rightarrow int \mid int\ \texttt{,}\ s \\
int &\rightarrow space^*\ sign^?\ digit\ (\_^?\ digit)^*\ space^* \\
digit &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
sign &\rightarrow \texttt{+} \mid \texttt{-} \\
space &\rightarrow \text{␣} \mid \texttt{\textbackslash t} \mid \texttt{\textbackslash n} \mid \texttt{\textbackslash v} \mid \texttt{\textbackslash f} \mid \texttt{\textbackslash r}
\end{aligned}
$$

String {•}

```
xs = map(int, s.split(","))
```

[Int]

Toward Grammar Inference



The Chomsky hierarchy

$$L = \{a^n b^n \mid n > 0\}$$

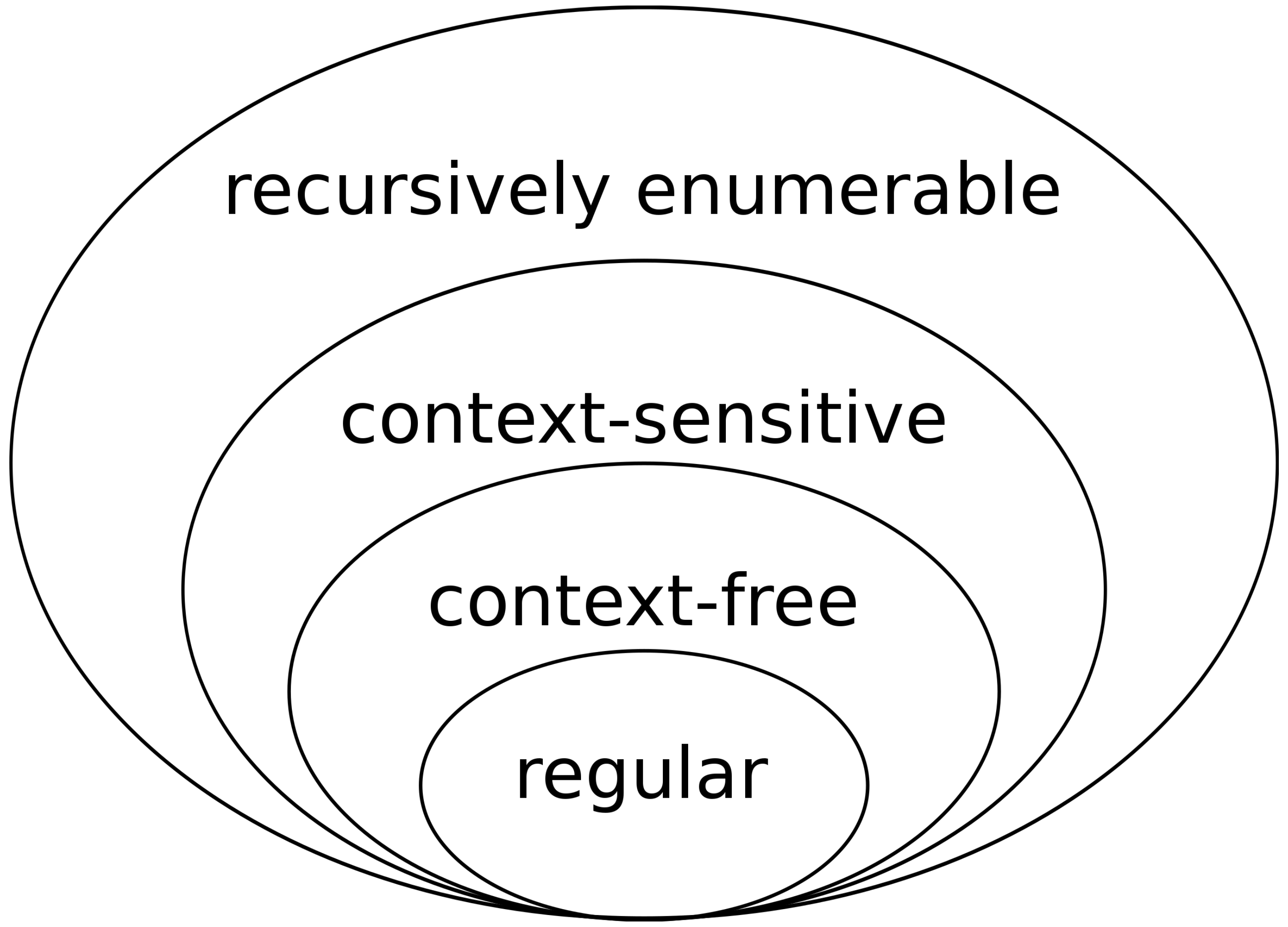
Language

Grammar

$$S \rightarrow aSb$$
$$S \rightarrow \varepsilon$$

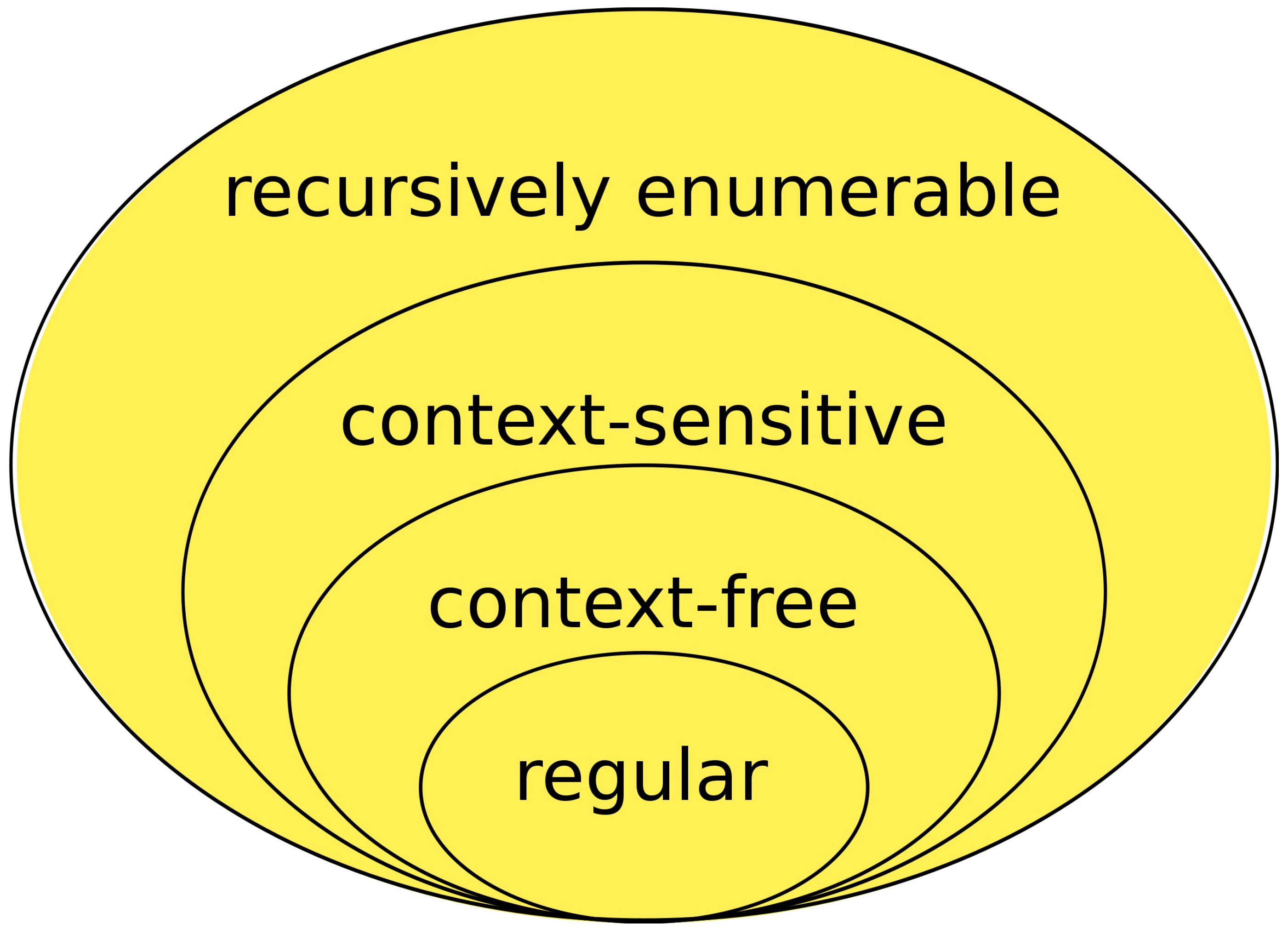
Machine

```
i = 0
while s[i] == ‘a':
  i = i + 1
n = i
while i < len(s):
  assert s[i] == 'b'
  i = i + 1
  n = n - 1
assert n == 0
```



The Chomsky hierarchy

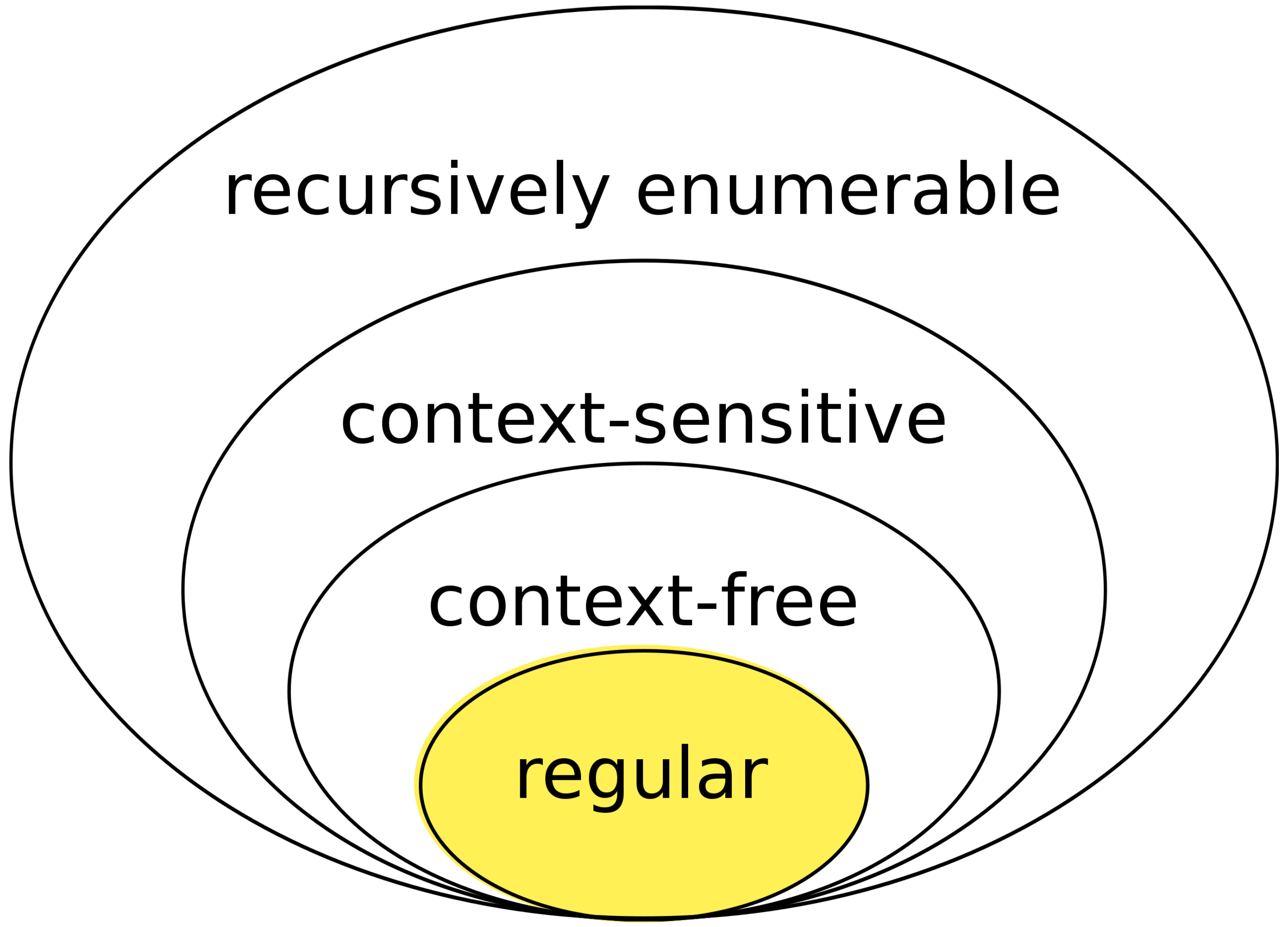
```python
import math

while True:
  line = input("enter 3d vector: ")
  xs = line.split(",")
  if len(xs) == 3: break
  else: print("please try again.")

[x,y,z] = map(int, xs)
n = math.sqrt(x**2 + y**2 + z**2)
print(f"vector length: {n}")
```



The Chomsky hierarchy

```
import math

while True:
  line = input("enter 3d vector: ")
  xs = line.split(",")
  if len(xs) == 3: break
  else: print("please try again.")

[x,y,z] = map(int, xs)
n = math.sqrt(x**2 + y**2 + z**2)
print(f"vector length: {n}")
```

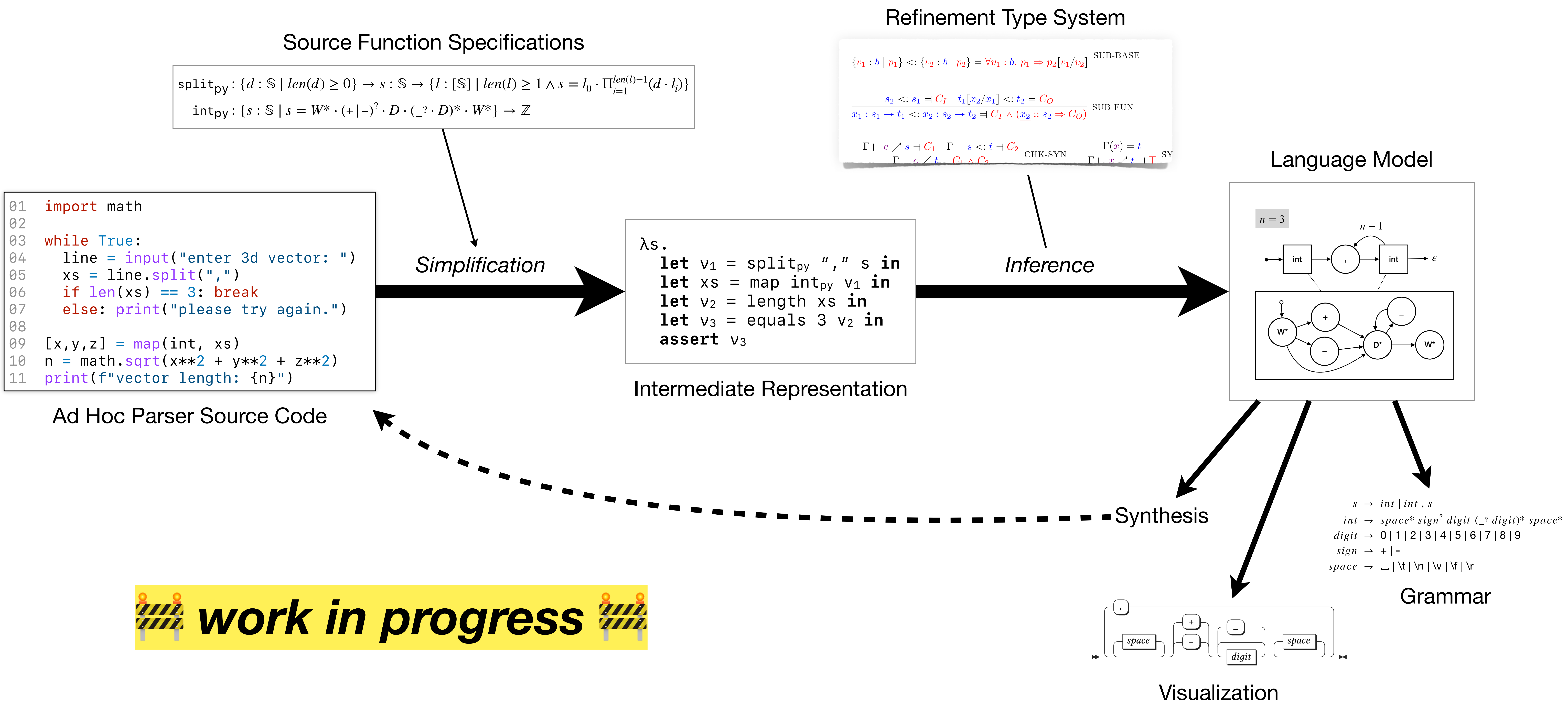
Intuition: Parsers are Embedded Machines



The Chomsky hierarchy

```python
import math

while True:
  line = input("enter 3d vector: ")
  xs = line.split(",")
  if len(xs) == 3: break
  else: print("please try again.")

[x,y,z] = map(int, xs)
n = math.sqrt(x**2 + y**2 + z**2)
print(f"vector length: {n}")
```

Vision: Automatic Grammar Inference



🚧 ***work in progress*** 🚧

New Possibilities

Interactive Documentation

NEW

- inferred grammar is always up-to-date
- closely linked to underlying source code

Inferred grammar:

$$
\begin{aligned}
\textsf{pred} &\rightarrow a = v_1 \mid a\ o\ v_2 \\
a &\rightarrow (\Sigma \backslash o)^* \\
v_1 &\rightarrow \Sigma^* \\
v_2 &\rightarrow \textsf{eval} \\
o &\rightarrow \textsf{<} \mid \textsf{<=} \mid \textsf{>} \mid \textsf{>=}
\end{aligned}
$$

```
def pred(s):
  ops = r"(=|<|<=|>|>=)"
  a,o,v = re.split(ops, s)
  if o[0] == "<" or o[0] == ">":
    v = eval(v)
  return p(a,o,v)
```

Inferred grammar:

$$
\begin{aligned}
\textsf{pred} &\rightarrow a = v_1 \mid a\ o\ v_2 \\
a &\rightarrow (\Sigma \backslash o)^* \\
v_1 &\rightarrow \Sigma^* \\
v_2 &\rightarrow \textsf{eval} \\
o &\rightarrow \textsf{<} \mid \textsf{<=} \mid \textsf{>} \mid \textsf{>=}
\end{aligned}
$$

```
def pred(s):
  ops = r"(=|<|<=|>|>=)"
  a,o,v = re.split(ops, s)
  if o[0] == "<" or o[0] == ">":
    v = eval(v)
  return p(a,o,v)
```

Inferred grammar:

$$
\begin{aligned}
\textsf{pred} &\rightarrow a = v_1 \mid a\ o\ v_2 \\
a &\rightarrow (\Sigma \backslash o)^* \\
v_1 &\rightarrow \Sigma^* \\
v_2 &\rightarrow \textsf{eval} \\
o &\rightarrow \textsf{<} \mid \textsf{<=} \mid \textsf{>} \mid \textsf{>=}
\end{aligned}
$$

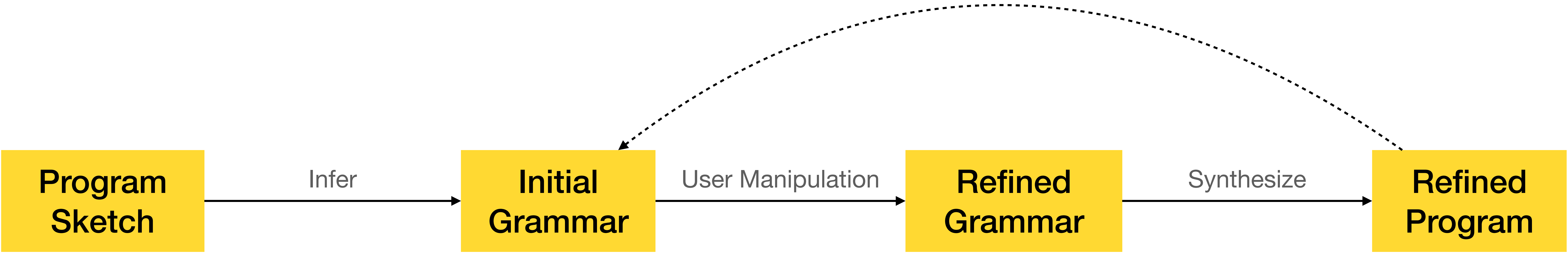
```
def pred(s):
  ops = r"(=|<|<=|>|>=)"
  a,o,v = re.split(ops, s)
  if o[0] == "<" or o[0] == ">":
    v = eval(v)
  return p(a,o,v)
```

Bi-Directional Parser Synthesis

NEW

- semantic program transformation via inferred grammar

  → program sketching




```python
def foo(s):
  i = 0
  while s[i] == "a"
    i += 1
```

foo $\rightarrow a^*(\Sigma \backslash a)\Sigma^*$

foo $\rightarrow a^*$

```python
def foo(s):
  i = 0
  while i < len(s):
    assert s[i] == "a"
```

Mining & Learning

NEW

- grammar as *equivalence class* over concrete implementations
  - → detecting code clones of ad hoc parsers
  - → grammar-enhanced semantic code search

s -> int | int , s

Viewer.py — 2 matches | Python

```
    ranges = self.find_ranges()
    split = str.split
    point = map(int, split(self.text.index(CURRENT), ','))
    for start, end in ranges:
      startv = map(int, split(start, ','))
```

transformScriptTags.ts — 1 match | TypeScript

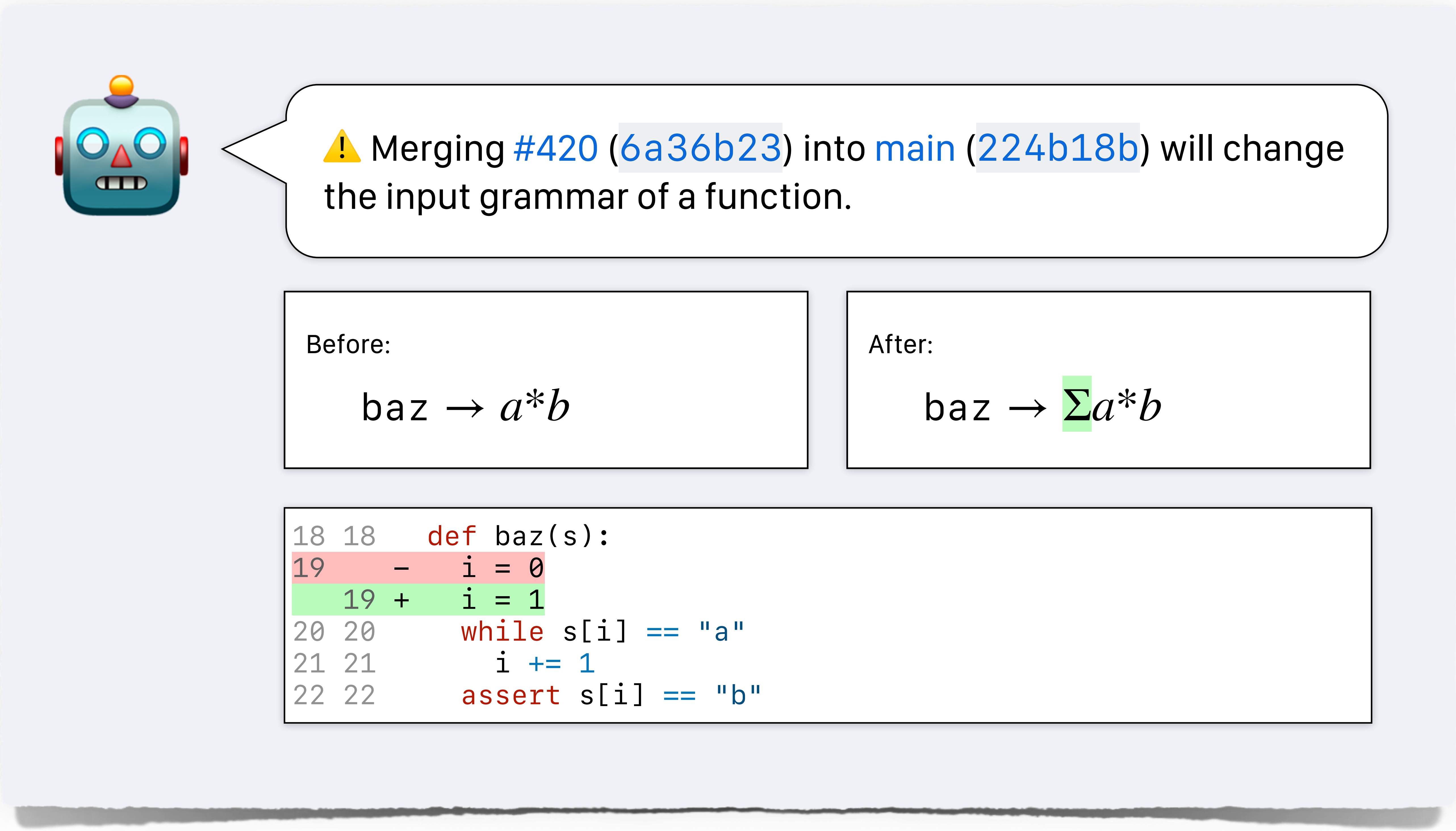
```
      return null;
    }
    return rawValue.split(",").map(item => parseInt(item));
  }

```

Mining & Learning

NEW

- learn how implicit input specifications evolve over time

   → grammar-aware semantic change tracking



Future Plans

- 🕵️ mining study of ad hoc parsers in the wild — ongoing
- 🔮 grammar inference via refinement types of simplified parsing IR — ongoing
- ∀x proving soundness of simplification and inference steps — tbd
- 🎯 evaluation on corpus of curated ad hoc parser samples — tbd
- 🤖 large-scale mining study of inferred grammars — tbd
- 🙆 user studies on grammar comprehension — tbd

Grammars for Free

Toward Grammar Inference for Ad Hoc Parsers

https://arxiv.org/abs/2202.01021

Michael Schröder
TU Wien
Vienna, Austria
michael.schroeder@tuwien.ac.at

Jürgen Cito
TU Wien and Meta Platforms, Inc.
Vienna, Austria
juergen.cito@tuwien.ac.at

New Ideas and Emerging Results, ICSE 2022
Pittsburgh, PA, USA

TU WIEN Informatics